UNIVERSITY OF APPLIED SCIENCES

FHDW

FACHHOCHSCHULE FÜR DIE WIRTSCHAFT
HANNOVER

Integrationsprojekt

FHDW OS

–

Handbuch

Studiengang Praktische Informatik
HFI410

Patrick Anke, Andreas Görzen, Jonas Kemper, Benjamin Kotke, Wolfgang Laut,
Michael Lierath, Timo Lorenz, Fabian Mierendorff, Marius Schultchen,
Sascha Sternheim, Sebastian Wolf

Betreuender Dozent:
Dr. Christoph Schulz

2. Oktober 2013

Das vorliegende Handbuch beschreibt die Grundlagen des Betriebssystems FHDW-OS. Dieses Betriebssystem ist im 5. und 6. Semester des Bachelor-Studiengangs Praktische Informatik an der Fachhochschule für die Wirtschaft in Hannover entstanden. Das Betriebssystem wurde auf Grundlage der IA-32-Architektur von Intel entwickelt. Die folgenden Kapitel beschreiben im Detail die Funktionen des FHDW-OS, so z. B. die Speicherverwaltung, das Konzept zu Interrupts und Ausnahmen und die Task-Verwaltung. Die einzelnen Teile des Dokuments sind dabei meist so strukturiert, dass auf die Analyse der IA-32-Architektur der Entwurf und die Besonderheiten der Implementierung folgen. Der Teil VI (Allgemeines) am Ende des Dokuments stellt hier eine Ausnahme dar. In diesem Teil sind Themen aufgeführt, die nur indirekt in die Entwicklung des Betriebssystems eingeflossen sind. So sind dort z. B. eine Übersicht über die Unterschiede von C++ zu Java, aber auch die Erstellung des Handbuchs dokumentiert.

Im Anhang findet der interessierte Leser die Dokumentation des C++-Quellcodes (siehe Kapitel A: Doxygen) sowie einen Index für das schnelle Nachschlagen von Stichwörtern. Das Dokument wurde für den Einsatz in PDF-Form optimiert und enthält daher sichtbare (farbig umrahmte) Querverweise.

Inhaltsverzeichnis

Abbildungsverzeichnis **xxiv**

Tabellenverzeichnis **xxvii**

Listings **xxix**

I Speicherverwaltung **1**

1 Speicherverwaltung der IA-32-Architektur **2**
1.1 Einleitung 2
1.2 Virtuelle Speicherverwaltung 2
1.3 Segmentierung 3
1.3.1 Funktionsweise der Segmentierung 3
1.3.2 Segmentierung im IA32 4
1.4 Paging 9
1.4.1 Paging Grundlagen 9
1.4.2 Paging im IA-32 10
1.4.3 Fazit 13

2 Speicher-Allokation **14**
2.1 Einleitung 14
2.2 Algorithmen 14
2.2.1 First-Fit-Algorithmus 14
2.2.2 Buddy-Algorithmus 15
2.2.3 Wahl eines Algorithmus 16
2.3 Grundinformationen 17
2.4 Allokator 17
2.4.1 Konstruktor 17
2.4.2 Destruktor 18
2.4.3 BuddyPageTable-Struktur 18
2.4.4 BuddyPageUsage-Klasse 18
2.4.5 Vorgehen beim Allozieren 20
2.4.6 Vorgehen beim Freigeben 22

2.4.7 Speicherausnutzung 23
2.5 Test des Allokators 24

3 Paging-Algorithmus 26
3.1 Datenstruktur zur physikalischen Speicherverwaltung 26
3.1.1 Übersicht möglicher Datenstrukturen 26
3.1.2 „physical memory management table(PmmTable)“ 28
3.2 Aufteilung des Adressraums 30
3.2.1 Lower Half Kernel 32
3.2.2 Higher Half Kernel 33
3.2.3 Umsetzung 34
3.3 Paging 35
3.3.1 Rekursives Paging Directory 36
3.3.2 Aufbau 36
3.3.3 Initialisierung des Pagings 40
3.3.4 Ablauf des Abbildens neuer Pages 40

4 Schutzkonzepte der IA-32-Architektur 42
4.1 Allgemein 42
4.2 Privilegstufen 42
4.2.1 Kommunikation zwischen Privilegstufe 43
4.2.2 Stack Wechsel 46
4.2.3 Call Gates 46
4.2.4 Zurückkehren nach einem Prozeduraufruf 47
4.2.5 Eingabe-Ausgabe Privilegstufen 48
4.3 Priviligierte Instruktionen 48
4.4 Limit-Prüfungen 49
4.5 Typ-Prüfung 49
4.6 Seitenschutz 50
4.6.1 Domäne und Typ einer Seite 50
4.6.2 Zusammenwirken der Schutzkonzepte 51

5 VFS+FAT 53
5.1 Grundlagen 53
5.1.1 Virtuelles Dateisystem 53
5.1.2 FAT16 55
5.2 Virtuelles Dateisystem – VFS 56
5.2.1 Analyse 56
5.2.2 Entwurf 57
5.2.3 Fazit 58

5.3 FAT16 58
5.3.1 Analyse 58
5.3.2 Entwurf 61

II Interrupts und Ausnahmen 65

6 Interrupts und Ausnahmen – Analyse 66
6.1 Allgemein 66
6.2 IDT 66
6.3 Unterbrechungen 67
6.3.1 Quellen 68
6.3.2 Externe Unterbrechungen 68
6.3.3 Software-generierte Unterbrechungen 69
6.4 Ausnahmen 69
6.4.1 Quellen 69
6.4.2 Ausnahmenklassifizierung 70
6.5 Parallel auftretende Unterbrechungen 71
6.6 Unterbrechungen und Ausnahmen Referenz 72

7 Interrupts und Ausnahmen – Entwurf 74
7.1 Analyse 74
7.2 Entwurf 75
7.2.1 isrManager 76
7.2.2 isrHandlerContainer 76
7.2.3 isrHandler 77
7.3 Besonderheiten der Implementierung 77

III Task-Verwaltung und Synchronisation 78

8 Synchronisation 79
8.1 Grundlagen 79
8.1.1 Kritischer Wettlauf (Race Condition) 79
8.1.2 Kritischer Abschnitt (Critical Region) 80
8.2 Verfahren zum wechselseitigen Ausschluss 80
8.2.1 Abschalten der Interrupts 80
8.2.2 Aktives Warten 81
8.2.3 Passives Warten 83
8.3 Synchronisationsprimitive 83
8.3.1 Compare-and-swap 83
8.3.2 Test-and-set 85

8.3.3 Fetch-and-add . . . 86
8.4 Entwurf . . . 87
8.4.1 Abhängigkeiten . . . 88
8.4.2 Mutex . . . 88
8.4.3 Semaphore . . . 90

9 Task-Verwaltung 93
9.1 Analyse . . . 93
9.1.1 Task State Segment . . . 93
9.1.2 Task-Register . . . 93
9.1.3 Task-Gates . . . 95
9.1.4 Thread-Zustände . . . 95
9.1.5 Prozess vs. Thread . . . 96
9.1.6 Einordnung von Prozessen, Threads und Tasks . . . 96
9.1.7 Global Descriptor Table . . . 97
9.1.8 Dispatcher . . . 99
9.2 Entwurf . . . 99
9.2.1 Klasse „GDTManager“ . . . 99
9.2.2 Threads . . . 100
9.2.3 Scheduler . . . 101
9.2.4 Dispatcher . . . 103
9.3 Benutzung von Prozessen und Threads . . . 104
9.3.1 Threaderstellung . . . 104
9.3.2 Zusammenspiel mit dem Scheduler . . . 104
9.4 Fazit und Ausblick . . . 104

10 Interprozesskommunikation 107
10.1 Grundlagen . . . 107
10.1.1 Sicherheitsstufen eines Prozesses . . . 107
10.2 Analyse . . . 108
10.2.1 Aufgabe . . . 108
10.2.2 Aufteilung des Speichers . . . 108
10.2.3 Allozieren von Speicher . . . 109
10.2.4 Kommunikation zwischen zwei Prozessen . . . 109
10.2.5 Wechsel des Privileglevels . . . 110
10.3 Entwurf . . . 111
10.3.1 Allgemein . . . 111
10.3.2 Objektifizierung der Nachrichten . . . 112
10.3.3 Kommunikation über Privileglevel-Grenzen . . . 113
10.3.4 Interprozesskommunikation . . . 113

10.4 Implementierung . . . 116
10.4.1 Linear Memory Management . . . 117
10.4.2 Identifikation eines Prozesses . . . 118
10.4.3 Shared Memory Management . . . 118
10.4.4 Kommunikation zwischen Prozessen . . . 118
10.5 Tests . . . 119
10.6 Verwendung . . . 120
10.7 Fazit . . . 120
10.8 Ausblick . . . 121

11 Scheduler 122
11.1 Einleitung . . . 122
11.2 Analyse . . . 122
11.2.1 Ausführungslevel . . . 122
11.2.2 Synchronisation . . . 125
11.2.3 Preemptives Multitasking . . . 126
11.2.4 Tasklets . . . 126
11.3 Entwurf . . . 127
11.3.1 Ausführungslevel . . . 127
11.3.2 Synchronisation . . . 131
11.3.3 Preemptives Multitasking . . . 132
11.3.4 Tasklets . . . 132
11.4 Fazit . . . 133

IV Anwendungen 134

12 Infrastruktur 135
12.1 Einleitung . . . 135
12.1.1 Toolchain Anpassung . . . 135
12.1.2 Reimplementation der bisherigen Treiber . . . 135
12.1.3 Programmlader . . . 135
12.2 Analyse . . . 135
12.3 Entwurf . . . 136
12.3.1 Programmlader . . . 136
12.3.2 Bibliothek . . . 137
12.4 Implementierung . . . 138
12.4.1 Programmlader . . . 138
12.4.2 Bibliothek . . . 139
12.5 Test . . . 140
12.6 Fazit . . . 140

12.7 Ausblick 140

13 Kommandozeile 141

13.1 Analyse 141
13.2 Entwurf 141
13.2.1 Scanner 141
13.2.2 Parser 143
13.3 Implementierte Kommandos 144

V Gerätetreiber 146

14 Treiber-Framework 147

14.1 Was ist ein Treiber Framework 147
14.2 Treiberinitialisierung 147
14.2.1 Mehrstufige Initialisierung 147
14.2.2 Bedarfsinitialisierung 148
14.2.3 Gewählte Strategie 148
14.3 Aufbau 148
14.4 Geräte Klassen 150
14.5 Verwendung 151
14.5.1 Implementierung von Treibern 151
14.5.2 Benutzen von Treibern 151

15 Tastaturtreiber 153

15.1 Grundlagen 153
15.1.1 Aufbau von Tastaturen 153
15.1.2 Tastaturmodelle 154
15.1.3 Funktionsweise von Tastaturen 155
15.1.4 Scancodes 156
15.1.5 PS/2 Kontroller und Tastatur 158
15.2 Entwurf 158
15.2.1 Zusammenspiel der einzelnen Devices 159
15.2.2 PS/2 Device 159
15.2.3 PS/2 Tastatur Device 160
15.2.4 Keycode Device 160
15.2.5 Char Layout Device 164
15.3 Fazit und Ausblick 164

16 Konsole im VGA-Textmodus 166

16.1 Video-RAM 166
16.2 CRT-Controller 167

16.3 Entwurf 167
16.3.1 TextDevice 168
16.3.2 TextConsole 169
16.4 Offene Punkte 170

17 Zeitgeber Intel 8254 171
17.1 Analyse 171
17.2 Entwurf 174

18 Echtzeituhr 176
18.1 Zeitquelle 176
18.1.1 Der Zeitgeber des PIT8254 176
18.1.2 Echtzeituhr 176
18.2 Entwurf und Umsetzung 177
18.2.1 Klassenmodell 177
18.2.2 Der RTC-Treiber 178

19 ATA Festplattentreiber 181
19.1 Einleitung 181
19.2 Analyse 182
19.2.1 Logische Blockadressierung (LBA) 182
19.2.2 Primärer und Sekundärar Bus 182
19.2.3 Master/Slave Laufwerke 183
19.2.4 Register 183
19.2.5 Befehle 187
19.3 Entwurf 188
19.3.1 Klassendiagramm 189
19.3.2 Aufbau der Treiber 189
19.4 Fazit 190

VI Allgemeines 191

20 Programmieren in C++ 193
20.1 Objektzugriff 193
20.1.1 Referenzen 193
20.1.2 Zeiger 194
20.2 Die Wahl des Speichers 196
20.2.1 Der Stapelspeicher – Stack 196
20.2.2 Der Haldenspeicher – Heap 196
20.2.3 Erstellen von Objekten an feste Adressen 198
20.3 Header und Implementierung 199

20.4 Klassendeklarationen 200
20.4.1 Konstruktor und Destruktor 202
20.4.2 Das Schlüsselwort `const` 204
20.4.3 Vererbung 205
20.4.4 Abstrakte Klassen 207
20.5 Überladung von Operatoren 208
20.6 Code-Konventionen 209

21 Refaktorierung und Test-Framework 211
21.1 Quelltexte strukturieren 211
21.1.1 Aktuelle Situation 211
21.1.2 Neue Verzeichnisstruktur 213
21.2 Testframework 215
21.2.1 Analyse 215
21.2.2 Entwurf 216
21.2.3 Verwendung des Testframework 217
21.2.4 Fazit 217
21.3 Strukturieren des Boot-Vorgangs 218
21.4 Initialisierung der Konsole 218

22 Erstellung des Handbuchs 220
22.1 Prozess der Handbuch-Erstellung 220
22.1.1 Struktur der Teildokumente 220
22.1.2 Aufbau des Hauptdokuments 221
22.1.3 Anlegen des Index 223
22.2 Doxygen 223
22.3 Überarbeitung des Handbuchs 224

Literaturverzeichnis 225

Anhang 226

A Doxygen 227
A.1 Hierarchical Index 227
A.1.1 Class Hierarchy 227
A.2 Data Structure Index 238
A.2.1 Data Structures 238
A.3 File Index 254
A.3.1 File List 254
A.4 Data Structure Documentation 263
A.4.1 io::vfs::test::AbstractDummyNode Class Reference 263

A.4.2 task::pipeandfilter::AbstractFilter< T, V > Class Template Reference . . . 263
A.4.3 io::vfs::AbstractMount Class Reference . . . 263
A.4.4 io::driver::keycode::scancodestates::AbstractScanCodeConverterState Class Reference . . . 265
A.4.5 memory::lmm::AddressSpaceRange Class Reference . . . 269
A.4.6 memory::allocator::Allocator Class Reference . . . 286
A.4.7 memory::allocator::test::AllocatorOutOfMemoryTestCase Class Reference . . . 288
A.4.8 test::memory::allocator::AllocatorTestCase Class Reference 289
A.4.9 memory::allocator::test::AllocatorTestCase Class Reference 289
A.4.10 api::memory::lmm::AllocBlockRequest Class Reference . . . 290
A.4.11 ipc::Array< T > Class Template Reference . . . 292
A.4.12 tool::collection::ArrayList< T > Class Template Reference . 294
A.4.13 tool::collection::ArrayListIterator< T > Class Template Reference . . . 302
A.4.14 tool::collection::test::ArrayListTestCase Class Reference . . 304
A.4.15 io::driver::block::ata::AtaBusDevice Class Reference . . . 305
A.4.16 io::driver::block::ata::AtaBusDriver Class Reference . . . 308
A.4.17 io::driver::block::ata::AtaBusIrqHandler Class Reference . . 309
A.4.18 io::driver::block::ata::AtaBusWorkerThread Class Reference 311
A.4.19 io::driver::block::ata::commands::AtaCommand Class Reference . . . 312
A.4.20 io::driver::block::ata::AtaCommandRegister Struct Reference 317
A.4.21 io::driver::block::ata::AtaDevice Class Reference . . . 318
A.4.22 io::driver::block::ata::AtaDigitalOutputRegister Struct Reference . . . 323
A.4.23 io::driver::block::ata::AtaDriveLbaHighest Struct Reference 323
A.4.24 io::driver::block::ata::AtaDriver Class Reference . . . 324
A.4.25 io::driver::block::ata::AtaErrorRegister Struct Reference . . 325
A.4.26 io::driver::block::ata::commands::AtaIdentifyCommand Class Reference . . . 325
A.4.27 io::driver::block::ata::AtaIdentifyResult Class Reference . . 327
A.4.28 io::driver::block::ata::AtaIoPorts Class Reference . . . 329
A.4.29 io::driver::block::ata::commands::AtaReadCommand Class Reference . . . 330
A.4.30 io::driver::block::ata::test::AtaReadWriteTest Class Reference 332
A.4.31 io::driver::block::ata::test::AtaReadWriteTest2 Class Reference 333
A.4.32 io::driver::block::ata::test::AtaReadWriteTest3 Class Reference 333
A.4.33 io::driver::block::ata::AtaStatusRegister Struct Reference . 334

A.4.34 io::driver::block::ata::AtaTasklet Class Reference 334
A.4.35 io::driver::block::ata::commands::AtaWriteCommand Class Reference 337
A.4.36 ipc::Attribute< T > Class Template Reference 339
A.4.37 ipc::Attribute< Array< T > > Class Template Reference . . 340
A.4.38 ipc::Attribute< Array< T const > > Class Template Reference 342
A.4.39 ipc::Attribute< T & > Class Template Reference 343
A.4.40 ipc::Attribute< T const > Class Template Reference 344
A.4.41 ipc::Attribute< T const & > Class Template Reference . . . 346
A.4.42 io::driver::classes::AudioClass Class Reference 347
A.4.43 tool::BitField< Base, Index, BitPosition, Length > Class Template Reference 348
A.4.44 io::driver::block::Block Struct Reference 350
A.4.45 io::driver::block::BlockDevice Class Reference 351
A.4.46 io::vfs::BlockDeviceVolume Class Reference 355
A.4.47 boot::BootConsole Class Reference 359
A.4.48 boot::BootManager Class Reference 364
A.4.49 io::vfs::fat::Bootsector Class Reference 367
A.4.50 boot::BootThread Class Reference 373
A.4.51 BPB Struct Reference 375
A.4.52 RootDirectory::Entry::Callback Struct Reference 377
A.4.53 io::driver::charlayout::CharLayoutDevice Class Reference . 378
A.4.54 io::driver::charlayout::CharLayoutDriver Class Reference . . 380
A.4.55 io::driver::charlayout::CharLayoutFilter Class Reference . . 380
A.4.56 io::driver::charlayout::test::CharLayoutTestCase Class Reference 382
A.4.57 boot::CleanupTasklet Class Reference 383
A.4.58 fosCli::commands::ClearCliCommand Class Reference . . . 384
A.4.59 fosCli::commands::ClearCliCommandCreator Class Reference 384
A.4.60 api::io::console::ClearScreenCommand Class Reference . . 385
A.4.61 api::io::console::ClearScreenRequest Class Reference . . . 386
A.4.62 fosCli::parser::CliCommand Class Reference 387
A.4.63 fosCli::parser::CliCommandCreator Class Reference 388
A.4.64 ClockThread Class Reference 388
A.4.65 memory::CodeOrDataSegmentDescriptor Class Reference . 389
A.4.66 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add > Class Template Reference 397
A.4.67 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_getAnswer > Class Template Reference 398

A.4.68 ipc::test::FantasticObjectProxy::Command< FantasticObject-Proxy::Id_increment > Class Template Reference 399
A.4.69 ipc::test::FantasticObjectProxy::Command< FantasticObject-Proxy::Id_incrementElements > Class Template Reference . 399
A.4.70 ipc::test::FantasticObjectProxy::Command< FantasticObject-Proxy::Id_incrementInPlace > Class Template Reference . . 400
A.4.71 ipc::test::FantasticObjectProxy::Command< FantasticObject-Proxy::Id_print > Class Template Reference 401
A.4.72 ipc::test::FantasticObjectProxy::CommandBase Class Reference 402
A.4.73 fosCli::parser::CommandDirectory Class Reference 403
A.4.74 fosCli::parser::CommandDirectoryMaster Class Reference . 404
A.4.75 fosCli::parser::CommandDirectorySlave Class Reference . . 405
A.4.76 ipc::CommandRelay Class Reference 405
A.4.77 tool::collection::Comparator< T > Class Template Reference 407
A.4.78 tool::collection::Comparator< char const ∗ > Class Template Reference 408
A.4.79 lib::Console Class Reference 409
A.4.80 io::console::ConsoleManager Class Reference 410
A.4.81 tool::collection::ConstArrayListIterator< T > Class Template Reference 412
A.4.82 tool::collection::ConstIterator< T > Class Template Reference415
A.4.83 tool::collection::ConstLinkedListIterator< T > Class Template Reference 416
A.4.84 task::spinlock::test::Counter Class Reference 419
A.4.85 api::task::lock::CreateSemaphoreRequest Class Reference . 419
A.4.86 memory::allocator::Allocator::CriticalSection Class Reference420
A.4.87 api::io::time::CurrentTimeRequest Class Reference 422
A.4.88 io::driver::graphics::Cursor Class Reference 423
A.4.89 io::console::DefaultConsole Class Reference 423
A.4.90 cpu::interrupt::DefaultExceptionHandler Class Reference . 427
A.4.91 cpu::interrupt::DefaultHandler Class Reference 428
A.4.92 io::driver::graphics::Delta Class Reference 429
A.4.93 memory::Descriptor Class Reference 432
A.4.94 io::driver::Device Class Reference 436
A.4.95 io::driver::DeviceClass Class Reference 437
A.4.96 io::driver::DeviceManager Class Reference 437
A.4.97 io::vfs::Directory Class Reference 440
A.4.98 io::vfs::fat::DirectoryEntry Struct Reference 447
A.4.99 Disk Class Reference 448

A.4.100 DiskAddress Class Reference 451
A.4.101 io::driver::Driver Class Reference 455
A.4.102 io::driver::DriverManager::DriverDescriptor Class Reference 457
A.4.103 io::driver::test::DriverFrameworkTestCase Class Reference . 459
A.4.104 io::driver::DriverManager Class Reference 459
A.4.105 io::vfs::test::DummyDirectory Class Reference 462
A.4.106 io::vfs::test::DummyFile Class Reference 467
A.4.107 io::vfs::test::DummyFileStream Class Reference 469
A.4.108 io::vfs::test::DummyFileSystem Class Reference 472
A.4.109 io::vfs::test::DummyLeaf Class Reference 474
A.4.110 io::vfs::test::DummyMount Class Reference 474
A.4.111 io::vfs::test::DummyNode Class Reference 475
A.4.112 memory::E820_entry Struct Reference 476
A.4.113 fosCli::commands::EchoCliCommand Class Reference . . . 478
A.4.114 fosCli::commands::EchoCommandCreator Class Reference 478
A.4.115 RootDirectory::Entry Struct Reference 479
A.4.116 memory::allocator::Environment Class Reference 481
A.4.117 io::driver::keycode::scancodestates::EOnePhaseOneScanCode-ConverterState Class Reference 483
A.4.118 io::driver::keycode::scancodestates::EOnePhaseTwoScanCode-ConverterState Class Reference 485
A.4.119 runtime::test::ExceptionTestCase Class Reference 487
A.4.120 api::task::scheduler::ExitThreadRequest Class Reference . . 489
A.4.121 io::driver::keycode::scancodestates::EZeroScanCodeConverter-State Class Reference 489
A.4.122 ipc::test::FantasticObject Class Reference 492
A.4.123 ipc::test::FantasticObjectImpl Class Reference 493
A.4.124 ipc::test::FantasticObjectProxy Class Reference 493
A.4.125 memory::FarPointer Struct Reference 496
A.4.126 FAT Class Reference 497
A.4.127 FAT::FAT12Type Class Reference 502
A.4.128 FAT::FAT16Type Class Reference 504
A.4.129 io::vfs::fat::FatAccess Class Reference 506
A.4.130 io::vfs::fat::FatDirectory Class Reference 509
A.4.131 io::vfs::fat::FatFile Class Reference 513
A.4.132 io::vfs::fat::FatFileStream Class Reference 515
A.4.133 io::vfs::fat::FatFileSystem Class Reference 517
A.4.134 io::vfs::fat::FatMount Class Reference 518
A.4.135 FatObject Class Reference 519
A.4.136 io::vfs::fat::FatRootDirectory Class Reference 519

A.4.137 FAT::FATType Class Reference 524
A.4.138 io::vfs::File Class Reference 525
A.4.139 io::vfs::FileStream Class Reference 528
A.4.140 io::vfs::FileSystem Class Reference 532
A.4.141 io::vfs::FileSystemNode Class Reference 534
A.4.142 io::vfs::FileSystemNodeVisitor Class Reference 538
A.4.143 task::pipeandfilter::FilterThread< T, V > Class Template Reference 539
A.4.144 fosCli::FosCli Class Reference 540
A.4.145 fosCli::scanner::elements::FosCliElement Class Reference . 541
A.4.146 fosCli::scanner::elements::FosCliElementVisitor Class Reference 542
A.4.147 fosCli::scanner::states::FosCliExecuteScannerState Class Reference 542
A.4.148 fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState Class Reference 543
A.4.149 fosCli::scanner::FosCliScanner Class Reference 544
A.4.150 fosCli::scanner::FosCliScannerState Class Reference 545
A.4.151 fosCli::scanner::FosCliScannerStateResult Class Reference . 548
A.4.152 fosCli::scanner::elements::FosCliStringElement Class Reference 548
A.4.153 api::memory::lmm::FreeBlockRequest Class Reference . . . 550
A.4.154 memory::GateDescriptor Class Reference 552
A.4.155 cpu::GDTManager Class Reference 555
A.4.156 api::kernel::GetVersionRequest Class Reference 559
A.4.157 fosCli::commands::HelpCliCommand Class Reference 560
A.4.158 fosCli::commands::HelpCliCommandCreator Class Reference 561
A.4.159 task::priorityinheritance::test::HighPrioThread Class Reference 561
A.4.160 task::scheduler::IdleThread Class Reference 562
A.4.161 io::driver::keycode::scancodestates::InitialScanCodeConverterState Class Reference 563
A.4.162 cpu::interrupt::InterruptHandler Class Reference 567
A.4.163 cpu::interrupt::InterruptHandlerContainer Class Reference . 568
A.4.164 cpu::interrupt::InterruptManager Class Reference 569
A.4.165 io::IOPort Class Reference 573
A.4.166 ipc::test::IPCReceiverThread Class Reference 575
A.4.167 ipc::test::IPCSenderThread Class Reference 576
A.4.168 ipc::test::IPCTestCase Class Reference 578
A.4.169 io::driver::interrupt::IRQHandler Class Reference 578

A.4.170 tool::collection::Iterator< T > Class Template Reference . . 579
A.4.171 memory::KernelEnvironment Class Reference 582
A.4.172 fosCli::commands::KernelVersionCliCommand Class Reference584
A.4.173 fosCli::commands::KernelVersionCliCommandCreator Class Reference 584
A.4.174 io::driver::keycode::test::KeyboardTestCase Class Reference 585
A.4.175 io::driver::keycode::Keycode Struct Reference 585
A.4.176 io::driver::keycode::KeycodeDevice Class Reference 586
A.4.177 io::driver::keycode::KeycodeDriver Class Reference 587
A.4.178 io::driver::keycode::KeycodeFilter Class Reference 588
A.4.179 io::driver::keycode::KeycodeTasklet Class Reference 588
A.4.180 io::driver::charlayout::keymap_entry Struct Reference . . . 589
A.4.181 tool::collection::KeyValuePair< Key, Value, KeyComp, ValueComp > Class Template Reference 590
A.4.182 memory::lmm::test::LASMTest Class Reference 591
A.4.183 io::driver::block::LBAddress Struct Reference 591
A.4.184 cpu::level::LevelManager Class Reference 592
A.4.185 memory::lmm::LinearAddressSpaceManager Class Reference597
A.4.186 tool::collection::LinearMap< Key, Value, KeyComp, ValueComp > Class Template Reference 606
A.4.187 tool::collection::test::LinearMapTestCase Class Reference . 612
A.4.188 tool::collection::LinkedList< T > Class Template Reference . 613
A.4.189 tool::collection::LinkedListEntry< T > Class Template Reference 620
A.4.190 tool::collection::LinkedListIterator< T > Class Template Reference 624
A.4.191 tool::collection::test::LinkedListTestCase Class Reference . 627
A.4.192 tool::collection::List< T > Class Template Reference 628
A.4.193 api::task::LoadProgramRequest Class Reference 633
A.4.194 task::lock::Semaphore::Lock Class Reference 636
A.4.195 task::lock::SpinLock::Lock Class Reference 637
A.4.196 LockTesterThreadDown Class Reference 638
A.4.197 LockTesterThreadUp Class Reference 639
A.4.198 Logger Class Reference 640
A.4.199 task::priorityinheritance::test::LowPrioThread Class Reference640
A.4.200 tool::collection::Map< Key, Value, KeyComp, ValueComp > Class Template Reference 642
A.4.201 ipc::MapRequest Class Reference 647
A.4.202 io::driver::interrupt::MaskingHandler Class Reference . . . 648
A.4.203 io::driver::interrupt::MasterPICController Class Reference . 651

A.4.204 tool::MD5 Class Reference 654
A.4.205 api::task::lock::ModifySemaphoreRequest Class Reference . 655
A.4.206 io::vfs::Mount Class Reference 656
A.4.207 tool::collection::Queue< T >::Node Struct Reference 659
A.4.208 task::pipeandfilter::Pipe< T >::Node Struct Reference . . . 660
A.4.209 task::priorityinheritance::test::NormalThread Class Reference660
A.4.210 api::io::console::OutputStringRequest Class Reference . . . 662
A.4.211 memory::allocator::Page Class Reference 663
A.4.212 memory::paging::PageContext Class Reference 668
A.4.213 memory::paging::PageDirectory Class Reference 672
A.4.214 memory::paging::PageEntry Class Reference 682
A.4.215 memory::paging::PageFaultExceptionHandler Class Reference684
A.4.216 memory::paging::PageFlags Class Reference 685
A.4.217 memory::allocator::PageHeader Struct Reference 688
A.4.218 memory::paging::PageTable Class Reference 689
A.4.219 memory::allocator::PageTable Class Reference 690
A.4.220 memory::paging::test::PagingTestCase Class Reference . . 695
A.4.221 fosCli::parser::Parser Class Reference 696
A.4.222 memory::allocator::Page::PartDescriptor Class Reference . 697
A.4.223 ipc::Participant Class Reference 698
A.4.224 memory::pmm::PhysicalMemoryManager Class Reference . 699
A.4.225 io::driver::interrupt::PICController Class Reference 704
A.4.226 io::driver::interrupt::PICDevice Class Reference 708
A.4.227 io::driver::interrupt::PICDriver Class Reference 711
A.4.228 io::driver::interrupt::PICInterruptHandler Class Reference . 712
A.4.229 task::pipeandfilter::Pipe< T > Class Template Reference . . 715
A.4.230 io::driver::timer::PIT8254ControlWord Class Reference . . . 717
A.4.231 io::driver::timer::PIT8254Counter Class Reference 719
A.4.232 io::driver::timer::PIT8254Device Class Reference 722
A.4.233 io::driver::timer::PIT8254Driver Class Reference 726
A.4.234 memory::pmm::PmmTableEntry Struct Reference 727
A.4.235 io::driver::graphics::Position Class Reference 727
A.4.236 task::priorityinheritance::test::PriorityInheritanceTestCase Class Reference 737
A.4.237 task::Process Class Reference 737
A.4.238 api::task::ProcessIdRequest Class Reference 744
A.4.239 task::ProcessManager Class Reference 744
A.4.240 task::ProgramLoader Class Reference 747
A.4.241 io::driver::pstwo::PS2Device Class Reference 749
A.4.242 io::driver::pstwo::PS2Driver Class Reference 757

A.4.243 io::driver::pstwokeyboard::PS2KeyboardDevice Class Reference . . . 757
A.4.244 io::driver::pstwokeyboard::PS2KeyboardDriver Class Reference . . . 760
A.4.245 io::driver::pstwokeyboard::PS2KeyboardHandler Class Reference . . . 761
A.4.246 tool::collection::Queue< T > Class Template Reference . . . 762
A.4.247 api::io::console::ReadKeyRequest Class Reference . . . 764
A.4.248 object::Ref< T > Class Template Reference . . . 765
A.4.249 object::RefCountedObject Class Reference . . . 769
A.4.250 object::test::RefTestCase Class Reference . . . 770
A.4.251 ipc::Registry Class Reference . . . 771
A.4.252 ipc::RemoteObject Class Reference . . . 773
A.4.253 ipc::Request Class Reference . . . 775
A.4.254 task::Process::Request Class Reference . . . 777
A.4.255 ipc::test::FantasticObjectProxy::Request< 0 > Class Template Reference . . . 778
A.4.256 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_add > Class Template Reference . . . 779
A.4.257 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_getAnswer > Class Template Reference . . . 780
A.4.258 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_increment > Class Template Reference . . . 780
A.4.259 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementElements > Class Template Reference . 781
A.4.260 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementInPlace > Class Template Reference . . 782
A.4.261 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_print > Class Template Reference . . . 783
A.4.262 ipc::test::FantasticObjectProxy::RequestBase Class Reference 784
A.4.263 memory::lmm::LinearAddressSpaceManager::ReservationTable Struct Reference . . . 785
A.4.264 task::Thread::Resource Struct Reference . . . 786
A.4.265 ipc::Result< T > Class Template Reference . . . 788
A.4.266 ipc::Result< Array< T > > Class Template Reference . . . 789
A.4.267 ipc::Result< void > Class Template Reference . . . 790
A.4.268 RootDirectory Class Reference . . . 791
A.4.269 io::driver::rtc::RTCDevice Class Reference . . . 793
A.4.270 io::driver::rtc::RTCDriver Class Reference . . . 797
A.4.271 io::driver::rtc::RTCHandler Class Reference . . . 797

A.4.272 io::driver::rtc::RTCTasklet Class Reference 799
A.4.273 runtime::test::RTTITestCase Class Reference 800
A.4.274 task::scheduler::Scheduler Class Reference 801
A.4.275 task::scheduler::SchedulerHandler Class Reference 809
A.4.276 memory::Selector Class Reference 811
A.4.277 task::lock::Semaphore Class Reference 816
A.4.278 lib::Semaphore Class Reference 819
A.4.279 task::semaphore::test::SemaphoreTestCase Class Reference 820
A.4.280 io::driver::interrupt::SlavePICController Class Reference . . 821
A.4.281 io::driver::speaker::SpeakerDevice Class Reference 823
A.4.282 io::driver::speaker::SpeakerDriver Class Reference 823
A.4.283 task::lock::SpinLock Class Reference 824
A.4.284 task::spinlock::test::SpinLockTestCase Class Reference . . . 827
A.4.285 task::spinlock::test::SpinLockTestThread Class Reference . 827
A.4.286 boot::SplashScreen Class Reference 828
A.4.287 tool::test::StringTestCase Class Reference 829
A.4.288 ipc::SysCallHandler Class Reference 830
A.4.289 ipc::test::SysCallTestCase Class Reference 834
A.4.290 lib::System Class Reference 835
A.4.291 lib::System::SystemCallFailed Class Reference 839
A.4.292 memory::SystemSegmentDescriptor Class Reference . . . 840
A.4.293 task::tasklet::Tasklet Class Reference 842
A.4.294 task::tasklet::TaskletManager Class Reference 844
A.4.295 task::TaskStateSegment Struct Reference 846
A.4.296 runtime::TCB Struct Reference 847
A.4.297 tool::Terminal Class Reference 849
A.4.298 test::TestCase Class Reference 850
A.4.299 io::driver::test::TestDevice Class Reference 851
A.4.300 io::driver::test::TestDevice2 Class Reference 852
A.4.301 io::driver::test::TestDriver Class Reference 852
A.4.302 io::driver::test::TestDriver2 Class Reference 853
A.4.303 task::pipeandfilter::test::TesterPipe< T > Class Template Reference 854
A.4.304 test::TestManager Class Reference 855
A.4.305 task::tasklet::test::TestTasklet Class Reference 856
A.4.306 io::driver::graphics::TextChar Class Reference 857
A.4.307 io::console::TextConsole Class Reference 860
A.4.308 io::driver::graphics::TextDevice Class Reference 862
A.4.309 task::Thread Class Reference 865

A.4.310 task::semaphore::test::ThreadASemaphoreTestCase Class Reference . . . 875
A.4.311 task::scheduler::ThreadQueue Class Reference . . . 876
A.4.312 task::ThreadStack Class Reference . . . 879
A.4.313 io::time::Time Class Reference . . . 881
A.4.314 fosCli::commands::TimeCliCommand Class Reference . . . 884
A.4.315 fosCli::commands::TimeCliCommandCreator Class Reference 884
A.4.316 task::scheduler::TimerInterruptHandler Class Reference . . . 884
A.4.317 cpu::level::TransitionHandler Class Reference . . . 886
A.4.318 memory::TSSDescriptor Class Reference . . . 888
A.4.319 TypeInfo< int16_t > Struct Template Reference . . . 891
A.4.320 TypeInfo< int32_t > Struct Template Reference . . . 891
A.4.321 TypeInfo< int64_t > Struct Template Reference . . . 891
A.4.322 TypeInfo< int8_t > Struct Template Reference . . . 891
A.4.323 TypeInfo< uint16_t > Struct Template Reference . . . 892
A.4.324 TypeInfo< uint32_t > Struct Template Reference . . . 892
A.4.325 TypeInfo< uint64_t > Struct Template Reference . . . 892
A.4.326 TypeInfo< uint8_t > Struct Template Reference . . . 893
A.4.327 lib::runtime::UserEnvironment Class Reference . . . 893
A.4.328 task::UserModeThread Class Reference . . . 895
A.4.329 boot::UserSpaceInitThread Class Reference . . . 896
A.4.330 io::vfs::VFSManager Class Reference . . . 897
A.4.331 io::vfs::test::VFSTestCase Class Reference . . . 901
A.4.332 io::driver::graphics::vga::VgaChar Class Reference . . . 901
A.4.333 io::driver::graphics::vga::test::VgaCharTestCase Class Reference . . . 903
A.4.334 io::driver::graphics::vga::VgaCursor Class Reference . . . 903
A.4.335 io::driver::graphics::vga::VgaDriver Class Reference . . . 905
A.4.336 io::driver::graphics::vga::VgaTextDevice Class Reference . . . 906
A.4.337 io::vfs::Volume Class Reference . . . 909
A.4.338 io::time::WaitingQueue Class Reference . . . 914
A.4.339 io::time::WaitingQueueEntry Class Reference . . . 915
A.5 File Documentation . . . 917
A.5.1 a20.h File Reference . . . 917
A.5.2 AddressSpaceRange.h File Reference . . . 917
A.5.3 Allocator.h File Reference . . . 932
A.5.4 AllocatorOutOfMemoryTestCase.h File Reference . . . 932
A.5.5 AtaBusDevice.h File Reference . . . 933
A.5.6 AtaBusDriver.h File Reference . . . 933
A.5.7 AtaBusIrqHandler.h File Reference . . . 933

A.5.8 AtaBusWorkerThread.h File Reference 934
A.5.9 AtaCommand.h File Reference 934
A.5.10 AtaCommandRegister.h File Reference 934
A.5.11 AtaDevice.h File Reference 934
A.5.12 AtaDigitalOutputRegister.h File Reference 935
A.5.13 AtaDriveLbaHighest.h File Reference 935
A.5.14 AtaDriver.h File Reference 935
A.5.15 AtaErrorRegister.h File Reference 935
A.5.16 AtaIdentifyCommand.h File Reference 935
A.5.17 AtaIdentifyResult.h File Reference 935
A.5.18 AtaIoPorts.h File Reference 936
A.5.19 AtaReadCommand.h File Reference 936
A.5.20 AtaReadWriteTest.h File Reference 936
A.5.21 AtaReadWriteTest3.h File Reference 936
A.5.22 AtaTasklet.h File Reference 936
A.5.23 AtaWriteCommand.h File Reference 937
A.5.24 BitField.h File Reference 937
A.5.25 Block.h File Reference 937
A.5.26 BlockDevice.h File Reference 937
A.5.27 BootConsole.h File Reference 937
A.5.28 BootManager.h File Reference 938
A.5.29 BootThread.h File Reference 938
A.5.30 bpb.h File Reference 938
A.5.31 CleanupTasklet.h File Reference 938
A.5.32 Command.h File Reference 939
A.5.33 CommandRelay.h File Reference 939
A.5.34 Comparator.h File Reference 940
A.5.35 cpu.h File Reference 940
A.5.36 CreateSemaphoreCommand.h File Reference 945
A.5.37 CreateSemaphoreRequest.h File Reference 945
A.5.38 Cursor.h File Reference 946
A.5.39 DefaultConsole.h File Reference 946
A.5.40 defs.h File Reference 946
A.5.41 Delta.h File Reference 946
A.5.42 dir.h File Reference 946
A.5.43 disk.h File Reference 947
A.5.44 Environment.h File Reference 947
A.5.45 error.h File Reference 948
A.5.46 ExitThreadCommand.h File Reference 949
A.5.47 ExitThreadRequest.h File Reference 949

A.5.48 fat.h File Reference 949
A.5.49 GetVersionCommand.h File Reference 950
A.5.50 GetVersionRequest.h File Reference 950
A.5.51 highmem.h File Reference 950
A.5.52 HighPrioThread.h File Reference 952
A.5.53 InterruptManager.h File Reference 952
A.5.54 IRQHandler.h File Reference 953
A.5.55 KernelEnvironment.h File Reference 953
A.5.56 LASMTest.h File Reference 953
A.5.57 LowPrioThread.h File Reference 953
A.5.58 memmap.h File Reference 954
A.5.59 MemoryConstants.h File Reference 956
A.5.60 ModifySemaphoreCommand.h File Reference 957
A.5.61 ModifySemaphoreRequest.h File Reference 957
A.5.62 nmi.h File Reference 957
A.5.63 NormalThread.h File Reference 957
A.5.64 OutputStringCommand.h File Reference 958
A.5.65 OutputStringRequest.h File Reference 958
A.5.66 Page.h File Reference 958
A.5.67 PageFaultExceptionHandler.h File Reference 958
A.5.68 PageTable.h File Reference 959
A.5.69 paging.h File Reference 959
A.5.70 Participant.h File Reference 961
A.5.71 PICDevice.h File Reference 961
A.5.72 PICInterruptHandler.h File Reference 961
A.5.73 Position.h File Reference 962
A.5.74 ProcessManager.h File Reference 962
A.5.75 Proxy.h File Reference 962
A.5.76 Ref.h File Reference 965
A.5.77 RefCountedObject.h File Reference 965
A.5.78 Registry.h File Reference 966
A.5.79 RemoteObject.h File Reference 966
A.5.80 Request.h File Reference 966
A.5.81 runtime.h File Reference 966
A.5.82 runtime.h File Reference 966
A.5.83 SpinLockTestThread.h File Reference 967
A.5.84 StringComparator.h File Reference 967
A.5.85 SysCallHandler.h File Reference 967
A.5.86 TextChar.h File Reference 967
A.5.87 ThreadPriority.h File Reference 967

A.5.88 ThreadQueue.h File Reference 968
A.5.89 types.h File Reference 968
A.5.90 UserModeThread.h File Reference 969
A.5.91 UserSpaceInitThread.h File Reference 970
A.5.92 utils.h File Reference 970
A.5.93 Version.h File Reference 974
A.5.94 VgaCursor.h File Reference 975

B Index **976**
Index – Handbuch 977

Abbildungsverzeichnis

1.1 Segmentierung 4
1.2 Segmentselektor 5
1.3 Aufbau Segmentdeskriptor 6
1.4 Basic Flat Model 8
1.5 Protected Flat Model 8
1.6 Umwandlung der linearen Adressen zu physikalischen Adressen . . 12

2.1 Blockaufteilung nach dem Buddy-Algorithmus 16
2.2 allocate 21

3.1 PmmTable 30
3.2 Speicher allokieren 31
3.3 Speicher freigeben 32
3.4 Lower Half Kernel 33
3.5 Higher Half Kernel 33
3.6 User- und Kernel-Space im physikalischen Speicher 35
3.7 Rekursiver Verweis im Paging Directory 37
3.8 Lineare Adressen zum Zugriff auf die Pagingstrukturen 37
3.9 Klasse PageContext 38
3.10 Klassenstruktur des internen Bereiches 39
3.11 Ablauf des Abbildens 41

4.1 Ringmodell der Privilegstufen 43
4.2 Segment Zugriff 45
4.3 Offset bei Verwendung von Call Gates 47
4.4 Kombination von Segment- und Seitenschutz 51

5.1 Dateisystem-Kompositum 54
5.2 VFS in modernen Betriebssystemen 54
5.3 VFS Klassendiagramm 57

6.1 Beziehung zwischen IDT und dem IDTR 67

7.1 Organisation der Behandler 74
7.2 Klassendiagramm 75

7.3 Implementierung der LinkedList 77

8.1 Ablauf des Lock-Verfahrens mit aktivem Warten 82
8.2 Klassendiagramm 87
8.3 Anfordern des Mutexes 89
8.4 Mutex freigeben 89
8.5 Methode down der Semaphore 91
8.6 Methode up der Semaphore 92

9.1 Task-State-Segment 94
9.2 Threadzustände 96
9.3 Threadzustände 101
9.4 Scheduler 102
9.5 loadNextProcess(processToStart) 103
9.6 Klassendiagramm Tasks I 106

10.1 Aufbau eines Call Gates 110
10.2 Kapselung von Nachrichten in Commands 112
10.3 Aufbau gemeinsamen Speichers 114
10.4 SMM und LASM 114

11.1 Verhalten der Ausführungslevel beim Threadwechsel 123
11.2 Verhalten der Ausführungslevel bei mehreren Interrupts 124
11.3 Übersicht über die verschiedenen Ausführungslevel 127
11.4 Klassendiagramm der Ausführungslevel 129
11.5 Klassendiagramm der Ausführungslevel 130
11.6 Synchronisations Beispiel 131
11.7 Tasklets 133

12.1 Klasse ProgramLoader 137
12.2 Entwurf der Bibliothek 138

13.1 Ablauf des CLI 142
13.2 Klassendiagramm Scanner 143
13.3 Parser für CLI 144

14.1 Ablauf der Geräteinitialisierung 149
14.2 Aufbau des Treiberframeworks 150

15.1 PC Tastatur (PC/XT) 154
15.2 AT-Tastatur 155
15.3 MF II-Tastatur 155
15.4 Zusammenspiel der einzelnen Devices 159

15.5 PS/2 Device 160
15.6 PS/2 Tastatur Device 161
15.7 Zustandsdiagramm zur Behandlung geschützer Scan-Codes 163
15.8 Keycode Device 163
15.9 Char Layout Device 164

16.1 Struktur eines Zeichens im Video-RAM 166
16.2 4-Bit-Farbpalette im VGA-Textmodus 167
16.3 Klassendiagramm des TextDevice und der TextConsole 168
16.4 Klassendiagramm des VGA-Texttreibers 169

17.1 Hardwareschaubild des Intel 8254 171
17.2 PIT8254 Klassenmodell 174
17.3 PIT8254 Klassenmodell Hilfsklassen 174

18.1 Klassenmodell zu Zeit und Warten 177

19.1 Klassendiagramm 189

20.1 Mixin-Beispiel UML 206

21.1 Verzeichnisstruktur vor dem Umbau (Revision 868) 212
21.2 Verzeichnisstruktur nach der Refaktorierung (Revision 873) 214
21.3 Basisklassen des Testframework 216
21.4 Klassenmodell zum ConsoleManager 219

22.1 Interne Struktur des Handbuchs 221

Tabellenverzeichnis

1.1 Beispiel von Code- und Datensegmentdeskriptortypen 7
1.2 Die verschiedenen Paging Modi 10
1.3 Bit-Belegung CR3, PDE und PTE 11

2.1 Bedeutung usedParts und neighbors 22
2.2 Beispiel usedParts und neighbors 23

3.1 Bitbelegung der Tabelle zur physikalischen Speicherverwaltung . . 29
3.2 sinnvolle Kombinationen der Bitbelegung 29

4.1 Privilegstufen Prüfung 44

5.1 Typische Dateisystemoperationen 53

6.1 Prioritäten bei gleichzeitig auftreten Unterbrechungen und Ausnahmen 71
6.2 Liste der Vektornummernzuordnungen 72

9.1 Verweis auf die GDT 97
9.2 Struktur GDT-Eintrag 98

14.1 Geräte Klassen und ihre Operationen 151

15.1 IO Ports 158
15.2 Keycodes 162

18.1 RTC Bytes 0x00 – 0x09: Datum und Uhrzeit 178
18.2 RTC Register A (Frequenzwahl) 179
18.3 RTC-Statusregister B 179
18.4 RTC-Statusregister C (Read-Only) 180

19.1 Register 183
19.2 Statusregister 184
19.3 Alternatives Statusregister 184
19.4 Befehle 185
19.5 Gerätekontrollregister 185

19.6 Geräte/Kopfregister 185
19.7 Fehlerregister 186
19.8 Digitales Ausgaberegister 186
19.9 Laufwerkadressregister 187

20.1 Vererbung mit Sichtbarkeitsmodifizierern 205

Listings

2.1 Konstruktor-Operation des Allokators 17
2.2 alloc und free Operationen des Allokators 17
2.3 Headerdatei der BuddyPageTable 18
2.4 Headerdatei der BuddyPageUsage 19
2.5 heap.h 20

8.1 Beispiel strenge Alternation 81
8.2 Compare-and-swap in C++ 83
8.3 Compare-and-swap für kritische Bereiche 84
8.4 Test-and-set in C++ 85
8.5 Test-and-set für kritische Bereiche 85
8.6 Fetch-and-add in C++ 86
8.7 Fetch-and-add für kritische Bereiche 86

9.1 Laden der GDT 99
9.2 Laden der TSS 103

10.1 Einfügen eines Elements nach einem bestimmten 117
10.2 Konstruktor von Prozess 118
10.3 Aufruf des Konstruktors der Klasse Prozess 118
10.4 CommandRelay::relayCommand 119
10.5 Abschicken des ConsoleCommands 120

20.1 Nutzung von Referenzen 193
20.2 Call by value/reference 194
20.3 Zeiger verwenden 195
20.4 Speichern auf dem Stack 196
20.5 Stack und Heap 197
20.6 Placement new 198
20.7 Fehler: Aufruf der Funktion ohne Deklaration 199
20.8 Forward declaration mit Hilfe von Header-Dateien 199
20.9 Klassendeklaration - Datei `person.h` 201
20.10 Implementierung - Datei `person.cpp` 202
20.11 Konstruktoren 203

20.12 Übergabe eines const-Zeigers 204
20.13 Deklaration der Klasse `Person` für Listing 204
20.14 Virtuelle Methoden 205
20.15 Mixin-Deklaration 206
20.16 Abstrakte Klasse 207
20.17 Klassendeklaration für Brüche - Datei `bruch.h` 208
20.18 Implementierung für Brüche - Datei `bruch.cpp` 208
20.19 Verwendung der Klasse Bruch - Datei `main.cpp` 209

21.1 Boot-Vorgang 218

22.1 Ergänzung der LaTeX-Vorlage 221
22.2 Automatisches Präfix für Kapitel 222
22.3 Verwendung des Präfixes an Bezeichnern 222
22.4 Makeindex Style 223
22.5 Auszug aus der Doxyfile 224

Teil I

Speicherverwaltung

1 Speicherverwaltung der IA-32-Architektur

1.1 Einleitung

Die Verwaltung der Ressource Speicher ist für ein Betriebssystem eine wichtige Aufgabe um den Programmierer bei seiner Arbeit zu unterstützen.

Den Teil des Betriebssystems, der diese Aufgabe durchführt, nennt man die **Speicherverwaltung**. Seine Aufgabe ist es, die Übersicht darüber zu behalten, welche Teile des Speichers bereits belegt und welche noch frei verfügbar sind. Ferner übernimmt er die Zuteilung von Speicher zu den anfordernden Prozessen und die Freigabe des Speichers, wenn die Prozesse ihn nicht mehr benötigen.

In der **IA-32-Architektur**[1] sind weite Teile der Speicherverwaltung in einer **Speicherverwaltungseinheit**[2] fest in der Hardware implementiert und unterstützen fortschrittliche Konzepte wie

- die virtuelle Speicherverwaltung,
- die Segmentierung
- und die Seitenverwaltung[3],

die im Folgenden genauer vorgestellt werden.

1.2 Virtuelle Speicherverwaltung

Die **virtuelle Speicherverwaltung** beschreibt die Abstraktion der realen **physikalischen Adressen** mit **virtuellen Adressen** um jedem Prozess einen eigenen exklusiven Speicherraum (der auch bei der gleichen Adresse beginnen kann) zur Verfügung zu stellen.

[1] Informationen zu der *IA-32* Architektur stammen aus [1].
[2] auf englisch: Memory Management Unit(MMU)
[3] Im Folgenden wird nur noch die englische Übersetzung **Paging** verwendet.

Damit gehen Schutzmechanismen einher die bewirken, dass Programme von einander separiert werden und nicht auf den Speicher anderer Programme zugreifen können und dass das Betriebssystem durch den Zugriff von fehlerhaften Programmen geschützt wird. Ferner kann die Speicherfragmentierung des linearen Adressraums vermieden werden, da zusammenhängende virtuelle Adressen auf auseinanderliegende physikalische Adressen abgebildet werden können.

Für die Organisation des virtuellen Speichers gibt es den **segmentorientierten Ansatz**, bei dem der Speicher in unterschiedlich große Teile aufgeteilt wird, und den **seitenorientierten Ansatz** bei dem diese Teile gleich groß sind.

1.3 Segmentierung

Segmentierung[4] ist die Unterteilung des benutzen Speicherraums in einzelne **Segmente** zwecks Schutzmechanismen oder virtueller Speicherverwaltung. Beispiele sind Programm-, Daten- und Stacksegmente zur Durchsetzung unterschiedlicher Zugriffsarten oder Priviligierungsebenen.

1.3.1 Funktionsweise der Segmentierung

In Abbildung 1.1 ist der Aufbau der Segmentierung dargestellt.

Die **logische Adresse**[5] ist die Adresse, die den Programmen zur Verfügung gestellt wird – also die *virtuelle Adresse* der *virtuellen Speicherverwaltung*. Sie unterteilt sich in den **Sementselektor**[6] und den **Adressversatz**[7].

Über den Segmentselektor wird in der **globalen Deskriptortabelle**[8] der **Segmentdeskriptor**, [9] ermittelt.

Die *globale Deskriptortabelle* ist eine Datenstruktur, die die Eigenschaften verschiedener Speicherbereiche, auch **Segmente**[10] genannt, während der Programmausführung verwaltet. Dazu gehören die **Basisadresse**[11], die **Segmentlänge**[12] sowie Rechteprivilegien z. B. für das Lesen oder Beschreiben des Segments.

[4] auf englisch: segmentation
[5] auf englisch: logical adress
[6] auf englisch: segment selector
[7] auf englsich: Offset
[8] auf englisch: Global Descriptor Table (GDT)
[9] auf englisch: segment discriptor
[10] auf englisch: segments
[11] auf englisch: segment base adress
[12] auf englisch: segment size

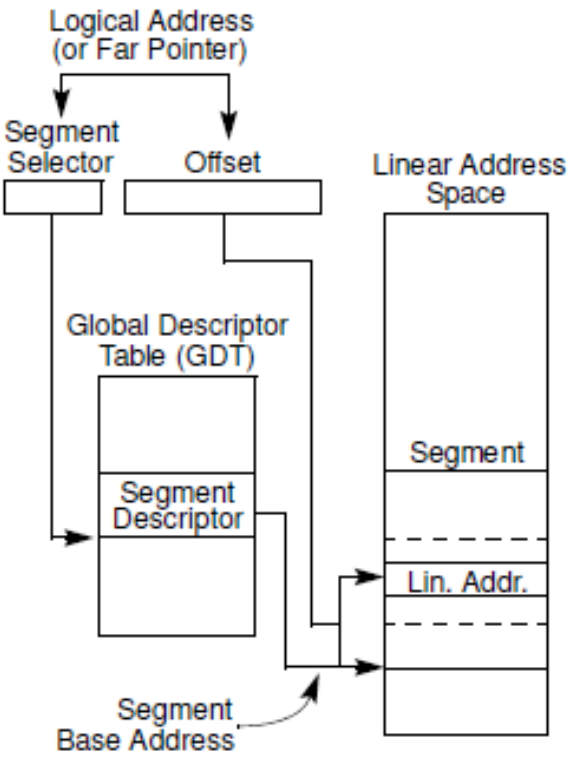


Abbildung 1.1: Segmentierung

Zur Vollständigkeite muss noch eine zweite Datenstruktur erwähnt werden. Die **lokale Deskriptortabelle**[13] kann ebenso wie die *globale Deskriptortabelle Segmente* verwalten, die jedoch privat und prozessspezifisch sind.

Aus dem *Segmentdeskriptor* können die *Basisadresse* und die *Segmentlänge* ausgelesen werden, wodurch das *Segment* eindeutig innerhalb des **linearen Adressraums**[14] bestimmt wird.

Durch die Verrechung der *Basisadresse* mit dem *Adressversatz* wird die *lineare Adresse* im *linearen Adressraum* eindeutig bestimmt. Falls kein *Paging* eingesetzt wird, entspricht die lineare Adresse direkt der physikalischen Adresse.

1.3.2 Segmentierung im IA32

Physikalische, lineare und logische Adressen

Im geschützten Modus der IA-32 Architektur findet der Zugriff auf jedes Byte über eine *logische Adresse* statt, die auf den *linearen Adressraum* abgebildet wird.

[13] auf englisch: local discriptor table (LDT)
[14] auf englisch: linear address space

Dabei besteht eine **logische Adresse** aus einem 16-Bit *Segmentselektor* und einen 32-Bit *Adressversatz*. Diese logische Adresse wird auf eine *lineare Adresse* abgebildet, welche wiederum durch *Paging* oder eins-zu-eins auf den *physikalischen Adressraum* abgebildet wird.

Der *lineare Adressraum* ist ein unsegmentierter Bereich mit einer Größe von 4 GByte. Dieser kann auf einen *physikalischen Adressraum* von bis zu 4 GByte abgebildet werden[15].

Globale Deskriptortabelle

Die *globale Deskriptortabelle* ist die grundlegende Datenstruktur der Segmentierung. In ihr können verschiedene Deskriptoren angelegt werden, um die einzelnen Segmente zu verwalten. Insgesamt haben 8192 Deskriptoren mit einer Größe von 8 Byte in ihr Platz. Um die *GDT* nutzen zu können, muss das **GDTR**-Register mit einer linearen Adresse, ab der die *GDT* beginnt, belegt werden.

Segmentselektoren

Ein *Segmentselektor* ist eine 16-Bit ID für ein Segment. Es verweist nicht direkt auf das *Segment*, sondern auf den *Segmentdeskriptor* der das *Segment* definiert. Wie in Abbildung 1.2 dargestellt, besteht ein *Segmentselektor* aus einem Index (Bit 3–15), durch den ein Deskriptor aus einer *Deskriptortabelle* bestimmt wird, einem Schalter zur Tabellenauswahl(Bit2 mit 0 = GDT, 1 = LDT) und einem Sicherheitslevel (Bit 0–1 mit 0 als privilegierter Stufe).

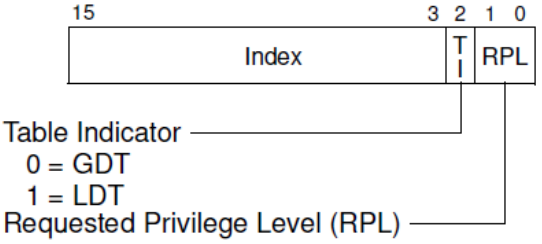


Abbildung 1.2: Segmentselektor

[15] Durch technische Erweiterungen wie die **Physical Address Extension(PAE)** kann ein größerer physikalischer Arbeitsspeicher angesprochen werden.

Segmentregister

Um Übersetungszeit zu sparen bietet der Prozessor Register um 6 *Segmentselektoren* zwischenzuspeichern. Jedes dieser *Segmentregister* unterstützt eine bestimmte Art des Speicherzugriffs(Code, Stack, oder Daten). Jedes *Segmentregister* hat einen „sichtbaren“ und einen „unsichtbaren“ Anteil. Der sichtbare Teil ist der *Segmentselektor*, der unsichtbare Teil enhält die *Basisadresse*, die *Segmentlänge* und die Zugriffsinformationen des Segments.

> Für jedes Programm sollten mindestens die Register für Code, Daten und Stack mit gültigen *Segmentselektoren* bestückt werden.

Segmentdeskriptor

Ein *Segmentdeskriptor* ist ein Eintrag in einer *globalen Deskriptortabelle* der dem Prozessor Informationen über die Größe und die Position eines Segmentes, sowie Informationen über die Zugriffsarten sowie Statusinformationen zur Verfügung stellt.

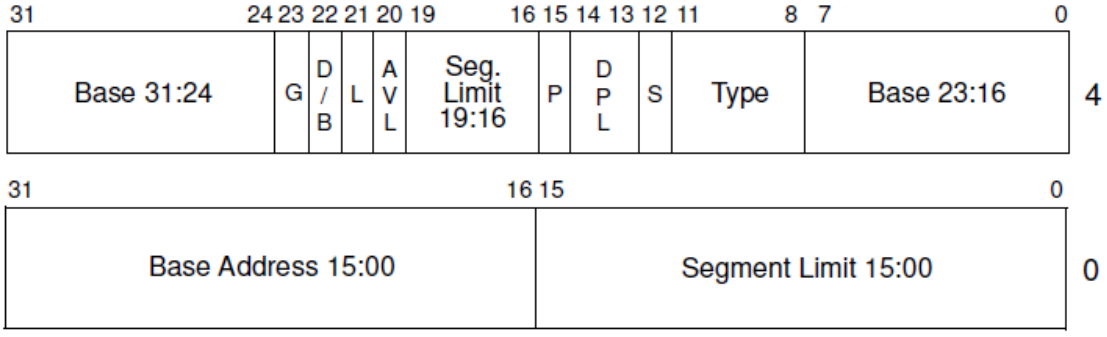


Abbildung 1.3: Aufbau Segmentdeskriptor

- **Limit**: Die *Segmentgröße* gibt die Größe des Segments an.
- **Base**: Die *Basisadresse* definiert den Start des Segments innerhalb des 4 GByte großen linearen Adressraums. [16]
- **Type**: Durch den Eintrag wird der Typ des Segmentes(in Abhängigkeit von S) bestimmt.
- **S**: Der Deskriptortyp gibt an, ob der *Segmentdeskriptor* für ein Systemsegment (S=0) oder ein Code- oder Datensegment(S=1) definiert ist.

[16] **Hinweis**: Aus Performanzgründen sollten Segmente immer auf 16-Byte Grenzen definiert werden.

- **DPL**: Der Deskriptorschutzlevel spezifiziert verschiedene Schutzlevel in einem Bereich von 0-3, wobei 0 der am meisten privilegierte Level ist.
- **P**: Dieses Bit zeigt an, ob das Segment im Speicher präsent ist (P=1) oder nicht (P=0).
- **D/B**: Die Standardoperationsgröße/Standardstapelgröße/Obere Grenze übernimmt verschiedene Funktionen, in Abhängigkeit des Typs des Segmentdeskriptors.[17]
- **G**: Das Granularitätsbit bestimmt die Granularität des Segmentlimits. Wenn G=0 ist, wird das Segmentlimit in Byte-Einheiten gezählt. Bei G=1 wird in 4-KByte-Einheiten gerechnet.
- **L**: Dieses Flag sollte auf 0 gesetzt werden.
- **AVL**: Bit 20 ist für Systemsoftware frei verwendbar.

Code- und Datensegmentdeskriptortypen

Einem *Segmentdeskriptor* können verschiedene *Code- und Datensegmentdeskriptortypen* zugeordnet werden, wenn das **S**-Bit auf 1 steht. Dabei sind 16 verschiedene Kombinationen aus Typ und Zugriffsberechtigungen möglich. Tabelle 1.1 zeigt eine Auswahl möglicher Kombinationen.

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
2	0	0	1	0	Data	Read/Write
		C	R	A		
8	1	0	0	0	Code	Execute-Only
10	1	0	1	0	Code	Execute/Read

Tabelle 1.1: Beispiel von Code- und Datensegmentdeskriptortypen

So lassen sich Datensegmente, die nur gelesen werden können, oder Codesegmente, die nur ausgeführt werden können, definieren.

Segmentierungsmodelle

Basic Flat Model In diesem Modell haben das Betriebssystem und die Programme Zugriff auf einen unsegmentierten Adressraum. Dabei verschleiert das Modell die Segmentierung soweit wie möglich, wie in Abbildung 1.4 zu sehen ist.

[17] **Hinweis**: Für 32-Bit Daten- und Segmentregister sollte das Bit gesetzt werden.

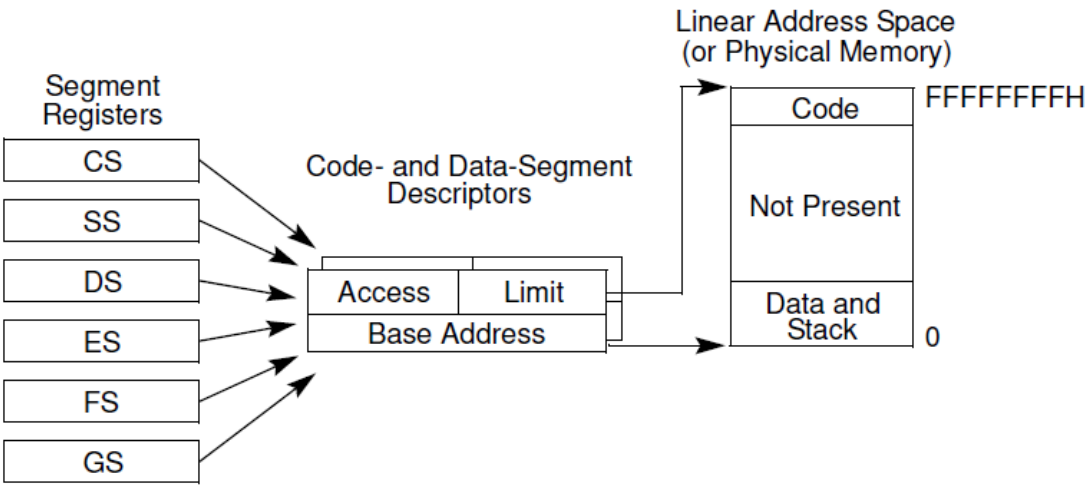


Abbildung 1.4: Basic Flat Model

Um das Basic Flat Model zu implementieren, müssen mindestens ein Code- und ein Datensegment definiert werden. Dabei werden sie auf einen gemeinsamen linearen Adressraum mit gleicher *Basisadresse* und *Segmentlänge* abgebildet, wodurch der Eindruck entsteht, es gebe nur ein Segment.

Anzumerken ist außerdem, dass keine Fehler generiert werden falls auf Adressen zugegriffen wird, die nicht physikalisch existieren.

Protected Flat Model Dieses Model ähnelt dem *Basic Flat Model* in großen Teilen, wobei jedoch Fehler generiert werden, um Zugriffe auf nicht existierenden physikalischen Speicher zu verhindern. Wie in Abbildung 1.5 zu sehen ist, werden mindestens zwei Segmente definiert, die nicht auf die gleichen *linearen Adressen* abgebildet werden.

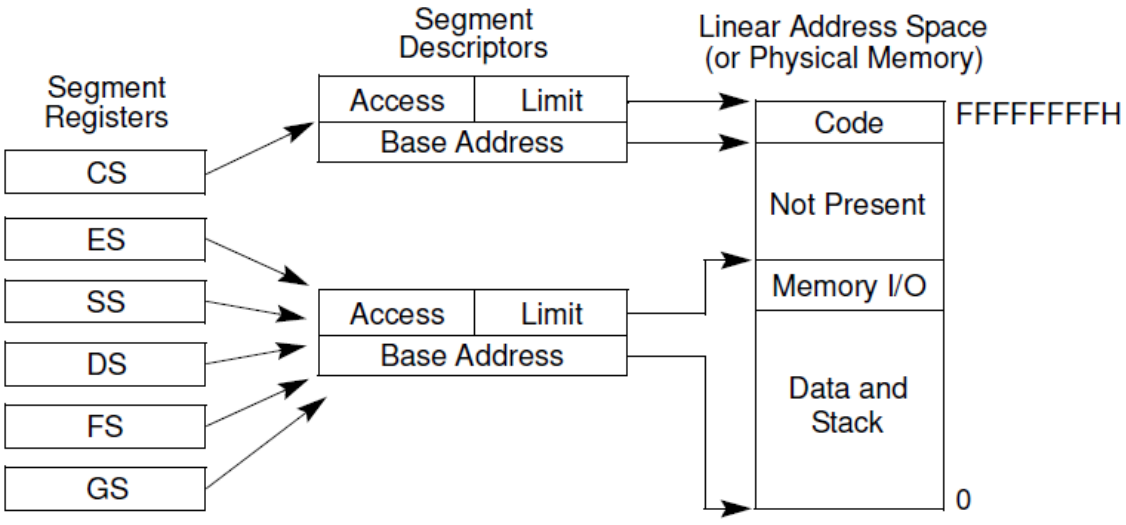


Abbildung 1.5: Protected Flat Model

Um die Sicherheit des Models weiter zu erhöhen, kann z. B. eine Isolation zwischen Benutzer und Administrator erreicht werden, indem 4 Segmente definiert werden: Code- und Datensegment für den Benutzer und Code- und Datensegment für den Kern.

> Dieses Model kann im Zusammenspiel mit einer einfachen Pagingstruktur eingesetzt werden, um das Betriebssystem vor Anwendungen zu schützen. Durch das Hinzufügen einer separaten Pagingstruktur für jeden Task und jeden Prozess kann außerdem verhindert werden, dass sich Applikationen gegenseitig beeinflussen.

Multi-Segment Model In diesem Modell erhält jedes Programm/jeder Task eine eigene *lokale Deskriptortabelle* und eigene Segmente. Diese Segmente können komplett privat sein oder mit anderen Programmen geteilt werden.

> Anzumerken ist, dass die meisten Konzepte des Speicherschutz in modernen Betriebssystemen durch *Paging* erledigt werden, so dass das *Multi-Segment Modell* nur selten zum Einsatz kommt und nur ein Minimalsatz von Segmenten übrig bleibt.

1.4 Paging

Das **Paging** ist eine Methode um virtuellen Speicher, also eine Abstraktionsschicht, zwischen *physikalischen Adressen* und den Programmen zu realisieren Dadurch kann z.B. das Problem der Speicherfragmentierung gelöst werden.

1.4.1 Paging Grundlagen

Das 32-bit Paging wird genutzt, um 32-bit lange *lineare Adressen* in *physikalische Adressen* zu übersetzen. Es wird eine Hierarchie von **Paging-Strukturen** genutzt, um die Adressen zu übersetzen. Das Kontrollregister CR3 enthält dabei die Adresse der ersten Paging-Struktur in der Hierarchie.

Um das 32-bit Paging einzuschalten, wird ein MOV Befehl auf das CR0 Register ausgeführt, um das CR0.PG Bit zu setzen. Vorher sollte sichergestellt werden, dass das CR3 Register die physikalische Adresse der ersten Paging-Struktur enthält.

Relevante Bits

- Der Bitschalter **PG** im Kontrollregister CR0 (bit 31) ist das Bit, das das Paging grundsätzlich ein- bzw. ausschaltet. Wenn CR0.PG = 0 wird kein Paging verwendet und der Prozessor behandelt alle linearen Adressen als wären sie physikalische Adressen.
- Bitschalter WP in CR0, das zur Absicherung der Paging Seiten dient.

Paging-Modi

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width	Page Sizes
None	0	N/A	N/A	32	32	N/A
32-bit	1	0	0	32	Up to 40	4 KB 4 MB

Tabelle 1.2: Die verschiedenen Paging Modi

In Tabelle 1.2 sind die verschiedenen Paging Modi dargestellt. Hierfür ist allerdings nur das 32-bit Paging relevant. Weitere Modi erweitern beispielsweise den physischen Adressraum, sind allerdings für dieses Projekt nicht von Bedeutung.

Paging-Modus Modifikatoren

Das Verhalten für die einzelnen Page-Modi wird durch sogenannte **Modifikatoren** gesteuert. Der einzige für das 32-bit Paging relevante Modifikator ist der Bitschalter **WP** im Kontrollregister CR0.

1.4.2 Paging im IA-32

Im Folgenden wird das Paging mit 4 KByte großen Seiten-Rahmen betrachtet. Für die Strukturierung gibt es dabei zwei verschiedene Arten von Seiten-Strukturen:

- PDE (Page Directory)
- PTE (Page Table)

Jede Art der Seiten-Strukturen (*PDE* oder *PTE*) ist in allen Modi 4 KByte groß und kann 1024 mal 4 Byte große Einträge beinhalten.

Ein Eintrag in der *PDE* beinhaltet unter anderem eine *physikalische Adresse*, an der sich eine *PTE* befindet. Ein Eintrag in der *PTE* beinhaltet unter anderem die *physikalische Adresse* des Seiten-Rahmens. Die erste Seiten-Struktur(*PDE*) befindet sich an der *physikalischen Adresse* mit dem Wert vom Register *CR3* (Bits 31..12).

Bit-Belegung des CR3 Registers

Die Bit-Belegung[18] des CR3 Registers und der Seitenstrukturen können der Tabelle 1.3 entnommen werden.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored			**CR3**
Bits 31:22 of address of 2MB page frame										Reserved (must be 0)				Bits 39:32 of address[2]					PAT	Ignored			G	1	D	A	PCD	PWT	U/S	R/W	1	**PDE: 4MB page**
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	**PDE: page table**
Ignored																															0	**PDE: not present**
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	**PTE: 4KB page**
Ignored																															0	**PTE: not present**

Tabelle 1.3: Bit-Belegung CR3, PDE und PTE

Reserved Bits müssen mit 0 belegt sein.

CR3: Bits 31..12: Physikalische Adresse des PDE Eintrags. Wird mit 1000h multipliziert für tatsächliche Adresse.

PDE: Bit 0 (Present) muss mit 1 belegt sein um einen PTE Eintrag zu referenzieren. Bit 7 (Page Size) muss mit 0 belegt sein für 4 KByte Paging. Bit 31..12: Physikalische Adresse des PTE Eintrags.

PTE: Bit 0 (Present) muss mit 1 belegt sein für 4 KByte Paging. Bit 31..12: Physikalische Adresse des Seiten-Rahmens.

Adressberechnung beim Paging

Beim Übersetzen einer *linearen Adresse* zu einer *physikalischen Adresse* geht der Prozessor iterativ vor:

[18] Lediglich die relevanten Bits und deren Belegung werden für das 4 KByte Paging beschrieben. Das Cacheverhalten und die Zugriffskontrolle werden nicht berücksichtigt.

- Zunächst wird der Eintrag aus der ersten Seiten-Struktur (*PDE*) geholt. Dazu werden die ersten 10 Bits (31..22) der *linearen Adresse* als *Adressversatz* zu CR3 genutzt.
- Die nächsten 10 Bits (21..12) werden als *Adressversatz* zu der hinterlegten Adresse im *PDE* genutzt, um den Seiten-Rahmen zu bestimmen.
- Die *physikalische Adresse* ist dann die Adresse des Seiten-Rahmens mit den verbleibenden Bits der *linearen Adresse* (11..0) als *Adressversatz*.

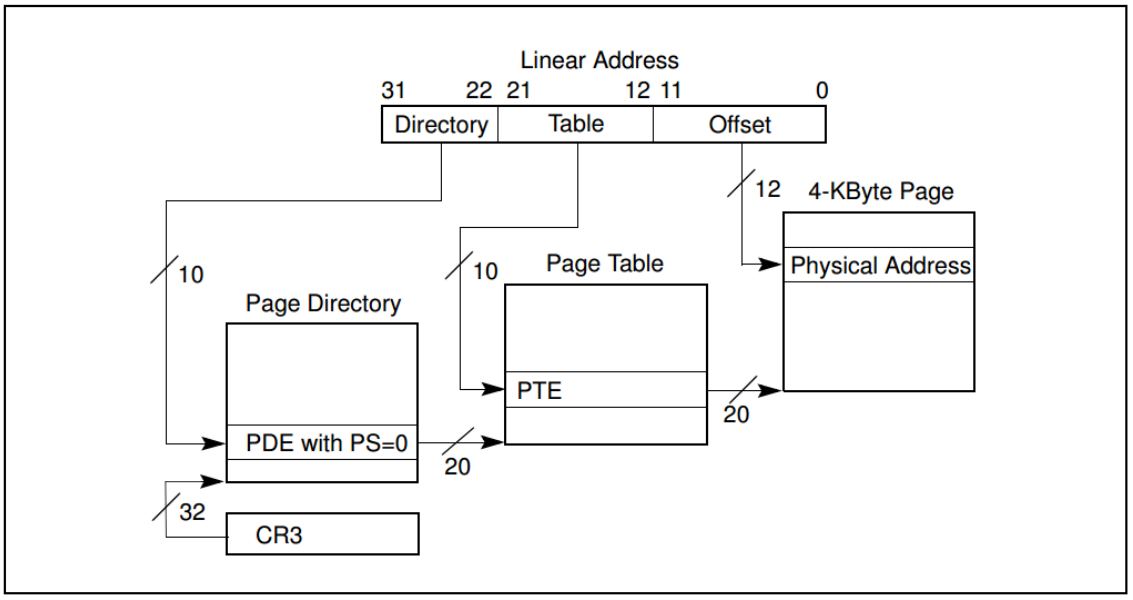


Abbildung 1.6: Umwandlung der linearen Adressen zu physikalischen Adressen

TLB

Um die Übersetzung der *linearen Adressen* zu *physikalischen Adressen* zu beschleunigen werden die Übersetzungen in sogenannten **TLB** (Translation Lookaside Buffer) zwischengespeichert. Dabei kann nicht immer davon ausgegangen werden, dass es einen *TLB* gibt. Sollte ein *TLB* vorhanden sein, untersucht der Prozessor beim Übersetzen einer *linearen Adresse* in eine *physikalische Adresse*, ob es einen Eintrag für die Übersetzung im *TLB* gibt. Wenn es einen gibt, wird dieser direkt zurückgegeben. Sollte es keinen geben, wird er wie in Abschnitt 1.4.2 beschrieben im Speicher gesucht, dem *TLB* hinzugefügt und zurückgegeben. Beim Ausgeben der Adresse aus dem *TLB* beachtet der Prozessor nicht Veränderungen, die in der Zwischenzeit vom der Software aus an der Speicherstruktur stattgefunden haben. Daher ist es Aufgabe der Software bei Änderungen an der Speicherstruktur die Einträge im *TLB* auf ungültig zu setzen oder den *TLB* zu leeren.

1.4.3 Fazit

Die IA-32 Architektur besitzt eine Menge Features um eine moderne Speicherverwaltung zu realisieren. Allein schon die Übersetzung der linearen zu physikalischen Adressen würde rein softwareseitig viel langsamer laufen. Das Betriebssystem muss nun lediglich noch eine Buchführung für den verbrauchten Speicher führen.

2 Speicher-Allokation

2.1 Einleitung

Sobald ein Prozess dynamischen Speicher (Heap) benötigt, wendet er sich an einen sogenannten **Allokator**. Ein Allokator verwaltet einen Speicherbereich, der ihm zuvor vom Kern zugeteilt wurde. Er ist dafür verantwortlich, den ihm gegebenen Speicherbereich möglichst effizient aufzuteilen und dafür zu sorgen, dass für benutzte Speicherbereiche über das Paging physikalische Seiten zugewiesen wurden. Wenn ein Prozess einen Speicherbereich nicht mehr benötigt, muss dieser für neue Anforderungen freigegeben werden und ggf. müssen Seiten über das Paging wieder freigegeben werden. Der Allokator muss dafür eine eigene Strategie und eigene Strukturen besitzen. In unserer Implementierung haben wir uns für die Verwaltung der Seiten für den **Buddy-Algorithmus** entschieden. Der Algorithmus und die dazugehörigen Funktionen werden im Folgenden vorgestellt und auch anhand des Quellcodes erläutert.

2.2 Algorithmen

Zum Ermitteln von Speicheradressen und Blockgrößen gibt es viele verschiedene Algorithmen, die sich teilweise stark in Komplexität und Effizienz unterscheiden. In diesem Kapitel werden zwei einfache Algorithmen vorgestellt.

2.2.1 First-Fit-Algorithmus

Beim First-Fit-Algorithmus wird eine Liste[19] gepflegt, in der freie Blöcke aus dem zu verwaltenden Speicher eingetragen werden. Wird ein Block B_1 angefordert, wird der erste Block aus dieser Liste geliefert, der mindestens die Größe des angeforderten Speichers hat. Hat dieser Block B_1 jedoch signifikant mehr Kapazität

[19] Oft als free-list bezeichnet

als notwendig, kann dieser noch einmal gesplittet werden. Der übrige, weiterhin freie Teil des Blocks B_1 würde als neuer Block B_2 in die Liste eingegliedert werden.

Die Blöcke in dieser Liste können nach unterschiedlichen Kriterien geordnet sein, z. B. nach ...

- Größe,
- Adresse oder
- der vergangenen Zeit seit seiner letzten Verwendung

Der First-Fit-Algorithmus ist dabei sehr einfach und arbeitet in akzeptabler Zeit beim Allozieren. Allerdings kann das Freigeben der Blöcke nicht in der selben Zeit geschehen. Hier muss darauf geachtet werden, dass der Block an die richtige Position in der Liste einsortiert wird um die gewählte Ordnung aufrecht zu erhalten. Dies kann, je nach verwendeter Ordnung, sehr aufwendig sein!

2.2.2 Buddy-Algorithmus

Um die korrekten Speicheradresse innerhalb einer Seite zu ermitteln wird der Buddy-Algorithmus verwendet. Dieser Algorithmus sieht im Wesentlichen ein sukzessives halbieren des Speichers innerhalb einer Seite vor, bis ein Block mit passender Kapazität gefunden wurde. Das Vorgehen wird im Einzelnen nun detaillierter vorgestellt.

Für den Buddy-Algorithmus benötigt man zwei Parameter. Zum Einen die Größe des größten zusammenhängenden Blocks (S_{max}) und zum Anderen die maximale Anzahl der Unterteilungen dieses Blocks (p). Somit ergibt sich die Größe des kleinsten Blocks: $S_{min} = \frac{S_{max}}{2^p}$. Die maximale Anzahl von Blöcken (der Größe S_{min}) ist somit 2^p. Mögliche Blockgrößen sind $\{S_{min} \times 2^0, S_{min} \times 2^1, \ldots, S_{min} \times 2^p\}$.

Soll nun Speicher der Größe S_{req} angefordert werden, muss zuerst berechnet werden, welche Blockgröße (S_B) benötigt wird. Hierfür bedient man sich folgendes Algorithmus:

$$
\begin{aligned}
&S_B \;:=\; S_{max} \\
&i \;:=\; p \\
&\texttt{SOLANG}\ \left(S_{req} \;\leq\; \tfrac{S_B}{2} \;\wedge\; i \;>\; 0\right) \\
&\qquad S_B \;:=\; \tfrac{S_B}{2} \\
&\qquad i \;:=\; i \;-\; 1
\end{aligned}
$$

Nun kann der erste freie Block der Größe S_B belegt und die Startadresse des Blocks zurückgegeben werden.

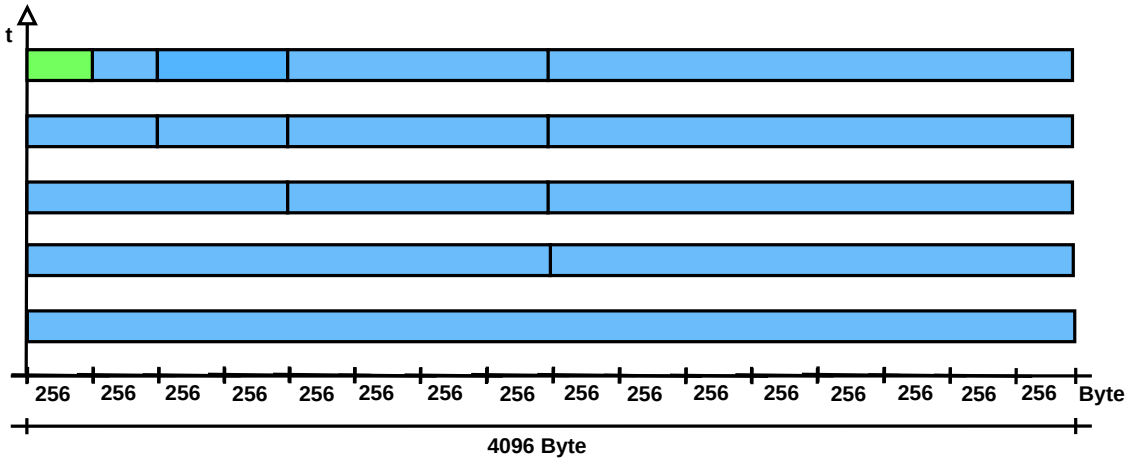


Abbildung 2.1: Blockaufteilung nach dem Buddy-Algorithmus

In Abbildung 2.1 wird der oben beschriebene Fall noch einmal dargestellt. Mit $S_{max} = 4096Byte$, $S_{min} = 256Byte$, $p = 4$. Es soll ein Block mit 256 Byte alloziert werden. An diesem Beispiel ist gut zu erkennen, dass der anfänglich große Block von 4 Kilobyte in immer kleiner werdende Teile zerlegt wird, bis eine passende Größe erreicht ist.

Irgendwann einmal ist der Zeitpunkt erreicht, an dem ein Programm allozierten Speicher wieder freigeben möchte (oder muss). Beim Freigeben wird nicht nur der belegte Block im Allokator freigegeben, er wird auch mit seinem Buddy[20] wieder zu einem größerem Buddy-Block zusammen gefasst, falls dieser auch frei ist. Dieses Vorgehen würde in dem Beispiel aus Abbildung 2.1 dazu führen, dass nach dem Freigeben des allozierten Blocks die Seite zu einem einzigen Block der Länge 4 Kilobyte zusammengefasst würde.

Beim Allozieren arbeitet der Buddy-Algorithmus trotz der, im Vergleich zum First-Fit-Algorithmus, etwas aufwändigeren Suche immer noch effizient. Die Suche nach einem passenden Block sollte nach etwa $\log_2(p)$ Schritten abgeschlossen sein. Beim Freigeben ist der Buddy-Algorithmus dabei effizienter als der First-Fit-Algorithmus, da das aufwändige Einsortieren in die Liste entfällt. Lediglich das Zusammenfügen von Buddy-Blöcken muss durchgeführt werden, was allerdings in linearer Zeit möglich ist.

2.2.3 Wahl eines Algorithmus

Aufgrund des einfachen und schnellen Freigebens von Speicher sowie der logarithmischen Zeit beim Ermitteln eines Blocks passender Größe, wird der Buddy-Algorithmus für die Entwicklung unseres Betriebssystems verwendet.

[20] Ein Block kann ausschließlich mit seinem Buddy zusammengefasst werden

2.3 Grundinformationen

Für die Implementierung des Allokators spielen folgende Headerdateien und Klassen eine Rolle:

- Allcator.cc & Allocator.h
- AllocatorConstants.h - Definition fester Konstanten
- BuddyPageTable.h
- BuddyPageUsage.cc & BuddyPageUsage.h
- heap.h & heap.cc

Im Folgenden bezeichnet uint32 einen Alias für **unsigned long**.

2.4 Allokator

2.4.1 Konstruktor

Um den Allokator zu nutzen muss zunächst eine Instanz eines Allokators erstellt werden. Dabei muss zu Beginn klar sein, wie viel Speicher diese Instanz verwalten soll. Dazu wird die erste Adresse des Speicherbereichs (**heapAddress**) und die Größe, die verwaltet werden soll (**sizeInBytes**) angegeben.

Listing 2.1: Konstruktor-Operation des Allokators

```
Allocator(void* heapAddress, uint32 sizeInBytes);
```

Um Speicher zu allozieren und wieder frei zu geben stellt der Allokator zwei Funktionen zur Verfügung, **allocate** und **free**

Listing 2.2: alloc und free Operationen des Allokators

```
void* allocate(uint32 size);
void free(void* address);
```

Beim Aufruf der allocate Funktion muss die Größe des Speichers angegeben werden, der alloziert werden soll, beim free wiederum ein Zeiger auf die erste Adresse des zuvor allozierten Bereiches. Der Allokator benutzt zur Verwaltung eine Struktur aus **BuddyPageTables** und **BuddyPageUsages**, die im Folgenden genauer vorgestellt werden.

2.4.2 Destruktor

Im Destruktor werden alle Seiten, die der Allokator zuvor vom Paging angefordert hatte, wieder freigegeben.

2.4.3 BuddyPageTable-Struktur

Eine Allokator-Instanz besitzt zum Zugriff auf die Verwaltungsstrukturen ein Attribut (**firstBuddyPageTable**) um einen initialen Zugriffspunkt zu ermöglichen. Da die BuddyPageTables als eine verkettete Liste aufgebaut sind, kann man auf weitere BuddyPageTables mit dem Pointer **nextBuddyPageTable** zugreifen (siehe Listing 2.3).

Außer einem Pointer auf die nächste BuddyPageTable besitzt eine BuddyPageTable nur ein Array **buddyPageUsages** vom Typ **BuddyPageUsage**. Dieser Typ wird eingesetzt, um die Nutzung der einzelnen Seiten zu speichern – dazu im nächsten Abschnitt mehr. Die Konstante **BUDDYPAGESINTABLE** legt fest, wie viele solcher Seiten in einer BuddyPageTable verwaltet werden können. Diese Konstante berechnet sich aus der Größe einer Seite (PAGESIZE – in unserem Fall 4096 Bytes), der Größe des Pointers auf die nächste BuddyPageTable (sizeof(void*)) und der Größe einer BuddyPageUsage Struktur (sizeof(BuddyPageUsage)). Die Konstante und somit die Größe des Arrays ist damit abhängig von den im System genutzten Werten. Durch die Berechnung der Konstanten kann sichergestellt werden, dass immer eine komplette Seite für die Verwaltung eingesetzt wird.

Listing 2.3: Headerdatei der BuddyPageTable

```
1 const uint32 BUDDYPAGESINTABLE =
2     (PAGESIZE - sizeof(void*)) / sizeof(BuddyPageUsage);
3
4 struct BuddyPageTable {
5         BuddyPageUsage buddyPageUsages[BUDDYPAGESINTABLE];
6         BuddyPageTable* nextBuddyPageTable;
7 } __attribute__((packed));
```

2.4.4 BuddyPageUsage-Klasse

Um die der einzelnen Seiten zu dokumentieren wird die **BuddyPageUsage**-Klasse eingesetzt. Sie hat ein Attribut **address**, welches auf den Speicher im

linearen Speicherbereich zeigt, welche von dieser BuddyPageUsage verwaltet wird. Um den Status der Blöcke dieser Seite zu speichern werden zwei Arrays genutzt, welche jeweils so viele Bits besitzen, wie die Seite Blöcke hat. Im Array usedParts steht ein Bit entweder für benutzt (1) oder für frei (0). Das Array neighbors wird genutzt, um anzugeben, ob der benutze Block im nächsten Block weitergeht (1).

Ein Beispiel:

```
Pagesize = 4 kB, kleinster Block = 256 B, Anzahl der Blöcke = 16
Nach Anlegen der BuddyPageUsage:
  usedParts 00000000 00000000
  neighbors 00000000 00000000

Allozieren von 100 B (1 Block)
  usedParts 10000000 00000000
  neighbors 00000000 00000000

Allozieren von 350 B (2 Blöcke)
  usedParts 10110000 00000000
  neighbors 00100000 00000000

Allozieren von 2000 B (8 Blöcke)
  usedParts 10110000 11111111
  neighbors 00100000 11111110
```

Listing 2.4: Headerdatei der BuddyPageUsage

```
1  class BuddyPageUsage {
2  public:
3          BuddyPageUsage();
4
5          AllocateReturn allocate(uint32 neededBytes,
6                                  bool continuesOnNextPage);
7          bool free(void* address);
8
9          bool isEmpty() const;
10         bool isValid() const;
11         void createAtAddress(void* address);
12         void setInvalid();
13
14         void* getAddress() const {
```

```
15                  return address;
16          }
17
18  private:
19          void* address;
20          unsigned char usedParts[BUDDYBITFIELDSIZE];
21          unsigned char neighbors[BUDDYBITFIELDSIZE];
22
23          void setUsed(uint32 base, uint32 numberOfParts,
24                       bool continuesOnNextPage);
25  } __attribute__((packed));
```

Die allocate-Methode der Klasse BuddyPageUsage gibt einen speziellen Typ zurück: den **AllocateReturn**. AllocateReturn ist ein C++ Struct mit zwei Attributen. Das Attribut success vom Typ bool signalisiert, ob der angeforderte Speicher alloziert werden konnte. In diesem Fall enthält der void-Pointer address die Startadresse des ersten, neu allozierten Blocks.

Der Inhalt von address ist für den Fall, dass kein Speicher alloziert wurde, nicht spezifiziert.

2.4.5 Vorgehen beim Allozieren

Wie bereits erwähnt arbeitet der Allokator nach dem Buddy-Algorithmus. Bis dieser allerdings zum Tragen kommt, muss noch einiges berücksichtigt werden.

Zunächst wird beim Anlegen eines Heap-Bereichs ein eigener Allokator erstellt. Um einen Heap-Bereich nutzen zu können, muss die Datei heap.h inkludiert werden. In dieser Header-Datei werden die gängigen Operatoren für dynamischen Speicher deklariert.

Listing 2.5: heap.h

```
1  void* operator new(unsigned int size);
2  void* operator new[] (unsigned int size);
3  void operator delete(void* p);
4  void operator delete[] (void* p);
```

Beim Aufruf des Operators new[21] wird der für den Heap-Bereich verantwortliche Allokator genutzt und dessen allocate-Methode aufgerufen.

[21] entsprechend natürlich auch bei new[]

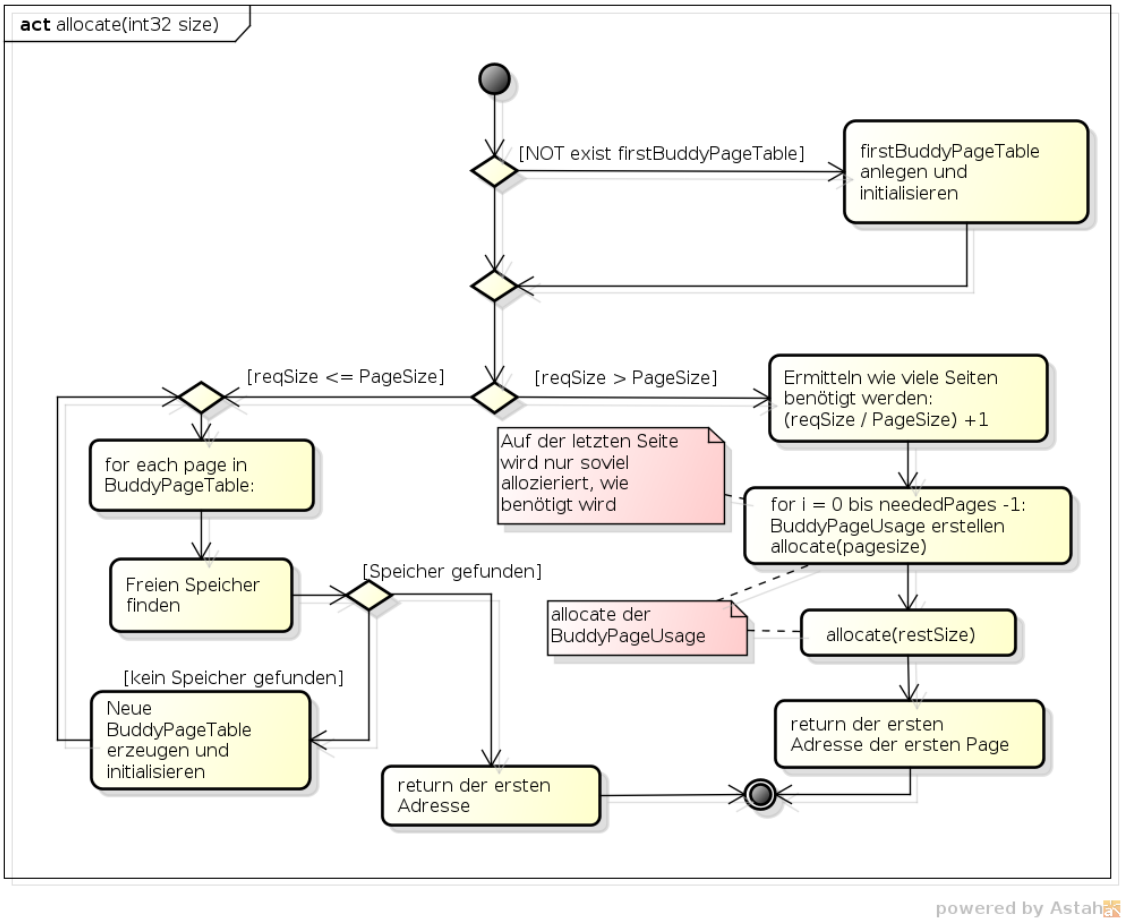


Abbildung 2.2: allocate

In Abbildung 2.2 ist der Ablauf der allocate-Methode dargestellt. Hier ist gut zu sehen, dass es bei einer angeforderten Größe über der Seitengröße zu einer Rekursion kommt, bis genügend Seiten alloziert wurden. Sehr abstrakt dargestellt ist allerdings die Aufgabe des Findens von freiem Speicher innerhalb einer Seite.

Für jede Seite gibt es eine BuddyPageUsage. In dieser ist vermerkt, welche Blöcke innerhalb der Seite bereits vergeben und welche noch frei sind. Daher gibt es in dieser Klasse ebenfalls eine allocate- und eine free-Methode, welche die freien Blöcke verwalten.

Dabei arbeitet die allocate-Methode der BuddyPageUsage jetzt nach dem Prinzip des Buddy-Algorithmus. Es wird zunächst über sukzessives Teilen ermittelt, wie viele minimale Blöcke[22] für die angeforderte Größe benötigt werden. Im Anschluss wird überprüft, ob in der aktuellen BuddyPageUsage noch ein zusammenhängender Block passender Größe vorhanden ist. Um dies zu ermitteln, werden zwei Attribute der BuddyPageUsage benötigt. In dem Attribut usedParts wird für jeden minimalen Block ein Bit vorgehalten, das signalisiert, ob dieser minimale Block bereits vorhanden ist. Für jeden dieser Blöcke gibt in dem Attribut neighbors ebenfalls ein Bit an der selben Stelle wie in usedParts. Daraus ergeben sich für eine Stelle an i-ter Position folgende Informationen:

usedParts	neighbors	Bedeutung
0	0	Block ist frei, ohne Nachfolger
0	1	ungültig
1	0	Block ist belegt, ohne Nachfolger
1	1	Block ist belegt, mit Nachfolger

Tabelle 2.1: Bedeutung usedParts und neighbors

Dabei ist zu beachten, dass ein Nachfolger immer zusammenhängende Blöcke kennzeichnet. Wird ein passender Block gefunden, so wird dessen Startadressen in das AllocateReturn geschrieben und dessen success-Attribut auf true[23] gesetzt. Außerdem wird der verwendete Speicherplatz als belegt gekennzeichnet. Falls kein entsprechender Block zur Verfügung steht, wird dieses Attribut auf false belassen. Durch Abfragen des success-Attributes kann nun der Allokator eindeutig erkennen, ob in der angefragten Seite noch freier Platz vorhanden war.

2.4.6 Vorgehen beim Freigeben

Das Freigeben bereits belegten Speichers ist weniger aufwendig als das Allozieren. Hierbei wird dem Allokator die Adresse des Blocks als Parameter der

[22] Ein minimaler Block ist ein Block mit der kleinst möglichen Größe
[23] default: false

free-Methode übergeben. Zu dieser Adresse wird über Iteration über die BuddyPageTables und schließlich in den einzelnen BuddyPageUsages der minimale Block ermittelt, dessen Adresse dem free übergeben wurde. Ist dieser Block einmal in einer BuddyPageUsage gefunden, werden ab dieser Position in usedParts und neighbors solange Blöcke freigeben (Bit im usedParts-Attribute auf 0 gesetzt), bis zu einem usedParts-Bit ein 0-Bit in neighbors gefunden wird. Dies erledigt die free-Methode der jeweiligen BuddyPageUsage. Falls sich der freizugebende Block über mehrere Seiten erstreckt, wird von der free-Methode ein bool-Wert zurück geliefert. In diesem Fall ist dieser auf „wahr“ gesetzt. Die free-Methode des Allokators ruft dann die free-Methode der nächsten BuddyPageUsage auf. Ein weiteres Beispiel für eine Belegung liefert Tabelle 2.2:

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
usedParts	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0	0
neighbors	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0

Tabelle 2.2: Beispiel usedParts und neighbors

Hier ist zu erkennen, dass die minimalen Blöcke an Position 0 – 3 über das neighbor-Bit verbunden sind. Beim Freigeben des Blocks an Position 0 würden daher auch die Blöcke 1, 2 und 3 freigegeben werden. Anders ist es bei den Blöcken an Position 8 – 11. Während die Blöcke 10 und 11 tatsächlich über das neighbor-Bit verbunden sind und nur gemeinsam freigegeben werden können, sind die Blöcke an Position 8 und 9 eigenständige Blöcke, die unabhängig voneinander sind.

Nachdem das Freigeben des Speicherbereichs erledigt ist, muss auch noch geprüft werden, ob einzelne Seiten nun wieder komplett unbenutzt sind. In diesem Fall kann die Seite auch im Paging freigeben werden. Dies geschieht in der free-Methode des Allokators. Hierzu wird geprüft, ob die BuddyPageUsage, in der gerade Speicher freigegeben wurde, nun unbenutzt ist. Ist dies der Fall, wird die Seite auch im Paging freigegeben. Ist im Anschluss die gesamte BuddyPageTable leer, wird diese gelöscht.

2.4.7 Speicherausnutzung

Der im Betriebsystem implementierte Buddy-Algorithmus benötigt selbst Speicher, um seine Informationen ablegen zu können. Dies betrifft vor allem die BuddyPageTables[24] sowie die darin enthaltenen BuddyPageUsages[25].

[24] In den folgenden Formeln als BPT bezeichnet
[25] In den folgenden Formeln als BPU bezeichnet

Unter Berücksichtigung der zum Zeitpunkt der Dokumentenerstellung eingestellten Größen lässt sich eine Hochrechnung zum verbrauchten Speicher des Buddy-Algorithmus anstellen. Angenommene Größen sind:

- Größe einer Seite (PageSize) = 4096 Byte
- Anzahl minimale Buddy-Blöcke einer Seite (BuddyParts) = 1024 Byte
- Zeiger auf eine Adresse[26] = 4 Byte

Aufgrund dieser Annahmen und der implementierten Strukturen lässt sich folgende Rechnung aufstellen:

$$percentageOfMememoryUsage = \frac{sizeof(BPU) * BPUsInBPT + NextBPT*}{PageSize}$$

$$mit$$

$$\begin{aligned} sizeOf(BPU) &= \frac{2 * BuddyParts}{8} + address* \\ &= \frac{2 * 1024}{8} + 4 \\ &= 260 \end{aligned}$$

$$und$$

$$\begin{aligned} BPUsInBPT &= \frac{PageSize - NextBPT*}{sizeOf(BPU)} \\ &= \frac{4096 - 4}{260} \\ &= 15,74 \end{aligned}$$

Diese 15,74 werden abgerundet, da nur ganze BPUs abgelegt werden.

Dies entspricht einer Speicherausnutzung von:

$$\frac{260*15+4}{4096} = 95,3\%$$

an Speicherverbrauch. Daraus wiederum ergibt sich ein Speicherverschnitt von 4,7 %.

2.5 Test des Allokators

Der implementierte Code wurde außerhalb der virtuellen Bochs-Umgebung getestet, da zum Zeitpunkt der Implementierung das Paging noch nicht vollständig zur Verfügung stand.

[26] Wie in C++ üblich mit einem * gekennzeichnet

Zum Testen wurde eine separate main()-Methode entworfen, in der der jeweils aktuelle Code zur Ausführung gebracht und dessen Aktivitäten auf der Konsole ausgegeben wurden.

Das Beispiel aus Kapitel 2.4.4 zeigt beispielsweise einen Teil der Ausgabe nach dem Allozieren von Speicher unterschiedlicher Größen. Hier wird dargestellt, wie der Speicher beim Allozieren fragmentiert wird. Diese Tests wurden auf unterschiedliche Fälle ausgeweitet. So konnte auch das Verhalten beim Freigeben von Speicher beobachtet werden.

3 Paging-Algorithmus

3.1 Datenstruktur zur physikalischen Speicherverwaltung

Durch die *physikalische Speicherverwaltung* wird das Allozieren und Freigeben von *physikalischem Speicher* ermöglicht. Dabei speichert eine Datenstruktur in der *physikalische Speicherverwaltung* welche Speicherbereiche im *physikalischen Speicher* frei sind. Die verwalteten Speicherbereiche haben eine Seitengröße von 4KB.

3.1.1 Übersicht möglicher Datenstrukturen

Um eine Datenstruktur zur physikalischer Speicherverwaltung zu implementieren, gibt es verschiedene Möglichkeiten. Im Folgenden sollen einige kurz vorgestellt werden.

Bitmap

Eine einfache Verwaltung der *physikalischen Adressen* lässt sich mit einer **Bitmap** realisieren. Dazu wird ein Bit an Information für jede Seite gespeichert. Dieses Bit gibt an, ob die Seite belegt oder frei ist. Der Vorteil an der *Bitmap* ist, dass der von der *Bitmap* selbst verwendete Speicher zur Laufzeit konstant ist. Dadurch kann es nicht zu einer Endlosschleife kommen, falls der Speicherstruktur selbst der Platz für weitere Informationen ausgeht und sie weiteren von sich selbst anfordern will. Ferner ist es möglich einen kontinuierlichen Speicherbereich zur Verfügung zu stellen. Ein Nachteil der Bitmap ist, dass sie keine zusätzlichen Informationen aufnimmt. Dadurch ist es schwer möglich *gemeinsam genutzten Speicher* zu realisieren, da nicht verwaltet werden kann, wieviele *lineare Adressen* auf diesen Speicher verweisen.

Verkettete Liste

Ferner können die Seiten durch eine **verkettete Liste** verwaltet werden. Dabei verwaltet ein Listenelement den Anfang von freiem Speicher und die Länge des fortlaufenden freien Speichers.

Beim Anfordern von Speicher müssen diese Elemente entfernt bzw. verkleinert werden. Beim Freigeben von Speicher werden sie vergrößert oder gegebenenfalls neu angelegt. Ein Nachteil der Struktur ist, dass man für sie dynamisch Speicherplatz allozieren muss und es so zu Problemen mit Endlosschleifen kommen kann.

Stack

Eine weitere Datenstruktur ist der **Stack**. In ihm werden alle freien Seiten verwaltet. Wird eine Seite angefordert wird sie vom Stack genommen. Beim Freigeben einer Seite wird sie wieder auf den Stack gelegt. Ein Vorteil des Stacks ist, das die Anforderung einer freien Seite in konstanter Zeit geschehen kann.

Ein Nachteil ist, dass sich die Struktur dynamisch verändert und nicht immer konstant ist. Dadurch muss sich damit beschäftigt werden, dass die Struktur entsprechend wachsen kann und der Speicher nicht ausgeht. Ebenso wie bei den bisherigen Datenstrukturen können in einem *Stack* keine Zusatzinformationen verwaltet werden.

Tabelle

Eine weitere Möglichkeit ist die Verwaltung in einer Tabelle. In ihr können alle physikalischen Speicherblöcke mit Zusatzinformationen verwaltet werden.

Dies ist z. B. dafür nötig, um zukünftig mehrere Prozesse verwalten zu können. Durch die Zusatzinformationen können z. B. Speicherblöcke als *gemeinsam genutzter Speicher* markiert werden.

Ferner könnten die Anzahl der Prozesse, die auf den Block zugreifen, mitgezählt werden, wodurch *gemeinsam genutzer Speicher* wieder frei gegeben werden kann. Ansonsten verhält sich eine Tabelle wie eine *Bitmap*, wobei natürlich, je nach dem Umfang an Zusatzinformationen, die Größe der festen Datenstruktur ansteigt.

3.1.2 „physical memory management table(PmmTable)“

Um die Implementierung zu erleichtern und nicht mit den Problemen einer dynamisch wachsenden Struktur umgehen zu müssen, wird die Tabelle in Form eines Arrays mit einer festgelegten Größe erzeugt.

Dabei wird die Größe abhängig von dem zur Verfügung stehenden *physikalischen Speicher* gewählt. Um die Information über die Speichergröße zu erhalten, werden die Informationen aus einer Boot-Struktur, den **E820-Entries**, entnommen, die beim Start dem Kernel übergeben werden.

Außer den eigentlichen Einträgen der Speicherblöcke, werden Zusatzinformationen wie Speichergröße, Kernelgröße und Blockanzahl erfasst, da sie noch für andere Verwaltungseinheiten benötigt werden. So müssen für die `PmmTable` an sich *lineare Adressen* auf den *physikalischen Speicher* abgebildet werden, da sie selbst ihre Einträge im Speicher ablegen muss. Dazu muss die Größe des Speichers bekannt sein, um die Größe der `PmmTable` zubestimmen.

Die Einträge an sich, die `PmmTableEntries`, repräsentieren jeweils einen 4KB Speicherblock im *physikalischen Speicher* und reichern ihn mit Zusatzinformationen an. So kann die Anzahl der *linearen Adressen*, die auf ihn zeigen, mitgezählt werden. Dadurch lässt sich „gemeinsamer Speicher“ realisieren, der auch wieder freigegeben werden kann. Ferner lassen sich durch weitere Bits Zusatzinformationen anlegen, die sich in der weiteren Entwicklung[27] nützlich erweisen können. In der Abbildung 3.1 ist die Bitbelegung der Einträge beschrieben. Auf Basis dieser Bitbelegung lassen sich viele verschiedene sinnvolle Kombinationen bilden, um die Blöcke im physikalischen Speicher zu charakterisieren. Einen Überblick dazu gibt Abbildung 3.2.

Ferner gibt das Klassendiagramm in Abbildung 3.1 einen genauen Blick in den Aufbau der Datenstruktur.

Die Operationen, die in der Abbildung zu sehen sind, dienen dazu, die `PmmTable` zu initialisieren, Bereiche im *physikalischen Speicher* als verfügbar oder nicht verfügbar zu deklarieren, sowie Bereiche für den Kernel festzulegen. Ferner werden die wichtigsten Operationen `allocBlock` und `freeBlock` spezifiziert, mit denen Speicher angefordert und freigegeben werden kann. Bei der Implementierung dieser beiden Operationen, veranschaulicht in den Aktivitätsdiagrammen 3.2 und 3.3, wird die Variable `_free_start_search` verwendet, um das Suchen nach freien Speicherblöcken zu beschleunigen. Dazu wird ein Pointer auf den untersten freien Speicherblock gesetzt. Wird dieser besetzt, zeigt der Pointer

[27] Zugriffsmethoden für die Zusatzinformationen wurden noch nicht entwickelt. Es wird sich im weiteren Verlauf des Projektes zeigen, welche Flags für das Multitasking nützlich sind.

Bit	Belegung	Bedeutung
0	0	Der physikalische Speicherblock ist laut der **Boot-Struktur** nicht verfügbar.
0	1	Der physikalische Speicherblock ist laut der **Boot-Struktur** verfügbar.
1	0	Der physikalische Speicherblock ist ist für die Benutzer.
1	1	Der physikalische Speicherblock ist für den Kernel.
2	0	Der physikalische Speicherblock kann soll nicht *gemeinsam genutzt* werden.
2	1	Der physikalische Speicherblock kann *gemeinsam genutzt* werden.
3	1	Der physikalische Speicherblock ist frei.
3	0	Der physikalische Speicherblock ist belegt.

Tabelle 3.1: Bitbelegung der Tabelle zur physikalischen Speicherverwaltung

Bit 3	Bit 2	Bit 1	Bit 0	Bedeutung
0	0	0	0	nicht verfügbarer Speicher laut E820-Entries
0	0	0	1	belegter Benutzerspeicher
0	0	1	1	belegter Kernelspeicher
0	1	0	1	belegter teilbarer Benutzerspeicher
0	1	1	1	belegter teilbarer Kernelspeicher
1	0	0	1	freier Benutzerspeicher
1	0	1	1	freier Kernelspeicher
1	1	0	1	freier teilbarer Benutzerspeicher
1	1	1	1	freier teilbarer Kernelspeicher

Tabelle 3.2: sinnvolle Kombinationen der Bitbelegung

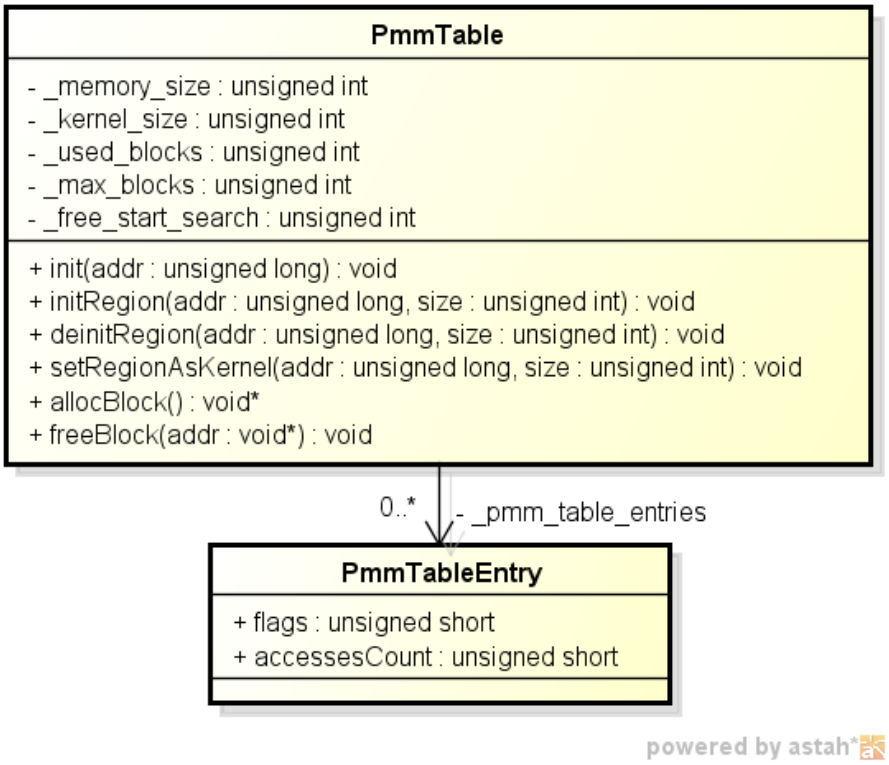


Abbildung 3.1: PmmTable

auf den nächsten Eintrag. Dadurch wird das fortlaufende Suchen beschleunigt. Wenn untere *physikalische Speicherblöcke* wieder frei werden, wird der Zeiger auf diese gesetzt. Dadurch sinkt die Performanz, da nachfolgende Anfragen die Tabelle von dem Pointer aus weiter durchsuchen müssen. Allerdings wird durch das Umsetzen des Pointers vermieden, dass Lücken im physikalischen Speicher entstehen, wodurch dann wieder in konstanter Zeit auf freien Speicher zugegriffen werden kann, wenn die Lücken gefüllt sind. Ist kein freier Speicher verfügbar, wird die als `Error` definierte Adresse „0x00000000“ zurückgegeben.

3.2 Aufteilung des Adressraums

Neben der Verwaltung des freien und belegten physikalischen Speichers, wurde überlegt, wie die Kommunikation zwischen Prozessen und Kernel stattfinden kann.

Dabei wurde auch berücksichtigt, dass ein Kontextwechsel[28] sehr aufwendig ist, also viel CPU Zeit benötigt, und wie ein unnötiger Wechsel nur für den Scheduler vermieden werden kann. Ein Kontextwechsel findet statt, wenn das CR3 Register

[28] Unterbrechung des aktuellen Prozesses inklusive Speicherung des Kontext und Wechsel zu einer anderen Routine

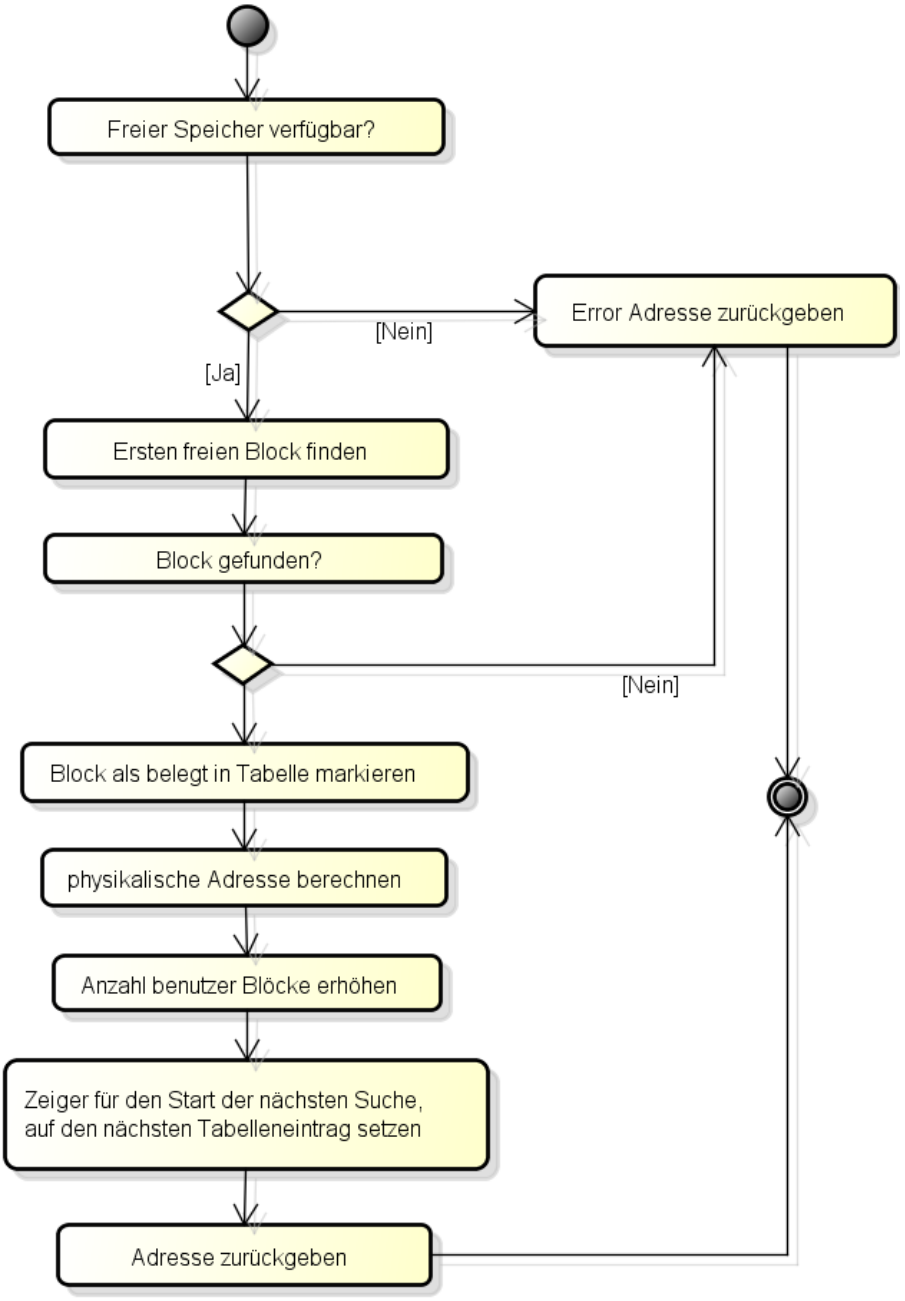


Abbildung 3.2: Speicher allokieren

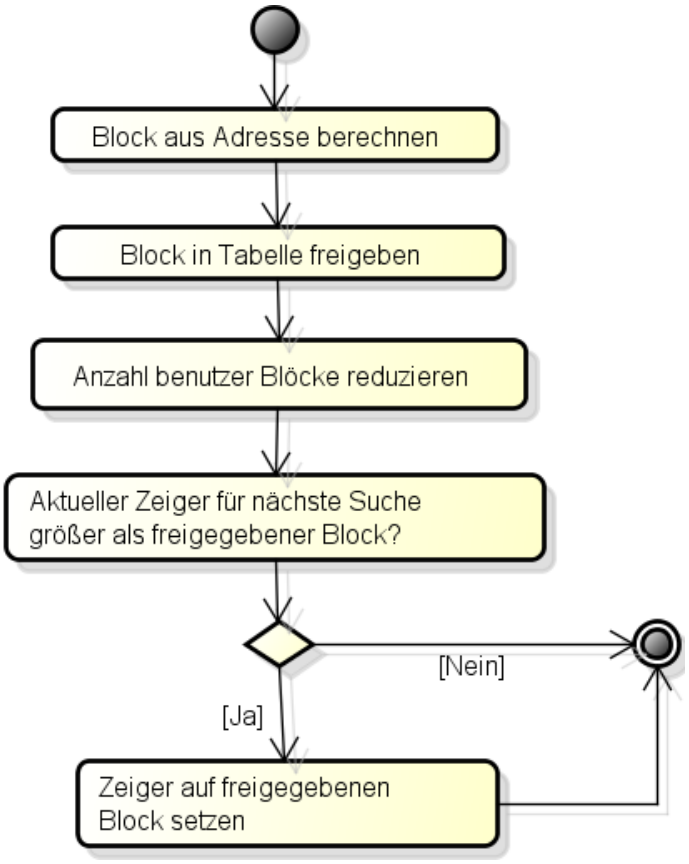


Abbildung 3.3: Speicher freigeben

(das Register, das den Page Directory Eintrag enthält) mit einer neuen Adresse belegt wird.

Üblicherweise wird der 4GB große virtuelle Adressraum, der mit 32-Bit adressiert werden kann, in zwei Teile aufgeteilt: Kernel- und User-Space.
Am häufigsten ist die Aufteilung mit 1GB für den Kernel-Space und 3GB für den User-Space. Dabei ergibt sich, dass Prozesse nur maximal 3G Speicher exklusiv für sich adressieren können.[29]

Für die Aufteilung diese Bereiche haben sich die folgenden zwei Vorgehensweisen etabliert:

3.2.1 Lower Half Kernel

$0 - 1$ GB sind für den Kernel vorgesehen und $1 - 4$ GB folglich für den Userspace. Durch die Zuweisung des Adressraums $0x00000000 - 0x3FFFFFFF$ für den Kernel kann dieser eins-zu-eins abgebildet werden. Dies ermöglicht eine einfachere Implementierung, kann allerdings auch zu Problemen führen. So müssen

[29] In der Regel weniger, denn auch bei 4GB physikalischen Speicher müssen diverse Geräte auf diesen abgebildet werden. Dieser Speicher kann also nicht angesprochen werden obwohl er physikalisch vorhanden ist.

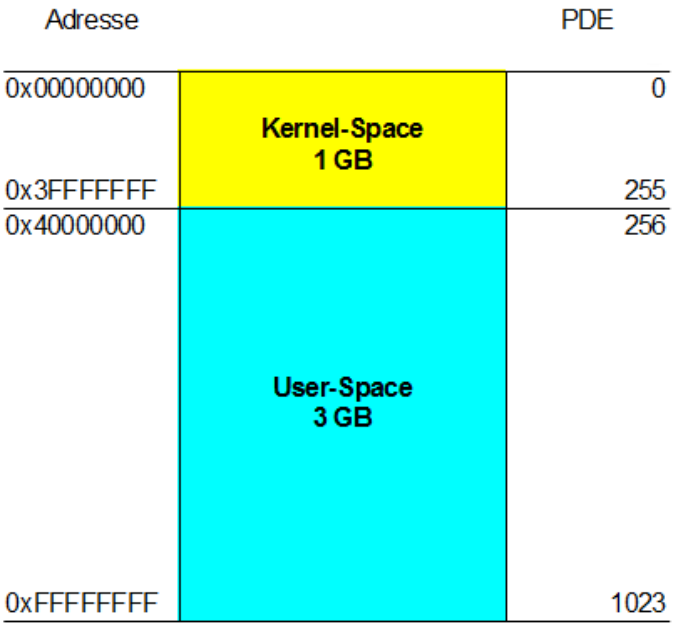


Abbildung 3.4: Lower Half Kernel

Anwendungsprogramme so kompilliert werden, dass sie erst bei einer 1 GB Startadresse beginnen.

3.2.2 Higher Half Kernel

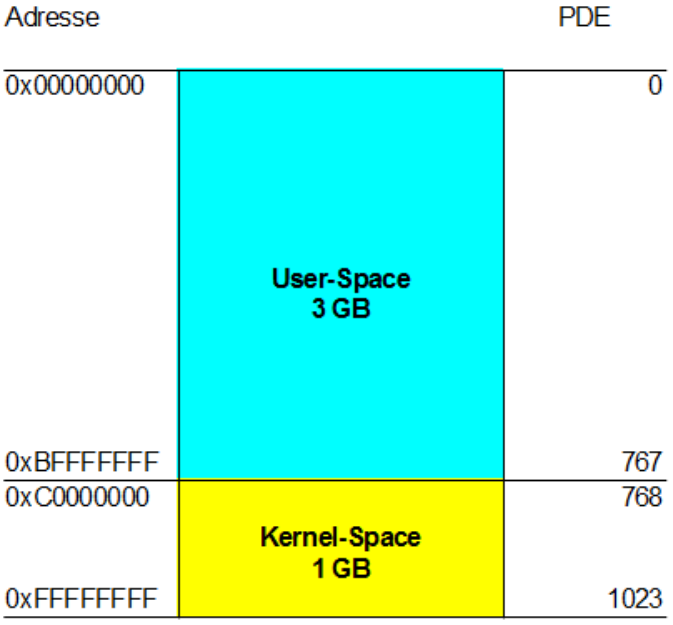


Abbildung 3.5: Higher Half Kernel

Der Higher Half Kernel zeichnet sich beim 32-Bit System dadurch aus, dass der Kernel Space nun im Bereich $0xC0000000 - 0xFFFFFFFF$ liegt. Bis zu Initialisierung des Pagings muss der Kernel in diesem Fall zunächst mit physiaklischen und

später mit virtuellen Adressen arbeiten. Diese zunächst als unnötig aufwendiger erscheinende Vorgehensweise bringt allerdings diverse Vorteile mit sich.

Bei einem 64-Bit System mit einem Higher Half Kernel (Kernel Space also oberhalb von $0xFFFFFFFF$) ermöglicht diese Aufteilung weiterhin, dass 32-Bit Anwendungen den gesamten 32-Bit Adressraum nutzen können, was bei einem Lower Half Kernel nicht möglich wäre.

3.2.3 Umsetzung

Bei der Umsetzung wird nur der Higher Half Kernel betrachtet.

Jeder Prozess erhält ein eigenes Page Directory im physikalischen Adressraum. Die letzten PDE Einträge (768 bis 1023) im Page Directory verweisen dabei auf den Kernel-Space und sind bei allen Prozessen gleich. Diese Einträge sollten dann durch die im Prozessor vorhanden Schutzmechanismen geschützt werden. Die restlichen PDEs adressieren den User-Space Bereich.

So können dieselben PDE Einträge für jeden Prozess für die Adressierung des Kernel-Spaces verwendet werden. Wenn man so vorgeht, benötigt der Kernel kein eigenes Page Directory, er wird stattdessen als Teil jedes Prozesses angesehen und nicht als ein eigenständiger Prozess. Außerdem kann man die PDEs mehrfach verwenden, spart also Speicher ein. Ohne die Aufteilung in Kernel- und User-Space würde man pro Prozess die gleichen PDEs erstellen müssen.

Bei einem Prozesswechsel muss das CR3 nur auf das Page Directory für den neuen Prozess gesetzt werden. Für den Scheduler muss also kein extra Kontextwechsel stattfinden. Zusätzlich ergibt sich, dass bestimmte Funktionen und Einsprungadressen für Kernelfunktionen, wie z. B. das Anfordern von Speicher oder die Auswahl des nächsten Prozesses, an der gleichen linearen Adresse für jeden Prozess bereit liegen.

Wenn man so vorgeht muss der Kernel nach Initialisierung also mindestens einen Prozess starten der niemals beendet werden darf. (Unter Linux ist es z. B. der Init Prozess). Für diesen Prozess werden die PDEs und dazugehörigen PTEs für den Kernel-Space aufgebaut. Beim Starten weiterer Prozesse wird dann in deren Prozess-eigenen Page Directories ebenfalls auf die Einträge für den Kernel-Space verwiesen. Man kopiert also die PDE Einträge (max. 256). [30]

[30] Es macht wenig Sinn sofort 256 PTEs zu erstellen. Dafür ist erstens 1MB physikalischen Speichers notwendig. Zusätzlich wird es in den seltensten Fällen dazu kommen, dass der gesamte Kernel-Space adressiert werden muss. Daher wird dieser Bereich dynamisch auf- und abgebaut.

Die Abbildung 3.6 verdeutlicht, wie der Virtuelle Adressraum (User- und Kernel-Space) von zwei Prozessen im physikalischen Speicher vorliegen kann:

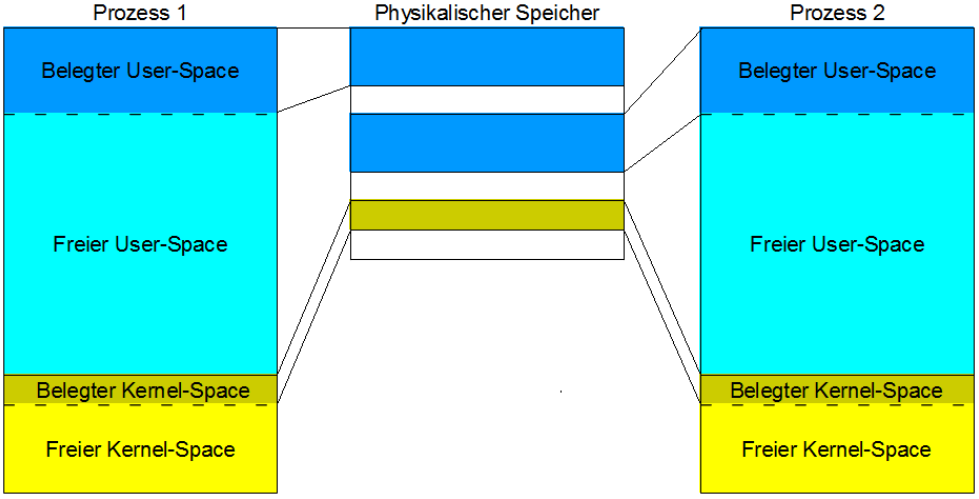


Abbildung 3.6: User- und Kernel-Space im physikalischen Speicher

Nach der Aufteilung in Kernel- und User-Space können Prozesse nun nur noch Speicher im virtuellen Adressraum exklusiv für sich reservieren, der unterhalb 3GB liegt – also ab Adresse 0xC0000000h. Das Anfordern von Speicher oberhalb dieser Adresse muss zu einem Fehler führen.

3.3 Paging

Das Paging setzt die linearen Adressen auf physikalische Adressen um. Somit ist der Zugriff auf den Speicher nur noch indirekt über das Paging beziehungsweise über lineare Adressen möglich. Das bedeutet auch, dass zur Verwaltung der Paging Directory und der Paging Tables eine vorhandene Abbildung auf die beiden Strukturen existieren muss. da keine direkte Verwaltung über physische Adressen mehr möglich ist. Da aber die Einträge in den beiden Pagingstrukturen physikalische Adressen sind und somit unterschiedlich zu den linearen Adressen sein können über die die Pagingstrukturen verwaltet werden können, ist es nötig dieses Problem bei der Implementierung der Pagingverwaltung zu beachten. Zur Lösung dieses Problem wurden zwei Herangehensweisen betrachtet die im folgenden kurz erläutert werden:

1. Bei der ersten Herangehensweise wird ein Teil des physikalischen Adressraums 1:1 auf den linearen Adressraum bzw. mit einem bekannten Offset abgebildet.

2. Die zweite Herangehensweise ist das Page Directory rekursiv abzubilden. Das bedeutet, dass ein Eintrag im Page Directory auf die physikalische

Adresse des Page Directory verweist. Somit kann dann über einen bestimmten Bereich im linearen Adressraum auf die Paging Struktur zugegriffen werden.

Von den beiden zur Auswahl stehenden Herangehensweisen wurde die zweite ausgewählt. Zwar ist der erste Algorithmus einfacher zu verstehen und zu implementieren. Die Herangehensweise hat aber den Nachteil, dass schon im Vorfeld sehr viel Speicher für die Pagingstrukturen reserviert werden muss, obwohl der Speicher noch nicht benötigt wird. Ein weiterer Punkt gegen die erste Herangehensweise ist die Anforderung von mehreren unterschiedlichen Pagingstrukturen für unterschiedliche Prozesse. Denn bei falscher Dimensionierung des 1:1 abgebildeten Bereiches kann es passieren, dass in diesem Bereich schnell kein Speicher mehr zur Verfügung steht oder falls immer nur die aktuelle Pagingstruktur in diesem Bereich vorgehalten werden soll die neue Pagingstruktur in den 1:1 abgebildeten Bereich kopiert werden muss.

3.3.1 Rekursives Paging Directory

Beim rekursiven Paging Directory wird ein Eintrag auf die physikalische Anfangsadresse des Paging Directories gesetzt. In der Implementierung des FHDW-OS wurde der letzte der Eintrag des Paging Directories dazu verwendet. Dies hat zur Folge, dass wenn auf eine lineare Adresse im Bereich von $0xFFC00000$ bis einschließlich $0xFFFFFFFF$ zugegriffen wird, der letzte Eintrag im Paging Directory genutzt wird. Da dieser wieder auf das Page Directory verweist wird nun das Page Directory wieder benutzt, aber diesmal in der Rolle einer Page Table. Die Page Table übernimmt dann die Rolle der Page im Speicher auf die zugegriffen wird. In Abbildung 3.7 ist beispielhaft der Ablauf für den Zugriff auf die Adresse $0xFFC001002$ dargestellt, mit der man den Zugriff auf den dritten Eintrag in der zweiten Paging Table erhält.

Somit ist es möglich, im linearen Adressraum von $0xFFC00000$ bis $0xFFFFFFFF$ auf die Pagingstrukturen zuzugreifen. In Abbildung 3.8 ist exemplarisch dargestellt, mithilfe welcher linearen Adressen, durch die Benutzung des rekursiven Paging Directory Ansatzes, ein Zugriff auf die Pagingstrukturen möglich ist.

3.3.2 Aufbau

Die zur Verwaltung der Pagingstruktur erstellte Klassen-Struktur ist in zwei Bereiche eingeteilt.

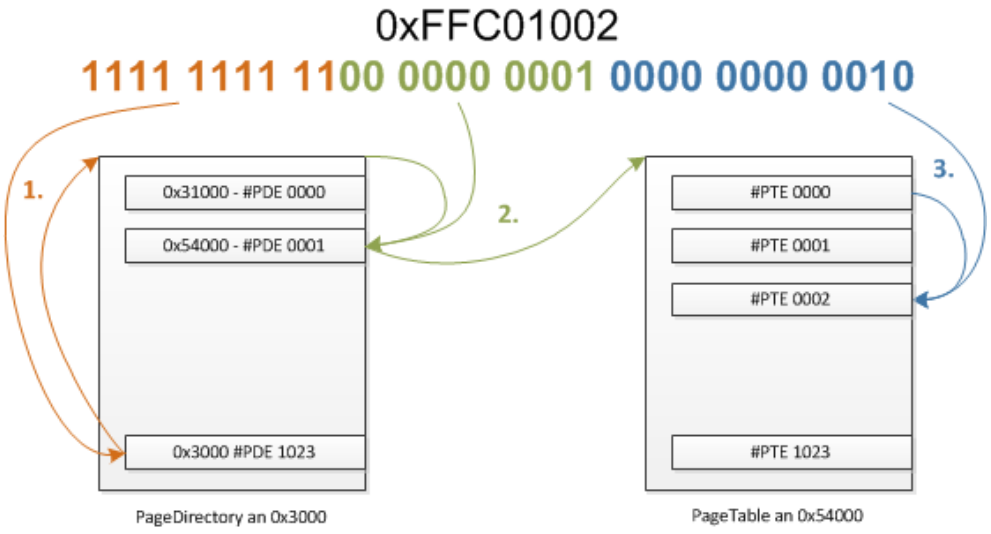


Abbildung 3.7: Rekursiver Verweis im Paging Directory

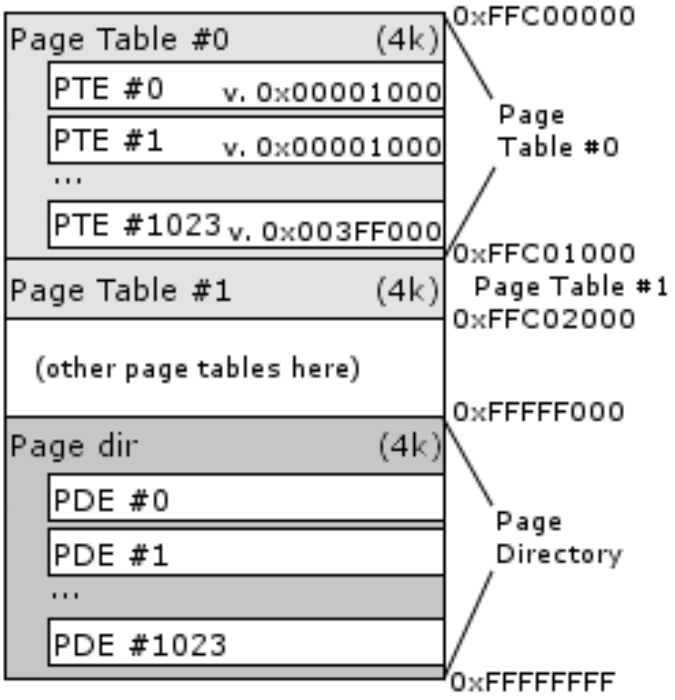


Abbildung 3.8: Lineare Adressen zum Zugriff auf die Pagingstrukturen

- Der erste Bereich enthält Funktionen und Schnittstellen, um von außerhalb verwaltende Aufgaben auszuführen. Dazu zählen hauptsächlich die Funktionen zum Abbilden oder Freigeben von Seiten und der Wechsel des aktuellen Paging Kontextes. Diese Aufgaben werden durch die Klasse *PageContext* bereitgestellt. Weiterhin repräsentiert ein Objekt dieser Klasse einen spezifischen Paging Kontext und stellt die statische Methode *Init* zur Verfügung, die die einmalige Initialisierung des Pagings durchführt. In Abbildung 3.9 ist die Klasse *PageContext* dargestellt.

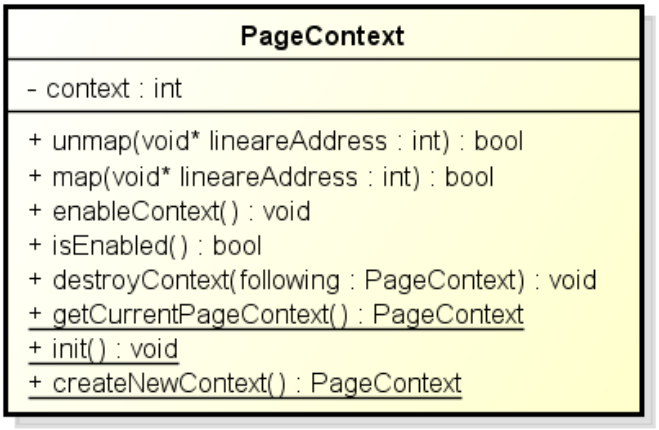


Abbildung 3.9: Klasse PageContext

- Im zweiten Bereich sind mehrere Klassen die zur internen Verwaltung der Pagingstrukturen benutzt werden. Ein Zugriff auf diese Klassen ist von außerhalb nicht vorgesehen. Eine Übersicht über die Klassen dieses Bereiche ist in Abbildung 3.10 abgebildet.

  PageEntry ist entweder ein Eintrag in der PageTable oder im Page Directory. Die Klasse bietet Operationen zum Ändern der Flags oder zum Setzen der physikalischen Adresse an. Die Klassen **PageDirectoryEntry** und **PageTableEntry**, die von der Klasse PageEntry ableiten, enthalten spezielle Implementierungen für den Ablauf.

  PageTable repräsentiert eine PageTable und bietet Operationen um einen Eintrag für einen bestimmte lineare Adresse auszuwählen.

  PageDirectory spiegelt ein PageDirectory wieder. Die Klasse bietet Operationen wie die Klasse PageTable an um einen Eintrag auszuwählen. Desweiteren gibt es Operationen um Abbildungen zu erstellen oder aufzuheben, zum Setzen des rekursiven Eintrages und Operationen die zum Erstellen eines neuen PagingDirectories benötigt werden.

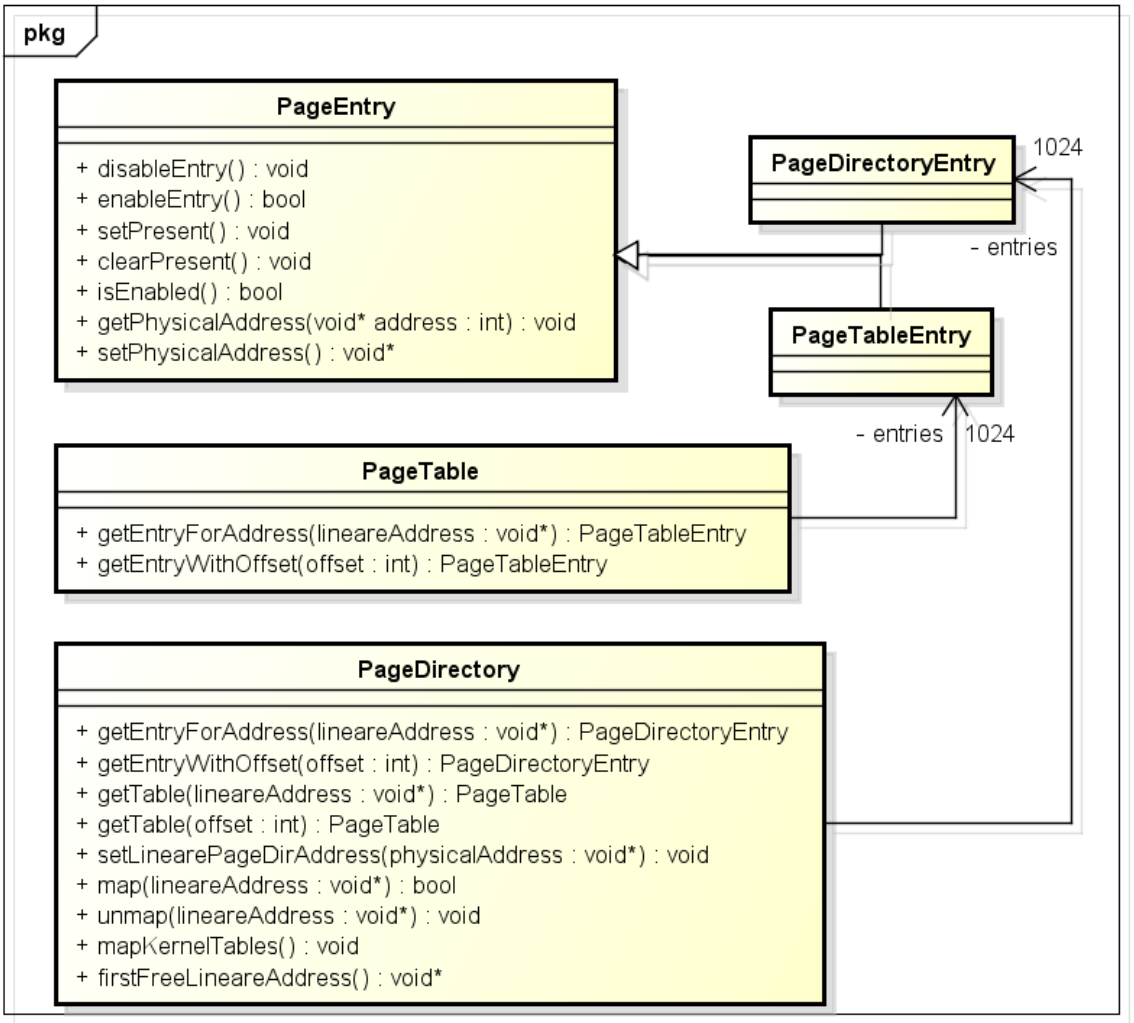


Abbildung 3.10: Klassenstruktur des internen Bereiches

3.3.3 Initialisierung des Pagings

Die Initialisierung des Paging findet vor der Initialisierung der physikalischen Speicherverwaltung statt. Dabei wird zunächst der Eintrag für das rekursive Paging gesetzt. Darauf werden die Page Tables für den Kernel Space erstellt auf die dann von jedem Page Directory verwiesen wird. Außerdem werden dabei gleichzeitig die minimalen Abbildungen für den Kernel Space angelegt. Dazu zählen z. B. die Abbildungen für den Kernel, für die physikalische Speicherverwaltung als auch den Bereich für den Videospeicher. In diesem Zustand ist das Paging funktionsfähig, aber es existieren noch Abbildungen die durch den Bootloader erzeugt wurden. Diese werden nach der grundlegende Initialisierung des Kernels entfernt.

3.3.4 Ablauf des Abbildens neuer Pages

Da das Abbilden neuer Pages die Hauptfunktion der Pagingverwaltung ist, ist in Abbildung 3.11 dargestellt wie das Abbilden einer bestimmten linearen Adresse abläuft.

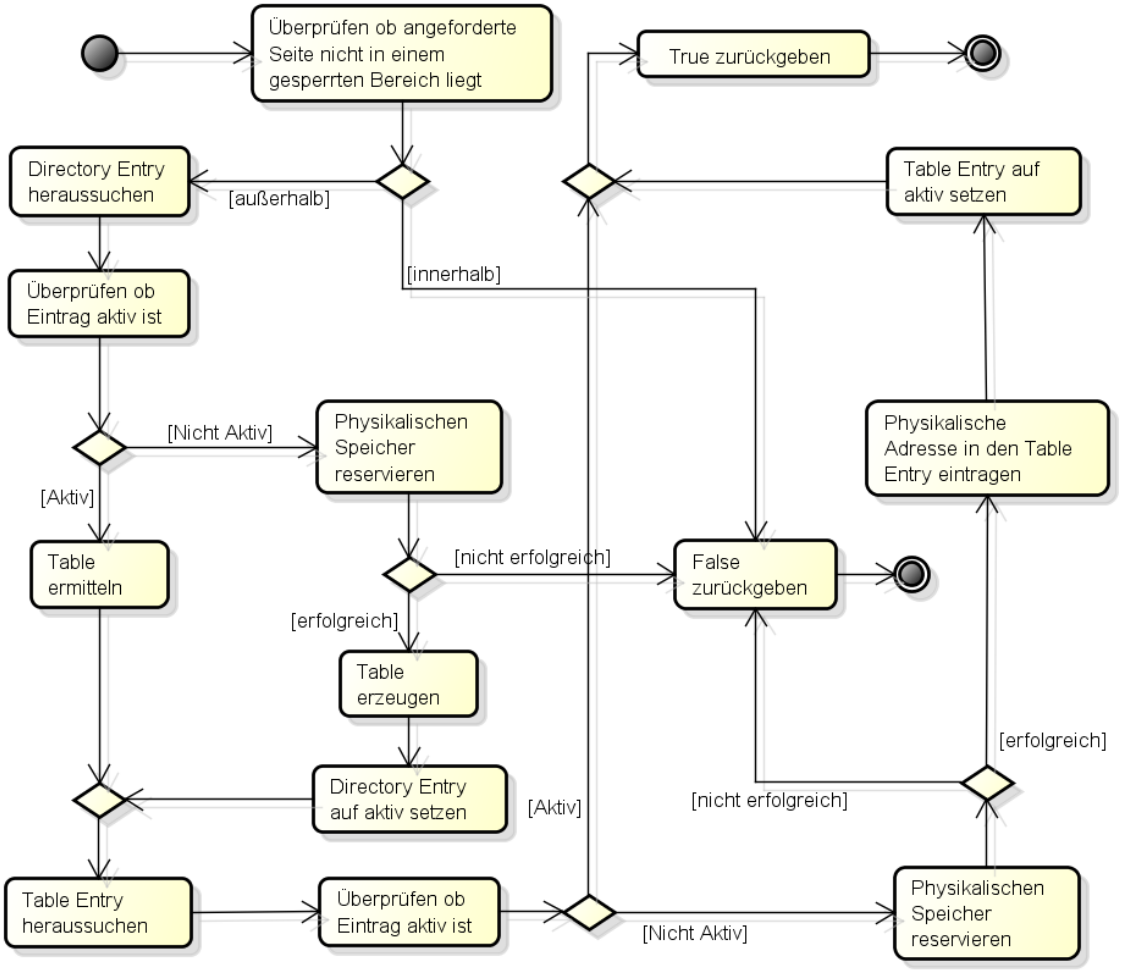


Abbildung 3.11: Ablauf des Abbildens

4 Schutzkonzepte der IA-32-Architektur

4.1 Allgemein

Intel Prozessoren mit ia-32 Architektur verfügen über ein umfassendes Sicherheitskonzept für die Speicherverwaltung, das dem Programmierer viele Einsatzmöglichkeiten bietet, aber auch Herausforderungen mit sich bringt.

Ein wesentlicher Punkt sind hier sicherlich die unterschiedlichen *Privilegstufen* sowie Limitierungen, die mit ihnen verbunden sind. Wie diese Limitierungen ausgeprägt sind und welche Möglichkeiten es dennoch gibt, wird im Folgenden ebenso erläutert wie die Verwendung von *Limit*- und *Typ-Prüfungen*.
In diesem Kapitel werden die wesentlichen Mechanismen erläutert, die zum Umgang mit Intels ia-32 Architektur notwendig sind. Allerdings sind die Erläuterungen abstrakt gehalten und der Umgang mit den Schutzmechanismen wird nicht im Detail beschrieben. Als Referenz ist dafür auf die offizielle Intel-Dokumentation [**?**] zurückzugreifen.

Ebenfalls ausgenommen sind Erweiterungen für 64 Bit Unterstützung sowie Neuerungen in der ia-32e Architektur. Des Weiteren werden die Techniken zu *conforming - nonconforming* und *pointer validation* nicht berücksichtigt. Außerdem wird nicht auf alle Sprungbefehle (wie SysEnter und SysExit) eingegangen.

4.2 Privilegstufen

In der ia-32 Architektur wird zwischen 4 *Privilegstufen* unterschieden, die von 0 bis 3 angegeben werden. Stufe 0 stellt die höchste und Stufe 3 die geringste Berechtigung dar. Eine niedrigere Nummer entspricht daher einer höheren Privilegstufe.

Dieses Konzept lässt sich anhand von Abbildung 4.1 grafisch darstellen. Dort sind die Stufen als Ringmodell (links) sowie zusätzlich einige Zugriffe als Pfeile

(rechts) dargestellt. Diese Schutzringe beziehen sich auf Bereiche (im Folgenden nur noch Segmente) im Hauptspeicher, in denen Code- und Datensegmente sowie der Stack abgelegt werden. Blaue Pfeile stehen hier für den Zugriff auf Code-Segmente und schwarze für den Zugriff auf Datensegmente. Verbotene Zugriffe enden direkt auf den Linien der Ringen, während erlaubte Zugriffe diese Grenzen überschreiten. Wie hier zu sehen, hängen die Berechtigungen auch von dem Typ des Segments ab, auf das zugegriffen werden soll.

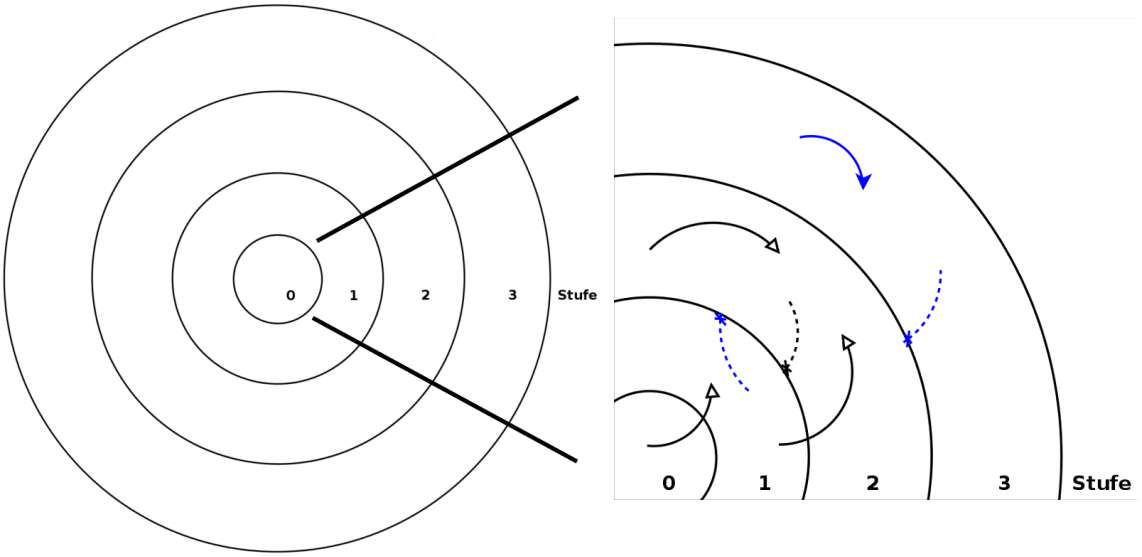


Abbildung 4.1: Ringmodell der Privilegstufen

In dem innersten Ring werden die schützenswertesten Segmente abgelegt. Diese gehören in der Regel direkt zum Betriebssystemkern. Nach außen verlieren die Segmente für den störungsarmen Betrieb immer mehr an Bedeutung. So können in Ring 1 beispielsweise Treiber, in Ring 2 Systemdienste und schließlich in Ring 3 Benutzerdaten oder Anwendungsprogramme liegen.

Die Privilegstufen sind strikt voneinander getrennt, sodass von einem äußeren Ring nicht ohne weiteres auf Segmente eines weiter innen liegenden zugegriffen werden kann, andersherum allerdings sehr wohl. So ist es beispielsweise einem Code-Segment in Privilegstufe 0 immer möglich ein Datensegment in Privilegstufe 1, 2, oder 3 zu verwenden. Ein Code-Segment in Privilegstufe 1 kann allerdings nicht auf Segmente in Privilegstufe 0 zugreifen.

4.2.1 Kommunikation zwischen Privilegstufe

Da eine Kommunikation zwischen den unterschiedlichen Privilegstufen erforderlich ist, gibt es Mechanismen, die unter bestimmten Bedingungen auch das Verwenden von Segmenten aus höheren Privilegstufen ermöglichen.

Hierfür ist es notwendig, zur Programmlaufzeit die Privilegstufen-Zugehörigkeit des gerade verarbeiteten Code-Segments zu kennen. Diese Information wird als **Current Privilege Level (CPL)** bezeichnet und entspricht immer der Nummer der Privilegstufe in dem sich das Code-Segment befindet. Die CPL wird dabei aus dem CS Register gelesen. Soll ein Segment s_dest von einem Code-Segment s_source geladen werden, muss ein entsprechender Segmentselektor für dieses zu ladende Segment verwendet werden. Dieser wiederum enthält ebenfalls eine Privilegstufe, das **Requested Privilege Level (RPL)**. CPL und RPL werden für einen Segmentaufruf zum **Effective Privilege Level (EPL)** zusammengefasst. Dieses Zusammenfassen kann man sich folgendermaßen vorstellen:

$$EPL = maxNumber(CPL, RPL)$$

Beim Laden von s_dest wird dieses EPL des gerade aktiven Prozesses mit dem **Descriptor Privilege Level (DPL)** des Segmentdeskriptors von s_dest vergleichen, bevor auf dieses zugegriffen wird.

Je nachdem ob ein Code- oder Datensegment oder der Stack geladen werden soll, gelten unterschiedliche Bedingungen. Tabelle 4.1 stellt die wesentlichen Aussagen dar. Es wird immer von numerischen Werten ausgegangen.

Zugriffsziel	Prüfung
Datensegment	$EPL \leq DPL$
Code-Segment	$(CPL \geq RPL) \wedge (CPL = DPL)$
Stack	$CPL = RPL = DPL$
Unterbrechung	$CPL \leq DPL$
Call Gate	siehe Kapitel Call Gates

Tabelle 4.1: Privilegstufen Prüfung

Eine solche Prüfung findet vor jedem Laden eines Segmentes statt und benötigt immer die CPL, RPL und DPL Informationen.

Abbildung 4.2 zeigt einige Beispiele für das Zugreifen auf ein Datensegment aus verschiedenen Privilegstufen. Hier ist unter anderem noch einmal dargestellt, dass auch ein Code-Segment aus Privilegstufe 0 nicht auf ein Datensegment in Privilegstufe 2 zugreifen kann, wenn es einen Segmentselektor aus Privilegstufe 3 verwendet, da $maxNumber(CPL_{code}, RPL_{selector}) > DPL_{data}$.

Dabei ist allerdings zu berücksichtigen, dass das RPL eines Segmentselektors durch Software veränderbar ist. In Abbildung 4.2 könnte beispielsweise Code-Segment D das RPL von Segmentselektor 3 auf 0 setzen und so auf Datensegment 1 zugreifen. Code-Segment C würde davon allerdings nicht profitieren, da seine CPL nach wie vor nicht ausreicht.

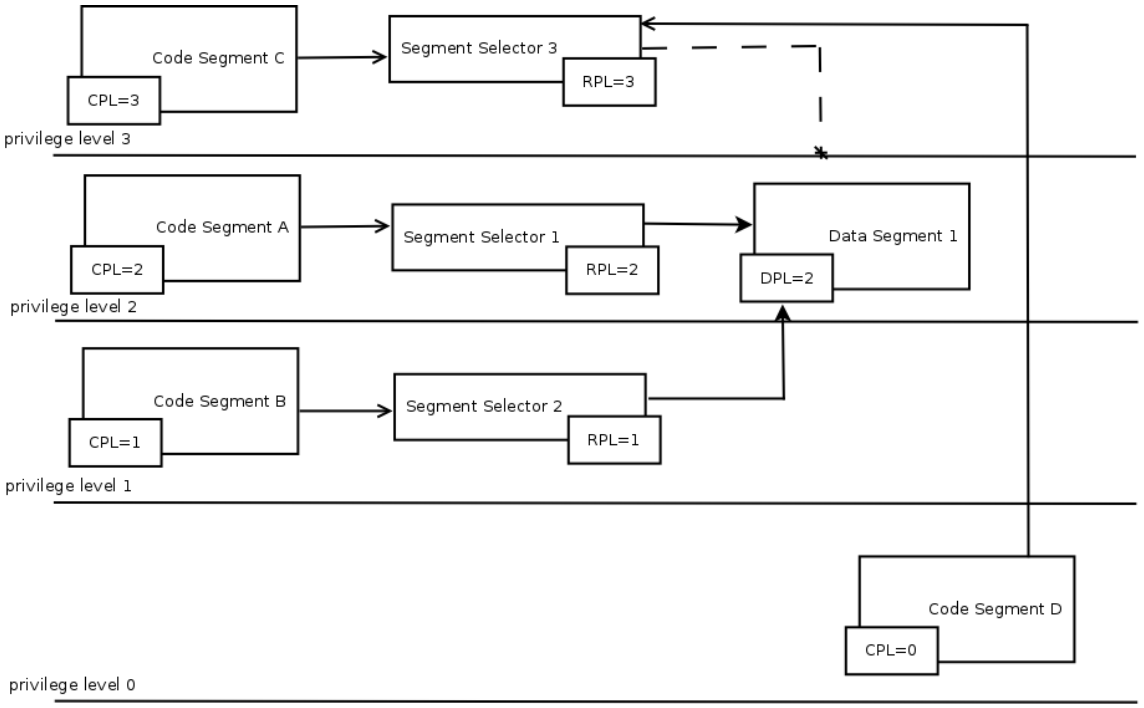


Abbildung 4.2: Segment Zugriff

Hinweis:
Wird versucht ein Segment mit einer nicht ausreichenden Privilegstufe zu laden, wird vom Prozessor eine “general-protection exception“ erzeugt.

Während beim Laden von Datensegmenten ein Zugriff aus nummerisch kleineren Privilegstufen immer möglich ist, gilt dies allerdings nicht beim Zugriff auf Code-Segmente. Schaut man sich in Tabelle 4.1 die Bedingung für Code-Segmente an, fällt unmittelbar auf, dass die Bedingung $CPL = DPL$ einen solchen Zugriff verbietet.

Hinweis:
Der Zugriff auf ein Code-Segment wird maßgeblich davon beeinflusst, ob das conforming oder nonconforming Verfahren verwendet wird. Da in diesem Projekt nicht beabsichtigt wird, conforming einzusetzen, wird auf dieses Verfahren nicht weiter eingegangen. Alle Beschreibungen beziehen sich daher auf das nonconforming Verfahren.

Wird eine Unterbrechung durch Software ausgelöst und ein Unterbrechungsbehandler über die Befehle INT n, INT 3 oder INTO geladen, muss ebenfalls eine Privilegstufen-Prüfung wie in Tabelle 4.1 beschrieben durchgeführt werden. Würde an dieser Stelle keine Prüfung erfolgen, könnten diese Unterbrechungen benutzt werden um Code-Segmente einer nummerisch kleineren Privilegstufe aufzurufen. Wird eine Unterbrechung allerdings durch Hardware ausgelöst, wird keine Privilegstufen-Prüfung durchgeführt.

4.2.2 Stack Wechsel

Beim Aufruf eines Code-Segments einer anderen Privilegstufe wird nach der Legitimierung ein Wechsel des Stacks notwendig. Die Informationen über den Ablageort des jeweiligen Stacks werden für die Privilegstufen 0-2 im TSS[31] abgelegt. Hier gibt es für jede Privilegstufe einen eigenen Bereich. Für Stufe 3 sind diese Informationen nicht im TSS abgelegt, da es keinen Wechsel in dieses Stufe geben kann[32].

Jede Privilegstufe benötigt einen eigenen Stack um sicherzustellen, dass beispielsweise Anwendungsprogramme keinen auf den Stack des Betriebssystems haben. Noch viel wichtiger als die Sicherheit der Daten ist allerdings das Wissen um den „Füllstand“ des eigenen Stacks. Folgendes Szenario soll dies unter einigen fiktiven Annahmen verdeutlichen.

Nehmen wir einmal an unser Stack hätte eine Adresslänge von 1 Byte (8 Bit) und somit 256 verfügbare Adressen. Ein Anwendungsprogramm könnte nun beliebig viel Speicher dieses Stacks in Anspruch nehmen, z. B. 200 Adressen. Wenn nun eine Betriebssystemroutine ebenfalls Daten auf den Stack schieben möchte, bleiben dieser nur noch 56 Adressen. Beim Schreiben des 57. Eintrages würde entweder eine Ausnahme erzeugt oder Daten des Anwendungsprogramms müssten überschrieben werden. Im schlimmsten Fall würde das gesamte System aufgrund eines Anwendungsprogrammes abstürzen.

Um ein solches Szenario zu vermeiden, müssten ständig die Stacks abgefragt und ggf. intelligente Mechanismen zum Gegensteuern erdacht werden. Allein um dies zu vermeiden ist es sinnvoll für jede Privilegstufe einen eigenen Stack bereitzustellen.

4.2.3 Call Gates

Call Gates stellen eine spezielle Methode des Segmentzugriffs dar. Über Call Gates ist es möglich, auf Code-Segmente einer höheren Privilegstufe zuzugreifen. Dabei ist der Legitimierungsprozess bei Verwendung von Call Gates in etwa derselbe wie bei herkömmlichen Segmentzugriffen. Zunächst wird über einen Call Gate Selektor das Call Gate angesprochen und die Privilegstufen-Prüfung durchgeführt. Hierzu wird zum einen die Prüfung $EPL \leq DPL_{GateDescr.}$ und eine weitere, anhängig des verwendeten Zugriffs, durchgeführt. Diese hängt davon

[31] task-state segment: Auf eine weitere Erklärung wird an dieser Stelle verzichtet
[32] Ein Aufruf eines Code-Segments einer nummerisch größeren Privilegstufe ist nicht gestattet

ab, ob der *CALL*[33] oder der *JMP*[34] Befehl verwendet wird. Bei Verwendung von CALL wird zusätzlich $DPL_{CodeDescr.} \leq CPL$ und bei JMP $DPL_{CodeDescr.} = CPL$ geprüft. Nach erfolgreicher Prüfung wird das entsprechende Code-Segmentgeladen. Bei der Privilegstufen-Prüfung für Call Gates werden daher 4 anstatt wie üblich 3 Informationen benötigt: das CPL, das RPL des Call Gate Selektors, sowie die DPL des Call Gates Deskriptor und des aufzurufenden Code-Segmentdeskriptors.

Dabei ist zu berücksichtigen, dass das zu ladende Code-Segment nicht allein vom verwendeten Call Gate adressiert wird. Für das Lokalisieren des auszuführenden Code-Segments wird über den Call Gate Selektor der Segmentdeskriptor des Code-Segments ausfindig gemacht. Die dort eingetragene Adresse wird mit einem Offset aus dem Call Gate kombiniert und das Segment an der zutreffenden Adresse angesprungen, falls die Privilegstufen-Prüfung erfolgreich war.

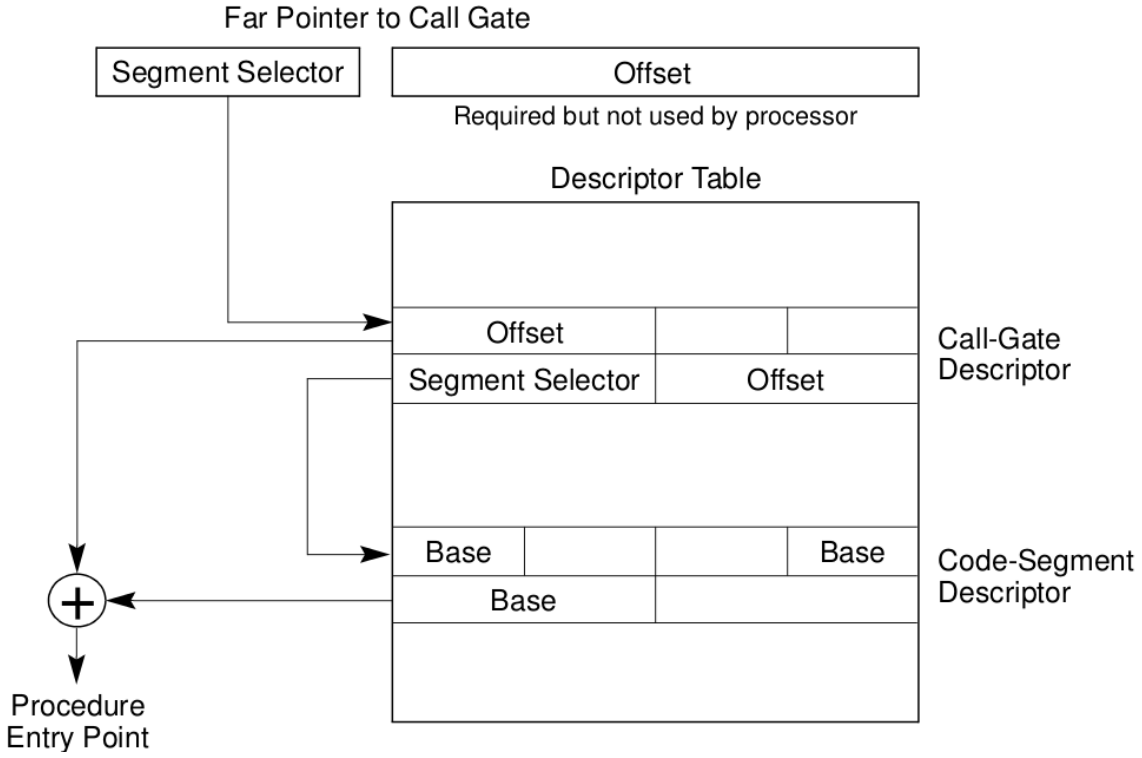


Abbildung 4.3: Offset bei Verwendung von Call Gates

In Abbildung 4.3 wird noch einmal die Verwendung des Offsets dargestellt.

4.2.4 Zurückkehren nach einem Prozeduraufruf

Auch beim Zurückspringen nach einem CALL Befehl muss ggf. eine Privilegstufen-Prüfung durchgeführt werden. Dies ist immer dann erforderlich, wenn durch das

[33] Aufruf einer Routine mit anschließendem Rücksprung nach Abarbeitung der Routine
[34] Sprung an eine angegebene Speicheradresse ohne Rücksprung

Zurückkehren das aktuelle Segment verlassen wird und notwendig, um eventuell manipulierten Rücksprungadressen entgegen zu wirken. Eine Privilegstufen-Prüfung wird hier genau wie auch beim Ausführen des CALL Befehls durchgeführt, allerdings genau andersherum.

4.2.5 Eingabe-Ausgabe Privilegstufen

Eine weitere Schutzfunktion ist das **Input-Output Privilege Level (IOPL)**. Das IOPL ist ein zwei-Bit Flag im EFLAGS-Register und gibt die Privilegstufe an, die ein Code-Segment mindestens haben muss, um auf die Hardware zugreifen zu können.

Typischerweise wird dieses Privileg nur dem System selbst gewährt, weshalb meist nur ein direkter Zugriff von Privilegstufe 0 aus möglich ist. Der Wert des IOPL Flags, kann nur von den Instruktionen POPF und IRET in Privilegstufe 0 geändert werden und kontrolliert alle direkten Hardwarezugriffe. Die Prüfung für einen solchen Zugriff entspricht dabei $CPL \leq IOPL$.

4.3 Priviligierte Instruktionen

Einige Systembefehle sind vor dem Aufruf durch Anwendungsprogrammen geschützt. Diese priviligierten Instruktionen dürfen ausschließlich mit der Privilegstufe 0 (höchste Privilegstufe) ausgeführt werden. Ein Aufruf durch eine niedrigere Privilegstufe löst eine Ausnahme aus. Folgende Befehle sind priviligierte Instruktionen:

LGDT Lade GDT-Register

LLDT Lade LDT-Register

LTR Lade Task-Register

LIDT Lade IDT-Register

MOV Kontrollregister laden/speichern

MOV Debug-Register laden/speichern

LMSW Maschinenstatuswort laden

INVD Cache invalidieren, ohne Zurückschreiben

WBINVD Cache invalidieren, mit Zurückschreiben

INVLPG TLB-Eintrag invalidieren

HLT Prozessor anhalten

4.4 Limit-Prüfungen

Der Prozessor nimmt Überprüfungen der Zugriffe auf Adressen eines Segments vor. Dies dient v. a. zum Auffangen von Programmierfehlern sowie fehlerhaften Zeigerberechnungen. Die dafür notwendigen Informationen sind im Segmentdeskriptor hinterlegt.

Ein Limit-Feld (20 Bit) gibt das Offset der letzten gültigen Adresse des Segments an. Die maximale Größe eines abgeschlossenen Segments beträgt 2^{20} (1 MB) bzw. 2^{32} (4 GB) bei gröberer Granularität. Die Granularität wird durch Setzen eines Bits (G) im Deskriptor entsprechend verringert. Dies bewirkt eine Multiplikation des Wertes des Limit-Feldes mit 2^{12} (4 KB). So wird die maximale Größe des Segments erhöht. Bei grober Granularität bedeutet z. B. ein Limit-Wert von 100, dass Zugriffe bis zu einem Offset von $100 * 4\mathrm{KB} = 400\mathrm{KB}$ erlaubt sind.

Dabei ist zu beachten, dass bei dieser Erweiterung jeder Zugriff auf Adressen im letzten Kilobyte (0–FFFh) gültig ist.

Außerdem unterliegen Deskriptortabellen ähnlichen Limit-Prüfungen. Diese verhindern den Zugriff auf Deskriptoren, die hinter der Tabelle liegen.

4.5 Typ-Prüfung

Der Prozessor unterscheidet zwischen System-, Code- und Datensegmenten. Der Typ eines Segments ist im Segmentdeskriptor charakterisiert. Bei Zugriffen auf Segmente erfolgt eine Prüfung des Typs je nach Anwendungsfall. Eine unvollständige Liste von Beispielszenarien folgt:

- Laden eines Selektors in ein Segmentregister. Einige Register dürfen nur bestimmte Segmenttypen beinhalten, z. B.:
  - CS-Register enthält nur Selektoren für Code-Segmente
  - SS-Register enthält nur beschreibbare Datensegmente
- Zugriff auf Segmente, wenn deren Deskriptoren bereits in einem Register geladen sind.
  - Kein Schreiben in ein ausführbares Segment
  - Kein Schreiben in ein schreibgeschütztes Segment

  - Kein Lesen eines ausführbaren Segments, wenn es als nicht lesbar markiert ist

- In internen Operationen, z. B.:
  - Bei einem CALL oder JMP wird durch das Typ-Feld des entsprechenden Deskriptoren indiziert entweder ein den Wechsel in ein Code-Segment oder einen Task-Wechsel.
  - Bei CALL oder JMP durch ein Call-Gate muss der dadurch erreichte Segmentdeskriptor ein Code-Segment auszeichnen.

4.6 Seitenschutz

Der Prozessor steuert auch die Zugriffe auf Seiten im Speicher. Dieser Schutzmechanismus ist zunächst unabhängig von den Schutzmechanismen für Segmente. Der Zugriffsschutz von Seiten ist in zwei Dimensionen charakterisiert. Diese Dimensionen, die Domäne und der Seitentyp, werden im Folgenden erläutert.

Die Auswirkungen der Kombination von Seitenschutz und Privilegstufen werden im letzten Unterabschnitt beschrieben.

4.6.1 Domäne und Typ einer Seite

Eine Seite wird einer Domäne zugeordnet. Dabei wird unterschieden zwischen dem *Supervisor-Modus* und dem *User-Modus*. Bei einem Segment-Level (CPL) von 0, 1 oder 2 wird automatisch der Supervisor-Modus verwendet. Lediglich Segment-Level 3 verwendet den User-Modus. Dieses Konzept ermöglicht eine klare Abgrenzung der Seiten von Benutzerprozessen zu Seiten der Systemprozesse. Zu beachten ist dabei, dass auch Systemprozesse Zugriff auf Seiten im User-Modus erhalten.

Beim Seitentyp wird unterschieden zwischen schreibgeschützten und beschreibbaren Seiten. Ist im CR0-Register das WP-Flag nicht gesetzt, so ignorieren alle Supervisor-Prozesse den Schreibschutz von Seiten. Um schreibgeschützte Seiten auch im Supervisor-Modus zu beachten, wird daher das CR0.WP-Flag gesetzt.

Die Informationen zu einer Seite sind an zwei Stellen zu finden: Page-Directory und Page-Table. Ist eine Seite an beiden Stellen eingetragen, so werden die Einstellungen folgendermaßen kombiniert:

- Sind ein oder beide Einträge schreibgeschützt (Read-Only), so ist die Kombination schreibgeschützt.
- Sind ein oder beide Einträge Supervisor-Modus, so ist die Kombination auch Supervisor-Modus.

4.6.2 Zusammenwirken der Schutzkonzepte

Der Prozessor überprüft zuerst die Schutzmechanismen für Segmente (Privilegstufen) und anschließend den vorliegenden Seitenschutz. Eine Schutzverletzung in einer der beiden Schichten löst eine Ausnahme aus und verhindert den Zugriff auf den Speicher. Wird bereits beim Zugriff auf das Segment eine Ausnahme ausgelöst, so entfällt die Prüfung der Seite. Dies ist in Abbildung 4.4 dargestellt.

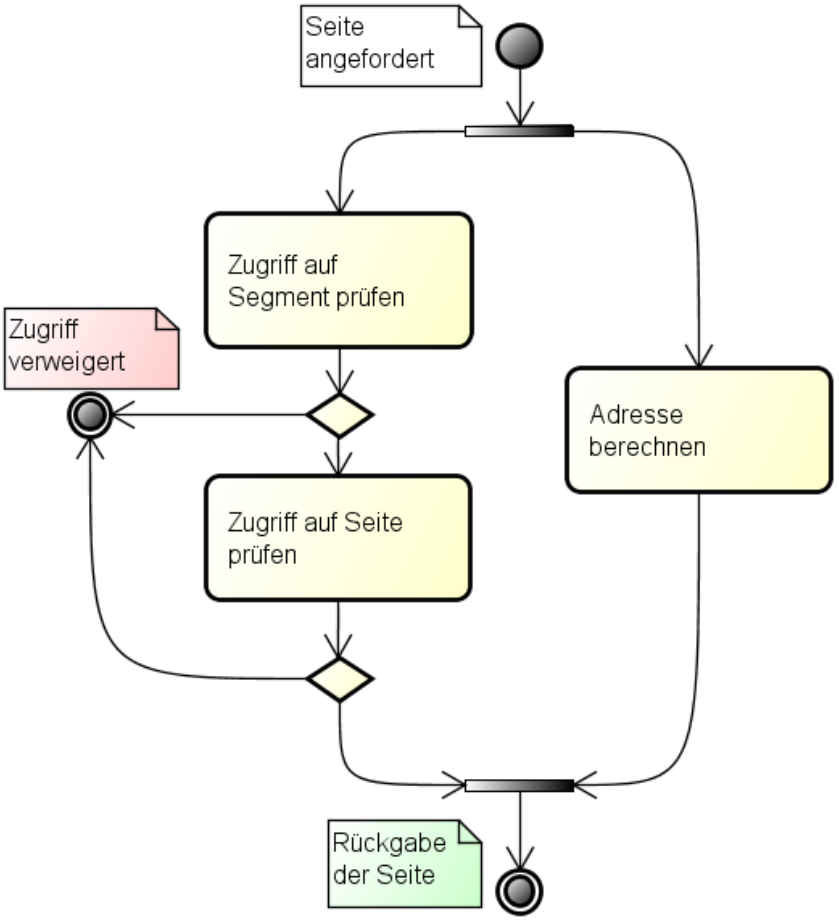


Abbildung 4.4: Kombination von Segment- und Seitenschutz

Dies hat zur Folge, dass die Privilegstufen der Segmente nicht durch Einstellungen auf Seitenebene aufgehoben oder ausgehebelt werden können. Ist z. B. ein Segment schreibgeschützt, so kann eine darin enthaltene Seite diesen Schreibschutz nicht umgehen, da schreibende Zugriffe bereits vor dem Zugriff auf die

Seite erkannt und geblockt werden. Dies trifft z. B. auf Code-Segmente zu, die immer schreibgeschützt sind.

5 VFS+FAT

5.1 Grundlagen

Jedes moderne Betriebssystem bietet die Möglichkeit zur Verwaltung von Dateien. Hierzu zählt das Schreiben und Lesen, und das Organisieren von Dateien in Verzeichnissen und Unterverzeichnissen. Diese Funktionalität fehlt im FHDW OS bisher vollständig.

Dieser Abschnitt soll zur Einführung in die Thematik dienen. Er beschreibt die Grundlagen eines virtuellen und eines ausgezeichneten konkreten Dateisystems.

5.1.1 Virtuelles Dateisystem

Ein virtuelles Dateisystem (*engl. virtual file system, VFS*) ist eine Abstraktion, welche für möglichst alle konkreten Dateisysteme verwendet werden kann. Dadurch muss ein Programm, das auf Dateien zugreift, nichts über die konkrete Implementierung des Dateisystems wissen. Der Zugriff auf verschiedene Dateisysteme wird vereinheitlicht. Das VFS stellt demnach eine API für Dateisystemoperationen zur Verfügung. Ein paar typische Operationen werden in Tabelle 5.1 aufgelistet.

Zusätzlich zu den einzelnen Operationen auf Dateien und Verzeichnissen muss ein VFS auch für die Organisation von Dateien und Verzeichnissen sorgen. Intuitiv wird hierfür ein Kompositum verwendet. Verzeichnisse bestehen aus Dateien und

Tabelle 5.1: Typische Dateisystemoperationen

Operation	Funktion
`mkdir, rmdir`	Erzeugen und Löschen eines Verzeichnisses
`opendir`, `closedir`	Öffnen und Schließen eines Verzeichnisses
`readdir`	Lesen von Verzeichniseinträgen
`chdir`	Wechseln in ein anderes Verzeichnis
`create, unlink`	Erzeugen und Löschen einer Datei
`open, close`	Öffnen und Schließen einer Datei
`read, write`	Lesen und Schreiben
`seek`	Neupositionieren des Schreib-/Lese-Zeigers

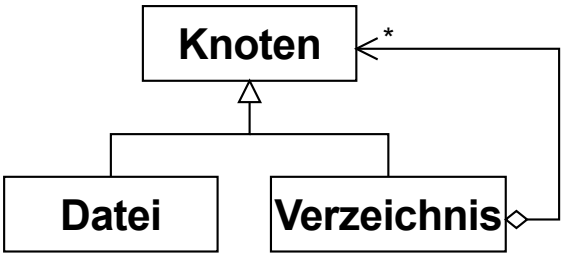


Abbildung 5.1: Dateisystem-Kompositum

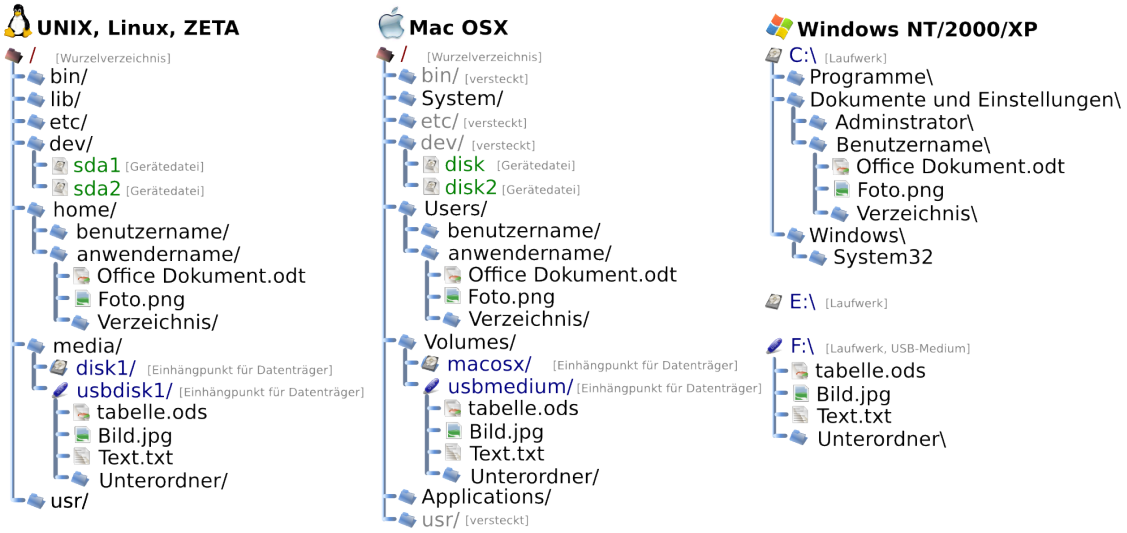


Abbildung 5.2: VFS in modernen Betriebssystemen

weiteren Verzeichnissen. Eine Datei hat keine weiteren Knoten. In Abbildung 5.1 ist ein Dateisystem-Kompositum zu sehen.

Diese Art des Kompositums wird in allen modernen Betriebssystemen verwendet. In Abbildung 5.2 werden verschiedene VFS-Implementationen gezeigt. Hierbei fällt auf, dass UNIX und Mac OSX ein einziges Kompositum für alle Dateisysteme verwendet, wohingegen Windows für jedes Dateisystem ein einziges Kompositum verwendet. Dies hat zwei Vorteile: Zum Einen ist es für den Benutzer einfacher, zwischen den verschiedenen Dateisystemen zu unterscheiden. Zum Anderen ist die Implementierung einfacher, da jedes Kompositum für sich betrachtet werden kann, ohne auf Abhängigkeiten zu achten.

Eine weitere Aufgabe des virtuellen Dateisystems ist das Ein- und Aushängen von Dateisystemen. Hierbei wird auf die konkreten Dateisystemtreiber zugegriffen. Das VFS muss über Schnittstellen den Zugriff auf alle eingehängten Dateisysteme ermöglichen.

5.1.2 FAT16

FAT16 ist ein von Microsoft entwickeltes Dateisystem. Es basiert auf Tabellen, in denen definiert ist welche Inhalte auf dem Datenträger gespeichert sind. Eine solche Tabelle heißt File Allocation Table (FAT). Die 16 steht dabei für die Anzahl an Bit mit denen die einzelnen Teile auf der Festplatte (Cluster) adressiert werden. Bei FAT16 sind das somit 2^{16} (65536) Cluster. Außer der Implementierung von FAT16 existieren FAT12 und FAT32. FAT16 wurde 1983 veröffentlicht, da die Adressierung seines Vorgängers (FAT12) mit 12 Bit nicht mehr für die damaligen Festplattengrößen ausreichte. Eine in FAT16 formatierte Festplatte gliedert sich in fünf Bereiche auf, die im Folgenden genauer erläutert werden.

Bootsektor Befindet sich auf den ersten 32 Byte der Festplatte. Hier werden einige generelle Definitionen festgelegt. Beispielsweise die Sektor-Größe der zugrunde liegenden Festplatte oder die Anzahl an File Allocation Tables auf der Festplatte. Weiterhin wird angegeben, um welches Dateisystem es sich bei der Festplatte handelt. Am ende des Bootsektors befindet sich der Bootloader.

Reservierte Sektoren Die Anzahl der reservierten Sektoren wird im Bootsektor spezifiziert. Hierzu zählt ebenfalls der Bootloader aus dem Bootsektor.

File Allocation Table Die FAT vermerkt welche Cluster zu welcher Datei gehören und, ob bestimmte Cluster leer sind. Es gibt meist mehrere Kopien der FAT, um Fehler durch Redundanz zu vermeiden. Die Verweise auf die Cluster beinhalten im letzen Byte eine Referenz auf weiterführende Cluster der selben Datei.

Wurzelverzeichnis Das Wurzelverzeichnis ist ein bestimmter Bereich auf dem Datenträger. Es besitzt eine Tabelle mit Verzeichniseinträgen (so wie jedes Verzeichnis in FAT). Seine Größe wird statisch festgelegt.

Datenregion Die Dateiregion beinhaltet alle Dateien und Verzeichnisse, die auf dem Datenträger vorhanden sind. Sie ist in Cluster unterteilt.

Wie in der Beschreibung zum Wurzelverzeichnis schon erwähnt, besitzt jedes Verzeichnis eine Tabelle mit Verzeichniseinträgen. Diese Einträge enthalten Meta-Informationen zu den Dateien im Verzeichnis. Beispiele für Informationen sind Dateiname, Dateiendung oder auch Erstellungsdatum.

5.2 Virtuelles Dateisystem – VFS

In diesem Abschnitt wird auf die Analyse und den Entwurf des virtuellen Dateisystems eingegangen.

5.2.1 Analyse

Die Aufgabenstellung fordert die Implementierung eines virtuellen Dateisystemes wie in den Grundlagen (Abschnitt 5.1.1) beschrieben. Ziel ist es eine Abstraktionsschicht zu schaffen, mit der verschiedene Dateisystem-Implementierungen eingebunden werden können.

Das VFS liefert dazu Schnittstellen mit Operationen, die vom konkreten Dateisystem implementiert werden müssen. Dies gewährleistet einen einheitlichen Zugriff auf alle eingehangenen Dateisysteme.

Die Organisation der Dateien und Verzeichnisse eines VFS lässt sich durch eine hierarchische Baumstruktur umsetzten. Jeder Einhängepunkt besitzt eine Wurzel (ein Verzeichnis), welches weitere Verzeichnisse und Dateien beinhaltet. Dabei haben alle Knoten weitere Eigenschaften wie Datei-/Verzeichnisnamen, Elternknoten, Dateigröße und weitere Attribute.

Bei einer objektorientierten Realisierung eines VFS ist es naheliegend, die Struktur über das Kompositum-Entwurfsmuster zu realisieren. Dabei ist jedoch zu beachten, dass beim klassischen Kompositum alle (abstrakten) Verzeichnisse und Dateien im Speicher gehalten werden müssen[35]. Es wurde sich deshalb für eine Variante des Lazy-Loading entschieden. Die einzelnen Inhalte eines Verzeichnisses werden erst dann vom Festspeicher geladen, wenn sie einzeln angesprochen oder aufgelistet werden sollen.

Zusätzlich sind Schnittstellen zum Ein- und Aushängen von Dateisystemen notwendig. Wie in den Grundlagen beschrieben, gibt es zwei Möglichkeiten: Ein Kompositum für alle Dateisysteme oder ein Kompositum pro Dateisystem. Hierbei wird sich für die zweite Möglichkeit entschieden, da diese einfacher zu implementieren ist.

[35] Natürlich wird nur die Organisation (der Baum) der Verzeichnisse und Dateien im Speicher gehalten, nicht jedoch die einzelnen Inhalte der Dateien

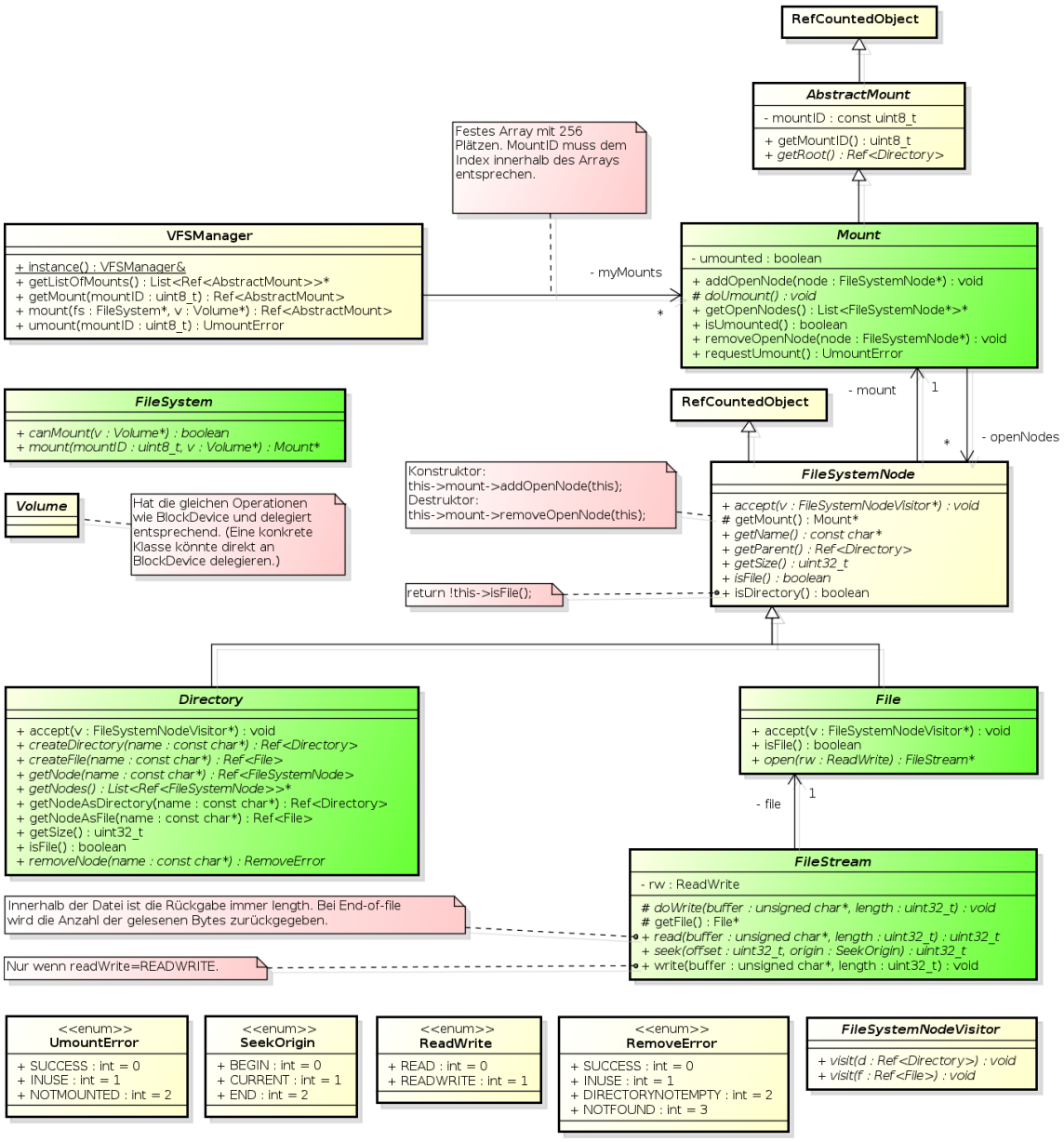


Abbildung 5.3: VFS Klassendiagramm

5.2.2 Entwurf

Der entstandene Entwurf beschreibt eine Möglichkeit, ein VFS zu realisieren. In Abbildung 5.3 ist das vollständige Klassendiagramm abgebildet.

Die zentrale Komponente innerhalb des VFS ist der VFSManager, der sowohl für das Ein- als auch für das Aushängen von Dateisystemen zuständig ist. Darüber hinaus verwaltet er alle eingehängten Dateisysteme in einem Array bestehend aus 256 Einträgen. Dieses Array beschränkt die Anzahl der gleichzeitig zu verwaltenden Dateisysteme auf 256. Jedes eingehängte Dateisystem in diesem Array erhält eine eindeutige „mountID“, welche mit dem Index innerhalb des Arrays korrespondiert. Durch diese „mountID“ ist ein gezielter und sehr schneller Zugriff auf das Array möglich.

Um sensible Operationen, wie das Aushängen von Dateisystemen, vor Zugriffen zu schützen, gibt der VFSManager immer nur AbstractMount-Objekte zurück.

Bei der Umsetzung der Baumstruktur über die FileSystemNode haben wir uns, wie in der Analyse beschrieben, bewusst gegen die Nutzung des Kompositum-Entwurfsmusters entschieden. Die Operation `getParent()` sollte beim Wurzelverzeichnis `NULL` zurückgeben.

Über Verzeichnisse ist es möglich, Kinder (also Dateien oder Unterverzeichnisse) zu löschen. Dies darf nur dann passieren, wenn das Kind-Element sich nicht mehr im Speicher befindet. Nichtleere Verzeichnisse können nicht gelöscht werden.

Um ein Dateisystem wieder auszuhängen ist es notwendig, dass alle Dateien geschlossen wurden und sich keine Dateien mehr im Speicher befinden. Da die Verwaltung offener Dateien und Verzeichnisse sehr aufwendig werden kann, wurden referenzzählende Objekte eingeführt. Diese Objekte rufen automatisch den Destruktor auf und geben den Speicher frei, sobald das Objekt von niemandem mehr Referenziert wird.

Das Lesen und Schreiben geschieht über die Klasse „FileStream“, welche gängige Operationen bietet.

Alle im Klassendiagramm grün eingefärbten Klassen müssen von einer konkreten Dateisystemimplementation implementiert werden.

5.2.3 Fazit

Es wurden alle in Tabelle 5.1 aufgelisteten Operationen umgesetzt. Teilweise erhielten sie neue Namen und wurden in die entsprechenden Klassen verschoben. Die Operation `close` muss in einer objektorientierten Umgebung nicht umgesetzt werden, dies übernimmt der Destruktor.

Zum Testen des VFS wurde ein virtuelles Dateisystem implementiert.

5.3 FAT16

5.3.1 Analyse

Während der Analysephase werden drei wichtige Aspekte untersucht, bevor ein Entwurf für das zu entwickelnde FAT16 Dateisystem erstellt wird. Im ersten Schritt

erfolgt eine Einarbeitung in die Grundlagen, welche bereits im Vorfeld beschrieben wurden. Anschließend erfolgt eine Analyse des entwickelten ATA-Treibers, der die Basis für das Dateisystem darstellt. Im letzten Schritt wird untersucht, welche Auswirkungen der bereits Erstellte Entwurf des VFS (Virtual File System) für das FAT-Dateisystem hat. Das folgende Schichtenmodell veranschaulicht die Positionen der einzelnen Ebenen:

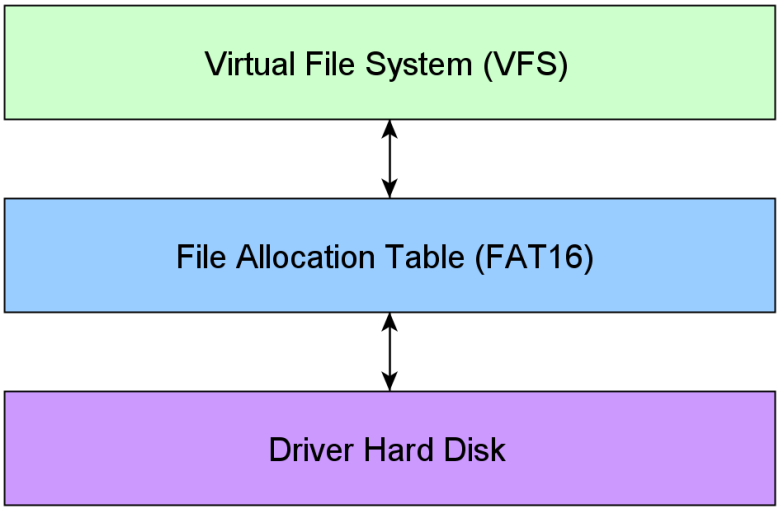


Abbildung 5.4: Schichtenmodel VFS, FAT, ATA-Treiber

Die Schnittstellen zum Treiber der Festplatte bietet die Klasse BlockDevice aus der Driver-Hard-Disk-Ebene. Hier werden alle Operationen bereitgestellt, die vom FAT16-Dateisystem benötigt werden. Die folgenden Funktionen werden vom Treiber genutzt:

- isReadOnly (Schaut ob von dem Gerät ausschließlich gelesen werden kann)
- read (Liest den Block an einer bestimmten Adresse)
- write (Schreibt den Block an einer bestimmten Adresse)
- reads (Liest eine gewünschte Anzahl von Blöcken ab einer Adresse)
- writes (Schreibt eine Anzahl von Blöcken ab einer bestimmten Adresse)
- maximalNumberOfBlocksToReadOrWriteAtOnce (Gibt die gesamte Anzahl der Blöcke zurück, die gelesen oder geschrieben werden können)
- getMinAddress (Liefert die niedrigste Adresse des Geräts zurück)
- getMaxAddress (Liefert die höchste Adresse des Geräts zurück)

Der Entwurf für das Virtual File System hat ergeben, dass folgende Klassen spezialisiert und anschließend die Methoden dieser Klasse implementiert werden müssen:

- FileSystem
- Mount
- Directory
- File
- FileStream

Die folgenden Gleichungen helfen bei der späteren Implementierung für die Ermittlung einer Sektornummer aus einer Clusternummer, die Position des ersten Datensektors und die Anzahl der Sektoren im Wurzelverzeichnis.
Um von einer Cluster-Nummer auf den Wert einer Sektor-Nummer zu kommen, in dem sich die eigentlichen Daten befinden, wird die folgende Gleichung verwendet:

$$SektorNum = ((ClusterNum - 2) * ClusterGroesseInSektoren) + PosErsterDatensektor$$

Die Cluster-Nummer wird im zwei verringert, da es keine Cluster null und eins mit Daten gibt. Daher dürfen in Verzeichniseinträgen die Cluster-Nummern nur größer oder gleich zwei sein sofern eine Datei Daten enthält. Die anschließende Multiplikation ist logisch, kann durchaus je nach Definition unterschiedlich sein. Die Addition ist besonders wichtig, da sich vor den Nutzdaten noch der Bootsektor, die File Allocation Tables und das Wurzelverzeichnis befinden. Die Position des Datensektor ist konstant und muss nur einmalig beim Mounten berechnet werden.

Folgend wird beschrieben, wie die konstante Position des ersten Datensektors berechnet wird:

$$PosErsterDatensektor = AnzReservierterSektoren + (AnzahlFATSektoren * AnzFATKopien) + AnzWurzelverzeichnisSektoren$$

Die gesamten Werte, die für diese Berechnung notwendig sind, können dem Bootsektor entnommen werden. Da auch der Bootsektor zu den reservierten Sektoren zählt, ist die Anzahl dieser mindestens eins, denn der Bootsektor muss immer vorhanden sein. Die Anzahl der FAT-Sektoren ergibt sich aus der Größe der FAT und diese muss je nach dem wie viele Kopien es von der FAT gibt multipliziert werden. Die Anzahl der Sektoren für das Wurzelverzeichnis muss ebenfalls berechnet und anschließend addiert werden.

Die Anschließende Formel beschreibt, wie die Berechnung für die Anzahl der Sektoren des Wurzelverzeichnisses durchgeführt wird:

$$AnzWurzelverzeichnisSektoren =$$

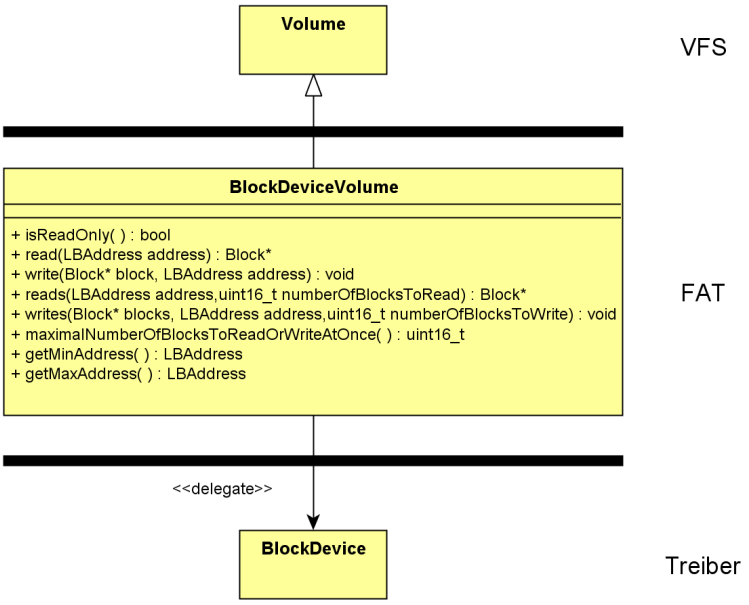


Abbildung 5.5: Klassendiagramm - BlockDeviceVolume

$AnzWurzelverzeichniseintrge/(GroesseSektorenInByte/32)$

Die Informationen hierfür können wie schon zuvor dem Bootsektor entnommen werden.

5.3.2 Entwurf

Die Entwurfsphase stellt die Basis der Implementierung dar und definiert alle zu implementierenden Klassen inklusive ihrer Assoziationen.

Im ersten Teil dieses Abschnitts wird beschrieben, wie die Relation zum Treiber der Festplatte realisiert wird. Hierfür ist die Klasse BlockDeviceVolume zuständig, diese spezialisiert die Klasse Volume aus dem Virtual File System und delegiert von allen Operationen aus der Klasse BlockDeviceVolume zur Klasse BlockDevice vom Festplattentreiber. Die zweite Abbildung veranschaulicht dies.

Das sogenannte Mounten (Einhängen) der Festplatte wird FATFileSystem gesteuert. Die Klasse spezialisiert die Klasse FileSystem aus dem VFS bietet die Operationen canMount und Mount an. Bei der Operation canMount wird vor dem eigentlichen Mounten geprüft, ob es möglich ist, diese Festplatte zu mounten. Hierbei wird geprüft, ob sich auf dem Datenträger das FAT16-Dateisystem befindet. Ist dieses der Fall, kann die Festplatte über die Operation Mount eingehängt werden. Dafür wird ein Objekt der Klasse FATMount erzeugt, welches die Klasse Mount des VFS spezialisiert und dieses führt den Vorgang des Mounten durch. Des Weiteren wird beim Mount-Vorgang ein Objekt der Klasse BootSector er-

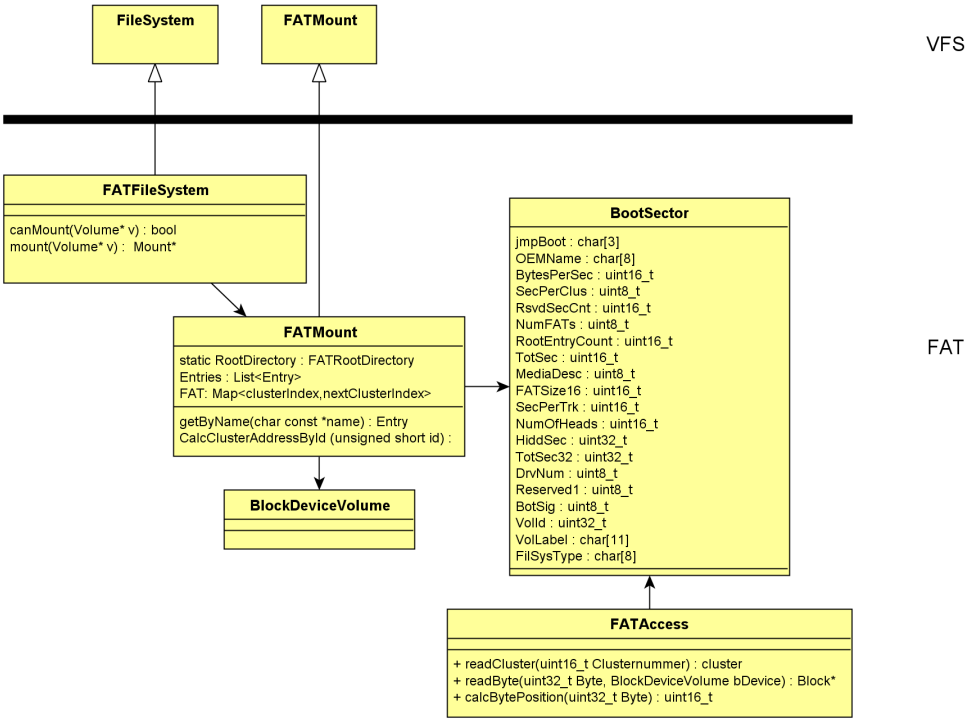


Abbildung 5.6: Klassendiagramm - FATFileSystem

zeugt, um anschließend alle Attribute darin mit den Informationen der Festplatte zu Füllen. Zusätzlich existiert noch eine Klasse FATAccess, die Hilfsoperationen für Berechnungen und das Lesen anbietet. In der dritten Abbildung wird dies veranschaulicht.

In den Grundlagen wurde bereits beschrieben, dass bei einem FAT16-Dateisystem das Wurzelverzeichnis eine feste Größe hat und diese nach der Definition nicht mehr verändert werden kann. Aus diesem Grund gibt es eine gesonderte Klasse FATRootDirectory, die das Wurzelverzeichnis repräsentiert. Alle weiteren Verzeichnisse sind Objekte der Klasse FATDirectory. Beide zuvor genannten Klassen spezialisieren die Klasse Directory aus dem VFS und besitzen identische Operationen. Über diese Operationen kann der Name, die Größe, alle Objekte und übergeordnete Objekte eines Verzeichnisses ermittelt werden. Des Weiteren können Dateien und Verzeichnisse erstellt oder gelöscht werden. Für Datei-Objekte ist die Klasse FATFile verantwortlich, sie bietet Operationen zum öffnen und löschen von Dateien an. Zusätzlich kann der Name, die Größe und der Vater der Datei ermittelt werden. Da ein Verzeichnis auch eine Datei mit leerem Inhalt darstellt spezialisieren sowohl die Klasse FATFile als auch die Klasse FATDirectory die Klasse FATObject. Somit gibt es eine gemeinsame Oberklasse in der Hilfsoperationen für die Iteration angeboten werden. Es liegt also eine Mehrfachvererbung vor. Veranschaulicht wird das in Abbildung 4.

Die zuvor beschrieben Klassen für Verzeichnisse (FATDirectory und FATRootDirec-

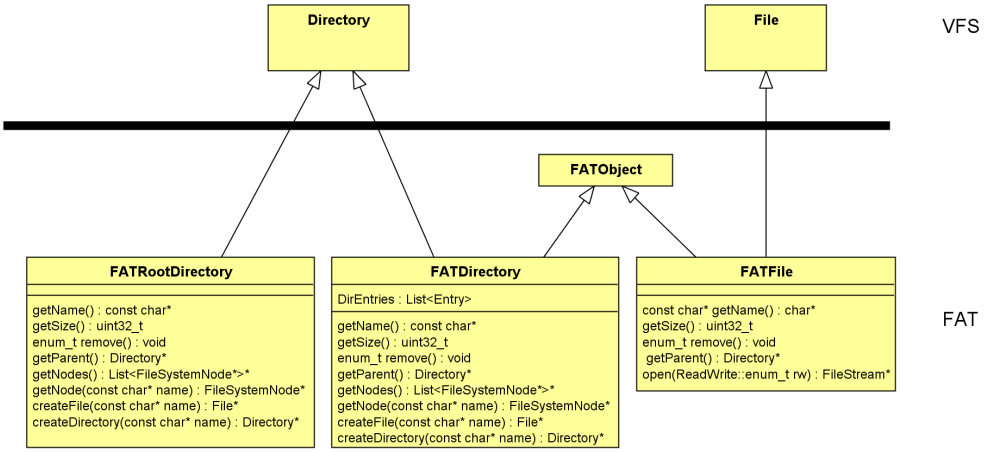


Abbildung 5.7: Klassendiagramm - FATObject

tory) haben jeweils eine Liste in der sich Objekte der Klasse Entry befinden. Diese repräsentieren einen Verzeichnis-Eintrag und enthalten Meta-Daten zu einer Datei bzw. einem Unterverzeichnis. Außerdem enthält ein Eintrag die Nummer des ersten Clusters der Datei. Die folgende Auflistung zeigt, welche Meta-Daten in einem solchen Eintrag enthalten sind:

- Dateiname
- Dateierweiterung
- Dateiattribute (Schreibschutz, Versteckt, Systemdatei, Volume-Label...)
- Erstelldatum
- Datum des letzten Zugriffs
- Datum des letzten Schreibzugriffs
- Erster Cluster
- Dateigröße

Das Klassendiagramm in Abbildung 5 zeigt den Aufbau der Klasse Entry. An dieser Stelle sei darauf hingewiesen, dass die Klasse Entry ausschließlich zur Veranschaulichung in drei Teile aufgeteilt ist. In der Implementierung ist alles in einer Klasse realisiert.

Im folgenden Abschnitt wird beschrieben wie das Lesen und Schreiben einer Datei funktioniert.

Lesen einer Datei:

Schreiben einer Datei:

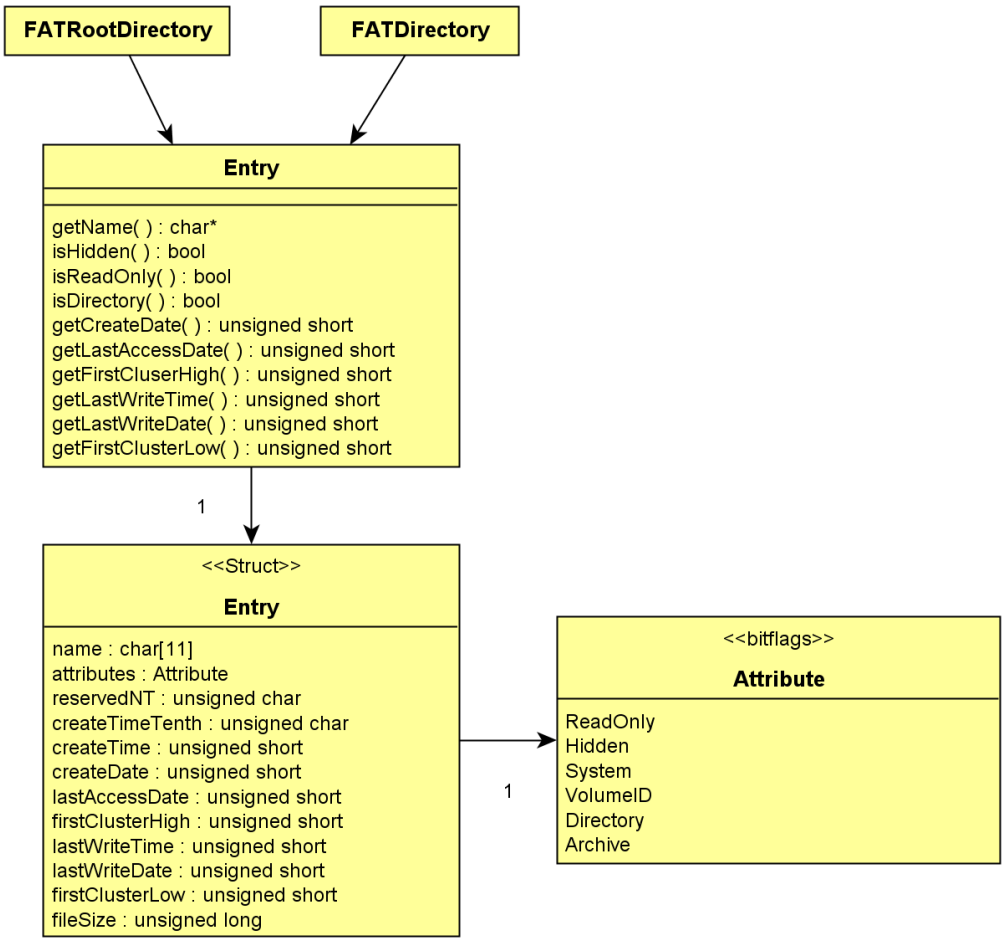


Abbildung 5.8: Klassendiagramm - Entry

Teil II

Interrupts und Ausnahmen

6 Interrupts und Ausnahmen – Analyse

6.1 Allgemein

Unterbrechungen und Ausnahmen sind Anforderungen von beliebiger Stelle im System die anzeigen, dass die Aufmerksamkeit des Prozessors an anderer Stelle im System benötigt wird. Typischerweise bedeutet das, dass das aktuell ausgeführte Programm unterbrochen wird und ein spezielles Programm (Unterbrechungs- bzw. Ausnahmenbehandler) aufgerufen wird, der die Anforderung abarbeitet. Wenn die Anforderung abgearbeitet wurde, wird das Programm an der Stelle und mit dem Status an dem es unterbrochen wurde weiter ausgeführt – außer das aktuelle Programm wurde im Behandler beendet.

Um mehrere Unterbrechungs- und Ausnahmetypen zu definieren gibt es eine Identifikationsnummer (Vektornummer). Jedem Typ kann ein spezieller Behandler zugewiesen werden, in dem eine an die Unterbrechung bzw. an die Ausnahme angepasste Abarbeitung stattfindet. Es stehen 256 unterschiedliche Vektornummern von 0 – 255 zur Verfügung. Dabei sind die Nummer von 0 – 31 schon fest für bestimme Unterbrechungen und Ausnahmen zugewiesen. Die Nummern von 32 – 255 stehen zur freien Verfügung.

6.2 IDT

Die Interrupt Descriptor Table (IDT) ordnet jeder Unterbrechung oder jeder Ausnahme einen Gate Deskriptor für die mit der Unterbrechung oder Ausnahme assoziierten Prozedur zu. Einfacher ausgedrückt ist in der IDT festgelegt, an welcher Stelle es weitergeht falls eine Unterbrechng oder eine Ausnahme auftritt. Jede Unterbrechung und jede Ausnahme in der IDT ist mit einer Nummer identifiziert, die Vektor genannt wird. Die IDT kann bis zu 256 Vektoren beinhalten. Bei nicht benötigten Einträgen sollte das Present-Flag auf 0 gesetzt sein. Die IA-32

Architektur hat 18 vordefinierte Unterbrechungen und Ausnahmen.

Die IDT darf an einer beliebigen Stelle im linearen Adressraum gespeichert sein. Der Prozessor lokalisiert die IDT indem er das IDT-Register (IDTR) benutzt. Dieses Register beinhaltet eine 32bit Basisadresse und ein 16bit Limit für die IDT. In der Abbildung 6.1 ist die Beziehung zwischen der IDT und dem IDTR grafisch dargestellt.

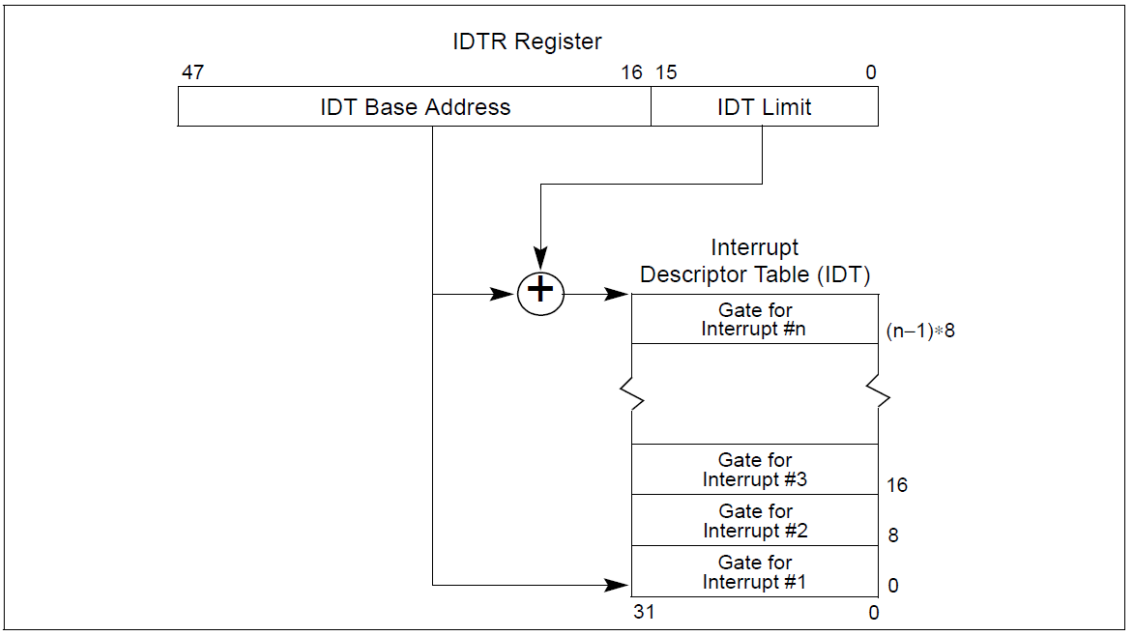


Abbildung 6.1: Beziehung zwischen IDT und dem IDTR

Mit den Anweisungen *LIDT* (load IDT register) und *SIDT* (store IDT register) werden die Inhalte des IDTR geladen und gespeichert. Die *LIDT* Anweisung lädt das IDTR mit einer Basisadresse und einem Limit, das in einem Speicheroperanden gehalten wird. Diese Anweisung kann nur ausgeführt werden wenn CPL 0 ist und wird normalerweise von dem Initialisierungscode eines Betriebssystems benutzt, wenn die IDT erstellt wird. Ein Betriebssystem kann sie auch benutzen um von einer IDT zu einer anderen zu wechseln. Die *SIDT* Anweisung kopiert den Basiswert und den Limitwert die in der IDTR gespeichert sind in den Speicher. Diese Anweisung kann von jeder Privilegstufe ausgeführt werden.

6.3 Unterbrechungen

Unterbrechungen können während der laufenden Ausführung eines Programms auftreten. Es handelt sich hierbei um Zustände, die nicht im Prozessor selber auftreten sondern an ihn gemeldet werden.

6.3.1 Quellen

Unterbrechungen können von zwei verschiedenen Quellen ausgelöst werden: extern von weiterer verbauter Hardware oder von der ausgeführten Software. Verfügt das System über mehrere Prozessoren so können diese untereinander Unterbrechungen auslösen.

6.3.2 Externe Unterbrechungen

Externe Unterbrechungen werden über den INTR Pin an der CPU oder den lokale APIC (Advanced Programmable Interrupt Controller) an den Prozessor gegeben. Der Prozessor wertet daraufhin die Vektornummer der Unterbrechung aus und führt den dazu passenden Behandler aus. Welche Prozedur das ist wird in der Interrupt Descriptor Table (IDT)[36] festgelegt. Die im System verbaute Hardware nutzt Unterbrechungen unter anderem dazu um Ereignisse an den Prozessor zu melden. Beispielsweise kann eine Unterbrechung ausgelöst werden um Ressourcen vom Prozessor anzufordern. Auch fordert die Tastatur über eine Unterbrechung die Verarbeitung einer Eingabe an.

Maskierbare Unterbrechungen können über das *Interrupt enable Flag* (IF) im EFLAGS Register „maskiert“ werden. Das bedeutet, dass wenn eine maskierter Unterbrechung auftritt, der Prozessor ihn nicht abarbeitet, sondern einfach die derzeitige Anweisung weiterhin ausführt. Das ist nützlich für Vorgänge in denen keine Unterbrechung auftreten darf. Bis auf die Unterbrechung mit der Vektornummer 2 sind alle Unterbrechungen maskierbar.[37] Das IF kann mit dem Befehl *STI* (set interrupt-enabled flag) gesetzt und mit dem *CLI* (clear interrupt-enabled flag) gelöscht werden. Wird eine Unterbrechung abgearbeitet, so wird das IF gelöscht und die Interrupts werden maskiert.

Im Gegensatz zu maskierbaren Unterbrechungen hat das IF keine Auswirkung auf **nicht maskierbare Unterbrechungen** (NMI). Diese können vom Prozessor nicht „ignoriert“ werden und müssen abgearbeitet werden. NMI haben immer die Vektornummer 2 in der IDT. Während ein NMI bearbeitet wird, stellt der Prozessor sicher, dass keine andere Unterbrechung (NMIs eingeschlossen) die Ausführung unterbrechen kann.

[36] siehe Kapitel 6.2
[37] siehe Abschnitt 6.3.2

6.3.3 Software-generierte Unterbrechungen

Um Unterbrechungen von der Software auszulösen gibt es den Befehl *INT n*. Das *n* steht dabei für die Vektornummer in der IDT. Unterbrechungen die so ausgelöst werden können nicht maskiert werden.
Wird die Unterbrechung für einen NMI aufgerufen (*INT 2*) so reagiert der Prozessor anders, als wenn dieser durch die Hardware ausgelöst wurde. Zwar wird der korrekte Benandler aufgerufen, jedoch die Hardware, die im Prozessor die NMI behandelt, wird nicht aktiviert.

6.4 Ausnahmen

Ausnahmen sind spezielle Unterbrechungen. Sie fordern den Prozessor auf, das aktuelle Programm zu unterbrechen, wenn der Prozessor während der Abarbeitung einen Fehler entdeckt. Beispiele hierfür sind:

- Division durch 0
- Schutzverletzungen
- Seitenfehler
- Hardware- oder Busfehler, die durch die Maschinenüberprüfungsarchitektur ermittelt wurden

6.4.1 Quellen

Ausnahmen können durch die drei folgenden Möglichkeiten vom Prozessor ausgelöst werden:

Programmfehler Die erste Möglichkeit, bei der Ausnahmen durch den Prozessor ausgelöst werden können, ist während der Ausführung eines beliebigen Programmes. In diesem Fall werden Ausnahmen ausgelöst, wenn bei der Ausführung eines Befehls durch den Prozessor ein Fehler auftritt.

Software-Generiert Bei der zweiten Möglichkeit werden Ausnahmen durch den Aufruf einer Instruktion ausgelöst. Die Instruktionen *INTO*, *INT 3* und *BOUND* lösen eine Ausnahme aus. Bei den Instruktionen *INTO* und *BOUND* wird nur eine Ausnahme ausgelöst, wenn eine spezielle Fehlerbedingung zutrifft. Zum Beispiel wird bei der *INTO* Instruktion überprüft, ob ein Überlauf aufgetreten ist und im positiven Fall eine Overflow-Exception ausgelöst.

Maschinenüberprüfung Die letzte Möglichkeit, dass eine Ausnahme ausgelöst werden kann ist über den Maschinenüberprüfungsmechanismus der eine Maschinenüberprüfungsausnahme auslöst, falls der Mechanismus einen Fehler in der Hardware oder in einem Bus entdeckt.

6.4.2 Ausnahmenklassifizierung

Ausnahmen sind in drei unterschiedliche Klassen eingeteilt. Diese unterscheiden sich darin, welche und wie viele Informationen zur Verfügung gestellt werden um den Programmablauf fortzusetzen, also an welche Stelle im Programmablauf zurückgesprungen wird, wenn die Abarbeitung des Ausnahmenbehandlers beendet ist.

Fault Ausnahmen vom Typ Fault treten während der Ausführung einer Instruktion auf. Daraufhin wird der komplette Maschinenstatus abgespeichert, der vor der Ausführung der fehlgeschlagenen Instruktion gültig war. Das bedeutet, dass nachdem der Ausnahmebehandler abgearbeitet worden ist, die fehlgeschlagene Instruktion noch einmal ausgeführt werden kann. Ein typisches Beispiel für eine Fault-Ausnahme ist ein Seitenfehler. Dieser tritt auf, wenn ein Operand einer Instruktion nicht verfügbar ist. In diesem Fall kann der Ausnahmebehandler, der für einen Seitenfehler zuständig ist, den fehlenden Operanden nachladen und danach den Programmablauf an der fehlgeschlagenen Instruktion wieder anstarten.

Trap Ausnahmen vom Typ Trap werden direkt nach der Ausführung einer fehlschlagenden Instruktion ausgelöst. In diesem Fall wird der Maschinenstatus, der nach der Ausführung der fehlgeschlagenen Instruktion aktuell ist, abgespeichert. Somit wird das Programm, wenn aus dem Ausnahmebehandler zurückgekehrt wird, mit der Instruktion fortgesetzt, die direkt auf die fehlgeschlagene Instruktion folgt. Das hat zum Beispiel zur Folge, dass wenn bei der Ausführung eines *JMP* eine Trap-Ausnahme auftritt, nach dem Zurückkehren aus dem Ausnahmebehandler mit dem Ziel der *JMP* Instruktion fortgefahren wird.

Abort Ausnahmen vom Typ Abort treten nicht an einer präzisen Stelle auf. Somit ist es auch nicht möglich einen bestimmten Maschine-Status zu ermitteln. Abort-Ausnahmen werden meist dazu genutzt, Hardwarefehler oder nicht erlaubte Zustände in Systemtabellen zu berichten. Die Behandler von Abort-Ausnahmen werden meist dazu genutzt, Diagnoseinformationen zu sammeln und auszugeben und darauf bestmöglich den Programmablauf zu beenden und das System herunterzufahren.

6.5 Parallel auftretende Unterbrechungen

Wenn mehrere Unterbrechungen oder Ausnahmen gleichzeitig auftreten, werden die mit der höchsten Priorität bevorzugt abgearbeitet. Gleichzeitiges Auftreten bedeutet hierbei, dass während einer Instruktion mehrere Unterbrechungen ausgelöst wurden oder, dass durch die Abschaltung der Unterbrechungen während der Behandlung einer NMI sich mehrere Unterbrechungen angesammelt haben. Unterbrechungen und Ausnahmen sind fest 10 verschiedenen Prioritäten zugeordnet, die nicht angepasst werden können. Eine Übersicht über die Einteilung der Prioritäten ist in Tabelle 6.1 zu finden.

Priorität	Beschreibung
1 (Höchste)	Hardware Reset und Machine Check
2	Trap bei einem Task Switch
3	Externer Hardware-Eingriff
4	Trap in der vorherigen Instruktion (Haltepunkte, Debug)
5	Nicht maskierbare Unterbrechungen
6	Maskierbare Hardware Unterbrechungen
7	Code Breakpoint Fehler
8	Fehler beim Ermitteln der nächsten Instruktion
9	Fehler beim Decodieren der nächsten Instruktion
10 (Niedrigste)	Fehler beim Ausführen einer Instruktion

Tabelle 6.1: Prioritäten bei gleichzeitig auftreten Unterbrechungen und Ausnahmen

6.6 Unterbrechungen und Ausnahmen Referenz

Vektornummer	Mnemonic	Beschreibung	Typ
0	#DE	Division durch 0 *DIV* und *IDIV*	Fault
1	#DB	Debug	Fault/Trap
2		NMI Interrupt	Interrupt
3	#BP	Breakpoint *INT 3*	Trap
4	#OF	Overflow *INTO*	Trap
5	#BR	Bound Bereichsüberschreitung *BOUND*	Fault
6	#UD	Ungültiger Opcode	Fault
7	#NM	Gerät nicht verfügbar	Fault
8	#DF	Doppelter Fehler	Abort
9		Coprozessor Segment Überlauf	Fault
10	#TS	Ungültiger Task-Switch	Fault
11	#NP	Segment nicht vorhanden	Fault
12	#SS	Stack-Segment Fehler	Fault
13	#GP	Allgemeine Schutzverletzung	Fault
14	#PF	Seitenfehler	Fault
15		Von Intel reserviert	-
16	#MF	x86 FPU Floating-Point Fehler	Fault
17	#AC	Alignment Überprüfung	Fault
18	#MC	Machine-Check	Abort
19	#XM	SIMD FLoating-Point Fehler	Fault
20-31		Von Intel reserviert	-
32-255		Benutzerdefiniert Interrupt	Interrupt

Tabelle 6.2: Liste der Vektornummernzuordnungen

#DE Tritt auf, wenn das Ergebnis der DIV oder IDIV Instruktion nicht in das Ergebnisregister passt

#DB Ein oder mehrere Bedingungen für eine Debug-Ausnahme wurden gefunden

#BP Zeigt an, dass ein Haltepunkt erreicht wurde

#OF Wird ausgelöst, wenn ein Überlauf aufgetreten ist und die INTO Instruktion ausgeführt wurde

#BR Tritt auf, wenn die BOUND Instruktion aufgerufen wurde und der Index dieser nicht in der Reichweite vom Array liegt

#UD Zeigt an, dass eine Instruktion ausgeführt wurde, die eine nicht verfügbare oder abgeschaltete Funktion benötigt.

#NM Floating Point Unit (FPU) nicht verfügbar

#DF Wird ausgelöst, wenn während dem Aufruf eines priorisierten Behandlers eine weitere Unterbrechung auftritt

Coprozessor Segmentüberlauf Wird bei Intel386 Prozessoren ausgelöst, wenn im mathematischen Coprozessor eine Seiten- oder Segment-Verletzung auftritt

#TS Eine Ausnahme die anzeigt, dass bei einem Taskwechsel ein Fehler aufgetreten ist.

#NP Diese Ausnahme tritt auf wenn auf ein Segment zugegriffen wird bei dem das Verfügbar-Flag nicht gesetzt ist.

#SS Zeigt an dass bei einer Instruktion die in Beziehung mit dem Stack steht ein Fehler aufgetreten ist. Zum Beispiel wenn *POP* auf den leeren Stack ausgeführt wird.

#GP Ausnahme, die anzeigt, dass eine allgemeine Zugriffsverletzung stattgefunden hat.

#PF Page-Fault Ausnahmen treten auf, wenn beim Pagingmechanismus ein Fehler auftritt, wie zum Beispiel ein Zugriff auf eine Page die nicht verfügbar ist.

#MF Zeigt an, dass ein Fehler in der FPU aufgetreten ist.

#AC Zeigt an, dass der Prozessor einen nicht angepassten Speicheroperanden gefunden hat.

#MC Ausnahmen dieses Typs treten auf wenn ein Maschinenfehler oder Busfehler ermittelt wurde.

#XM Zeigt an, dass bei der Ausführung einer SIMD-Fließkommainstruktion ein Fehler aufgetreten ist.

7 Interrupts und Ausnahmen – Entwurf

7.1 Analyse

Die Verwaltung von Interrupt- und Ausnahmebehandlern erfolgt an zentraler Stelle im Kernel. Diese Verwaltung ermöglicht das Hinzufügen und Entfernen von Behandlern und koordiniert den Aufruf der Behandler beim Auftreten eines Interrupts bzw. einer Ausnahme.

Während das Auftreten eines Interrupts unbehandelt bleiben kann, ist das Auftreten einer Ausnahme kritisch für die Ausführung des Systems. Die Implementierung der Behandler sowie deren Aufruf muss entsprechend konzipiert sein. Es können sich beliebig viele Behandler an derselben Vektornummer anmelden. Meldet sich ein Behandler an einer Vektornummer an, für die bereits ein Behandler registriert ist wird er an die erste Stelle gesetzt. Soll nun ein Interrupt behandelt werden wird der zuletzt hinzugefügte Behandler zuerst ausgeführt. Solange der Interrupt nicht erfolgreich behandelt wurde wird der jeweils nächste Behandler aufgerufen bis letztendlich der Standardbehandler den Kern mit einer Fehlermeldung herunterfährt.

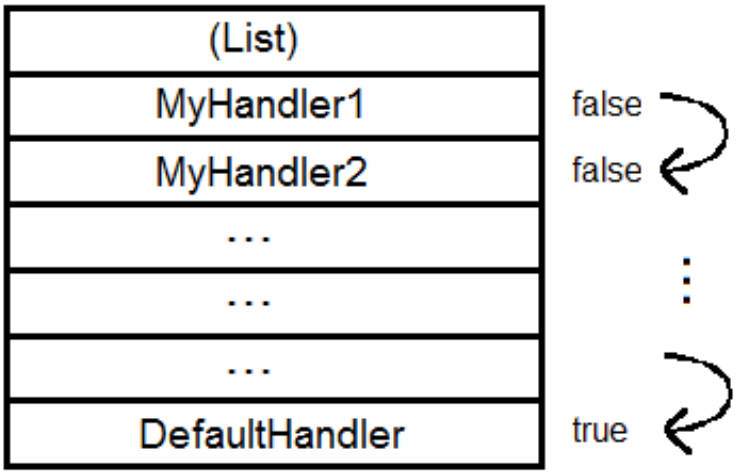


Abbildung 7.1: Organisation der Behandler

Der Intel 8259A ist ein Programmierbarer Interrupt Controller (PIC), der bis zu acht priorisierte Interrupt für die CPU behandeln kann. Er ist ohne weitere Schaltkreise auf bis zu 64 Interrupts kaskadierbar. Durch den 8259A soll Aufwand bei der Software minimiert werden. Der PIC fungiert als Manager. Er akzeptiert Anfragen von Peripheriegeräten, entscheidet welche der eingehenden Anfragen am wichtigsten ist und bestimmt ob eine eigehende Anfrage eine höhere Priorität hat als der momentan bearbeitete Prioritätslevel. Jedes Peripheriegerät hat normalerweise ein spezielles Programm oder Routine die mit den gerätespezifischen Funktionen zusammenhängt. Nachdem der PIC einen Interrupt an die CPU gemeldet hat muss er der CPU Informationen geben, die den ProgramCounter auf die mit dem Gerät assoziierten Service Routine setzt. Dieser Pointer ist eine Adresse in einer Vektortabelle, auch als Vektordaten bezeichnet.

7.2 Entwurf

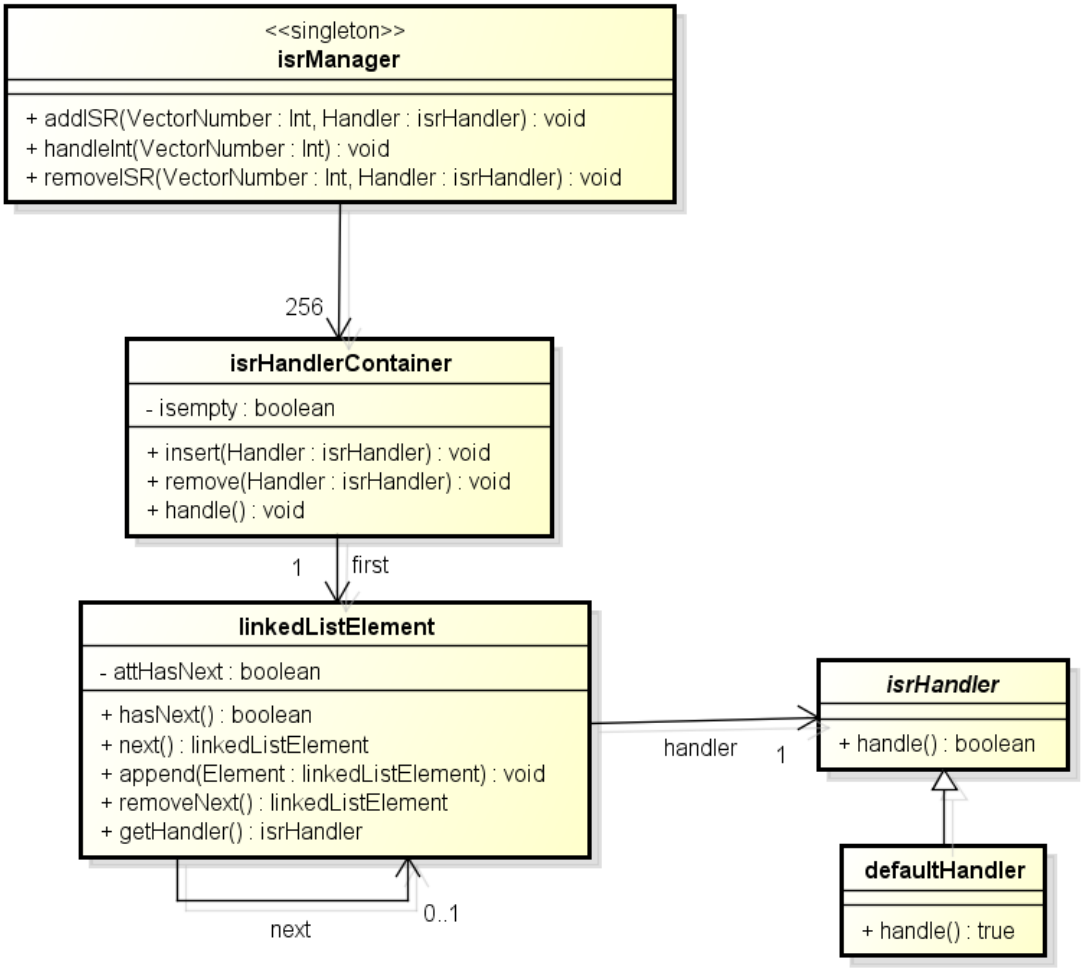


Abbildung 7.2: Klassendiagramm

7.2.1 isrManager

Als Verwaltung für Interrupts fungiert die Klasse ISR-Manager. Er ist die zentrale Schnittstelle für das Hinzufügen, Löschen und Behandeln von Interrupts. Sie kennt alle Vektornummern und stellt die Operationen `addISR`, `handleInt` und `removeISR` bereit und delegiert Aufrufe von außerhalb an die zugehörigen Klassen weiter. Der ISR-Manager ist nach dem Entwurfsmuster **Singleton** umgesetzt, da es wichtig ist, dass dieser nur einmal existiert. Hinter jeder Vektornummer verbirgt sich ein Container mit Unterbrechungs-Behandlern. Dieser Container besitzt intern wiederum eine Listenstruktur. Er kennt das erste Element der Liste und dieses Element kennt wieder das nächste, falls vorhanden. So können beliebig viele Elemente (Behandler) zu den einzelnen Vektornummern zugeordnet werden. Die Listenelemente zeigen auf den zugehörigen Behandler und haben einen Verweis auf das folgende Element, falls es existiert. Darüber kann die Liste abgearbeitet werden. Für jeden Container ist immer ein Standardbehandler vorhanden, der aufgerufen wird, falls kein passender Behandler vorhanden ist. Dieser Behandler wird bereits bei der Initialisierung angelegt.

7.2.2 isrHandlerContainer

Der isrHandlerContainer repräsentiert die Liste der Behandler. Dazu kennt der Container das erste LinkedListElement der Liste und über dieses auch die weiteren. Um zu ermitteln, ob die Liste leer ist, steht ein `isEmpty` Attribut zur Verfügung. Mit der Methode `insert()` können neue Handler der Liste hinzugefügt werden und mit remove() entsprechend auch wieder entfernt werden. Die Methode `handle()` gibt einen Aufruf an seine Behandler weiter.

Die Klasse `linkedListElement` stellt die einzelnen Elemente der Liste, die sich hinter jeder Vektornummer verbirgt, dar. Mit der Funktionen dieser Klasse können Operationen auf den einzelnen Elementen durchgeführt werden. Die zentralen Funktionen der Klasse sind `hasNext` und `next`. Mit der Funktion kann überprüft werden, ob noch ein weiteres Element, aus der Sicht des aktuellen Elements, vorhanden ist. Sollte noch ein nächstes Element vorhanden sein, so kann dieses mit der Funktion `next` ermittelt werden. Des Weiteren gibt es die Funktionen `append` und `removeNext` mit der Funktion `append` wird die bereits existierende Liste mit Behandlern zu einer Vektornummer hinter das hinzuzufügende Element eingefügt. Die Funktion `removeNext` löscht den direkt benachbarten Behandler aus der Liste. Bei der Initialisierung wird das Attribut `attHasNext` mit `false` initialisiert und dieser wird erst dann auf `true` gesetzt, wenn ein weiteres benachbartes Element eingefügt wird.

7.2.3 isrHandler

Die Klasse `isrHandler` stellt eine abstrakte Klasse mit der Methode `handle()` dar. Jeder Interrupthandler, der vom Anwender implementiert werden soll, muss von dieser Klasse abgeleitet werden. In dieser abgeleiteten Klasse kann dann die für den Interrupt zutreffende Implementierung vorgenommen werden. Wie im Klassendiagramm veranschaulicht, wird auch die Klasse `defaultHandler` von der Klasse `isrHandler` abgeleitet.

7.3 Besonderheiten der Implementierung

Zur Verwaltung der Interrupt-Behandler zu einer Vektornummer wird eine Implementierung einer verketteten Liste verwendet. Die Struktur einer solchen Liste ist in 7.3 dargestellt. Ein Element von dieser Liste kennt nicht nur seinen eigenen Inhalt, sondern ihm ist auch das folgende Element in dieser Liste bekannt. Somit gibt es eine Verbindung von jedem Element der Liste zu seinem nachfolgenden Element, falls eines vorhanden ist.

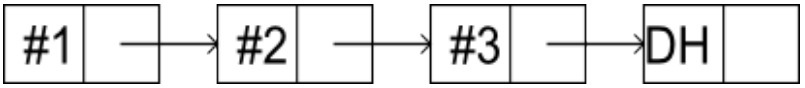


Abbildung 7.3: Implementierung der LinkedList

Teil III

Task-Verwaltung und Synchronisation

8 Synchronisation

8.1 Grundlagen

Durch eine Synchronisation mehrerer nebenläufiger Threads soll der Zugriff auf gemeinsam genutzte Ressourcen geregelt werden, um Konflikte zu vermeiden. So sollen z. B. Inkonsistenzen auf gemeinsam genutzten Speicher verhindert werden. Das zu lösende Problem ist ein *kritischer Wettlauf* zwischen verschiedenen Threads.

8.1.1 Kritischer Wettlauf (Race Condition)

Ein *kritischer Wettlauf* zwischen zwei Threads entsteht, wenn das Ergebnis der Operationen, die die Threads ausführen, abhängig von der genauen Ausführungsreihenfolge der einzelnen Teiloperationen der Threads ist.

Konkret bedeutet das z. B., dass zwei Prozesse auf gemeinsam genutzten Speicher zugreifen und ein Thread den Speicher des anderen überschreibt. Dies kann dadurch geschehen, dass beide Threads ihre Ergebnisse in eine Ergebnisliste im gemeinsam genutzten Speicher schreiben wollen. Ein fehlerhafter Ablauf sieht dann so aus:

Thread1	**Thread2**
sucht das letzte Listenelement der Ergebnisliste	
wird unterbrochen	
	sucht das letzte Listenelement der Ergebnisliste
	speichert sein Ergebnis als Nachfolgeeintrag des Listenelements
	wird unterbrochen
speichert sein Ergebnis als Nachfolgeeintrag des Listenelements	

Nach diesem Ablauf ist das Ergebnis von *Thread2* verloren gegangen.

8.1.2 Kritischer Abschnitt (Critical Region)

Um das Problem des *kritischen Wettlaufs* zu lösen, werden *kritische Abschnitte* eingeführt in denen Threads exklusiv auf die Betriebsmittel zugreifen können. Das bedeutet, dass die Betriebsmittel nicht parallel zu Programmabschnitten anderer Threads ausgeführt werden dürfen, in denen die gleichen Betriebsmittel ebenfalls verändert werden würden. Dieses Verfahren nennt man *Wechselseitiger Ausschluss* (engl. mutual exclusion, oder kurz Mutex).

8.2 Verfahren zum wechselseitigen Ausschluss

Um in der IA-32-Architektur einen *wechselseitigen Ausschluss* zu realisieren, stehen verschiedene Möglichkeiten mit unterschiedlichen Vor- und Nachteilen zur Verfügung.

8.2.1 Abschalten der Interrupts

Die einfachste Lösung besteht darin, die Interrupts im System auszuschalten. Dadurch wird in einem Einprozessorsystem gewährleistet, dass auf ein Betriebsmittel nur von einem Thread gleichzeitig zugegriffen werden kann. Denn ein Wechsel zwischen Threads findet erst statt, wenn der aktuell ausgeführte Thread, seine Ausführung unterbricht. Ein Scheduler, der zwischenzeitlich den Thread unterbrechen könnte, wird durch das Ausschalten der Interrupts übergangen, da er meist durch Interrupts eines Zeitgebers angesteuert wird.

Das Ausschalten der Interrupts hat jedoch einige Nachteile.

1. Die Latenz der Interruptbearbeitung verlängert sich, da die Interrupts zwischengespeichert werden und erst nach dem Einschalten wieder ausgeführt werden. Dies verlangsamt das gesamte System.

2. Das Betriebssystem kann nicht sicher auf Mehrprozessorsystemen ausgeführt werden, da selbst mit abgeschalteten Interrupts, von mehreren Threads auf ein Betriebsmittel zugegriffen werden kann. Dies ist dadurch bedingt, dass die Threads auf unterschiedlichen Prozessoren gleichzeitig ausgeführt werden.

3. Die Sicherheit des Systems leidet, da die Stabilität selbst eines mit *präemptivem Multitasking*[38] arbeitenden Betriebssystems abhängig von der

[38] Beim präemptiven Multitasking steuert der Betriebssystemkern die Abarbeitung der einzelnen Threads, wobei jeder Thread nach einer bestimmten Abarbeitungszeit zu Gunsten eines anderen

Funktionstüchtigkeit der einzelnen Threads wird. Denn sollte ein Thread die Interrupts ausschalten und dann in eine Endlosschleife geraten, kann der Scheduler nicht mehr eingreifen und das System wird funktionsunfähig.

8.2.2 Aktives Warten

Beim *aktiven Warten*, prüft ein Thread solange eine Bedingung in einer Schleife bis sie erfüllt ist. Damit wird sichergestellt, dass er den kritischen Abschnitt nur betritt, wenn die Bedingung es erlaubt. Der Nachteil der Methode ist, dass das aktive Warten die CPU unnötig in Anspruch nimmt und ein kooperatives Multitasking verhindert, da der Thread sich nicht unterbricht. Dafür muss dann ein Scheduler sorgen.

Es gibt einige Verfahren, die das *aktive Warten* einsetzen um einen *wechselseitigen Ausschluss* zu garantieren.

Strenge Alternation

Bei der *strengen Alternation* laufen die Threads streng nacheinander ab und wechseln sich mit dem Zugriff auf den kritischen Bereich ab. Diese Situation ist in Quelltext 8.1[39] zu sehen.

Listing 8.1: Beispiel strenge Alternation

```
/* Task 1 */
while (TRUE) {
    while (turn != 0) /* loop */ ;
    critical_region();
    turn = 1;
    noncritical_region();
}

/* Task 2 */
while (TRUE) {
    while (turn != 1) /* loop */ ;
    critical_region();
    turn = 0;
    noncritical_region();
}
```

Threads unterbrochen wird.

[39] Quelltext entstammt [4]

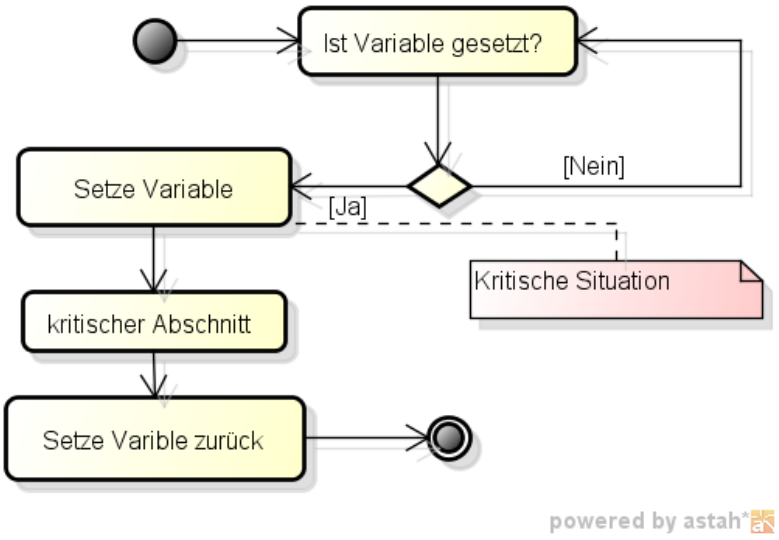


Abbildung 8.1: Ablauf des Lock-Verfahrens mit aktivem Warten

Mit der Variable **turn** wird zwischen den Tasks hin und her geschaltet. Der Nachteil ist, dass die Abläufe nur alternierend die kritischen Abschnitte betreten können und dass ein Task, der nicht im kritischen Abschnitt arbeitet, trotzdem warten muss bis der andere diesen wieder verlassen hat.

Lock

Ein weiteres Verfahren für den *wechselseitigen Ausschluss* ist das Verwenden eines Locks (Sperre). Dabei wird geprüft, ob eine Variable gesetzt ist. Der Ablauf ist in Abbildung 8.1 zu sehen. Da der kritische Abschnitt erst betreten wird, wenn die Lock-Variable nicht gesetzt ist, wird verhindert das der *kritische* Abschnitt von mehreren Threads betreten wird. Ist die Variable gesetzt, wartet der Thread aktiv, bis die Variable wieder frei wird. Das Problem des konkurrierenden Zugriffs auf ein Betriebsmittel hat sich dadurch auf den konkurrierenden Zugriff auf eine Lock-Variable verlagert. Das Problem ist dadurch jedoch noch nicht endgültig gelöst, da bei zwei nahezu zeitgleichen Anfragen auf den Status der Variablen, beide als Ergebnis frei erhalten, da die Variable noch nicht wieder gesetzt worden ist. Diese kritische Situation ist ebenfalls in der Abbildung dargestellt.

Um diese kritische Situation zu vermeiden gibt es softwareseitige Lösungen die ohne zusätzliche Prozessorbefehle auskommen wie der Dekker[40]- und der Peterson[41]-Algorithmus. Die IA-32-Architektur bietet jedoch auch Befehle, um das Prüfen-und-Setzen (Test-And-Set) atomar durchzuführen.

[40] Nähere Informationen in [2]
[41] Nähere Informationen in [3]

8.2.3 Passives Warten

Beim *passiven Warten* wird die Ausführung eines Threads unterbrochen, sobald er auf eine Ressource warten muss. Wird diese Ressource wieder freigegeben, kann der Thread erneut versuchen in den kritischen Bereich zu gelangen, sofern er vom Scheduler ausgewählt wurde.

Dieses Verhalten lässt sich mit dem Konzept der **Semaphore** realisieren, welches von Edsger Dijkstra[42] entwickelt wurde. Eine Semaphore ist eine Art Zähler mit den Operationen *up* und *down*. Dabei gibt der Wert des Zählers an, wie viele Threads noch gleichzeitig auf das – durch die Semaphore geschützte – Betriebsmittel zugreifen können. Bei einem Zählerstand von 0 ist das Betriebsmittel voll ausgelastet. Die Threads, die ein voll ausgelastetes Betriebsmittel anfordern (*down*), werden in eine Liste der Semaphore aufgenommen und auf den Thread-Zustand „Blockiert“ gesetzt. Das bedeutet, dass der Scheduler die Ausführung des Threads unterbricht. Sobald das Betriebsmittel wieder von einem Thread freigegeben wird (*up*), wird der Zähler der Semaphore inkrementiert und die wartenden Threads werden auf *bereit* gesetzt. Danach kann der Scheduler wieder einen dieser Threads starten.

8.3 Synchronisationsprimitive

Um Synchronisationsoperationen zu implementieren, kennt man in der Informatik die Methoden *Compare-and-swap*, *Test-and-set* und *Fetch-and-add*. Sie werden im Folgenden näher betrachtet.

8.3.1 Compare-and-swap

Eine Compare-and-swap-Operation hat drei Parameter: die Speicherstelle, die geprüft werden soll, der alte Wert, der erwartet wird und der neue Wert, der gesetzt werden soll. Der Wert der Speicherstelle wird nur geändert, falls die Speicherstelle zum Zeitpunkt des Aufrufes den erwarteten Wert hat. Als Rückgabe erhält man die Information, ob der Wert geändert wurde. In C++ würde compare-and-swap wie in Listing 8.2 implementiert werden, wobei C++ die Atomarität nicht sicherstellen kann.

Listing 8.2: Compare-and-swap in C++

```
1 // Atomare Operation!
```

[42] Nähere Informationen in [2]

```
2  bool compareAndSwap(unsigned *p, unsigned alt, unsigned neu) {
3    if(*p == alt) {
4      *p = neu;
5      return true;
6    }
7    else {
8      return false;
9    }
10 }
```

Mit Hilfe dieser Operation ist es also möglich, den Wert vorher auszulesen, entsprechend zu verändern (z. B. inkrementieren/dekrementieren) und nur dann zurückzuschreiben, falls sich der Wert seit dem Auslesen nicht verändert hat. Listing 8.3 zeigt eine einfachere Anwendung für die Implementierung eines kritischen Bereichs.

Listing 8.3: Compare-and-swap für kritische Bereiche

```
1  #define UNLOCKED 0
2  #define LOCKED 1
3
4  while(!compareAndSwap(mutex, UNLOCKED, LOCKED)) {
5    /* aktives Warten */
6  }
7
8  /* kritischer Bereich */
9
10 *mutex = UNLOCKED;
```

In der IA-32-Architektur gibt es den Befehl `CMPXCHG` (Compare and Exchange), welcher genau dieses Verfahren umsetzt. Er ist seit dem Intel 80486 implementiert. `CMPXCHG` erwartet den Vergleichswert (den alten Wert) im Register EAX (bzw. AX für 16 Bit, AL für 8 Bit). Der Befehl wird dann mit zwei Operanden aufgerufen: der erste Operand (destination operand) enthält die Speicherstelle, welche verändert werden soll, der zweite Operand (source operand) enthält den zu schreibenden Wert. Wurde der Wert geschrieben, wird das Zero-Flag (ZF) gesetzt. Wurde er nicht geschrieben, wird der Wert des ersten Operanden in das EAX-Register (bzw. AX, AL) geschrieben und das Zero-Flag gelöscht. Dem Befehl muss der Präfix `LOCK` vorangestellt werden, damit er atomar ausgeführt wird. Er kann nicht durch Interrupts unterbrochen werden.

8.3.2 Test-and-set

Eine Test-and-set-Operation hat nur ein einziges Argument: die Speicherstelle, die überprüft werden soll. Die Speicherstelle wird bei jedem Aufruf auf Nicht-Null (also bei einem bool z. B. true) gesetzt und der Wert der Speicherstelle vor dem Setzen wird zurückgegeben. Listing 8.4 zeigt die Implementierung in C++.

Listing 8.4: Test-and-set in C++

```
// Atomare Operation!
bool testAndSet(bool *p) {
  bool alterWert = *p;
  *p = true;
  return alterWert;
}
```

Test-and-set wird in der Regel für kritische Bereiche genutzt. Der Zustandswechsel der Sperrvariable, wenn zwei Threads auf einen kritischen Bereich zugreifen wollen, ist in der folgenden Tabelle beispielhaft dargestellt.

Sperrvariable	Thread 1	Thread 2
false		
true	testAndSet - Rückgabe false kritischer Bereich betreten	
true		testAndSet - Rückgabe true wartet
true		testAndSet - Rückgabe true wartet
false	Sperrvariable := false kritischer Bereich verlassen	
true		testAndSet - Rückgabe false kritischer Bereich betreten

Die Umsetzung eines kritischen Bereichs in C++ wird in Listing 8.5 gezeigt.

Listing 8.5: Test-and-set für kritische Bereiche

```
while(testAndSet(lock)) {
  /* aktives Warten */
}

/* kritischer Bereich */

*lock = false;
```

Der korrespondierende Befehl der IA-32-Architektur heißt `BTS` (Bit Test and Set). Dieser Befehl erwartet zwei Operanden: der erste Operand gibt ein Register oder eine Speicheradresse an, in der sich das zu überprüfende Bit befindet, der zweite Operand gibt den Offset des Bits innerhalb des Registers, bzw. eine Speicherstelle, in der sich der Offset befindet, an. Der Wert des so adressierten Bits wird in das Carry-Flag (CF) geschrieben und anschließend auf 1 gesetzt. Dem Befehl muss der Präfix `LOCK` vorangestellt werden, damit er atomar ausgeführt wird. Er kann nicht durch Interrupts unterbrochen werden.

8.3.3 Fetch-and-add

Mit Hilfe der Fetch-and-add-Operation wird eine gegebene Speicherstelle inkrementiert und der vorherige Wert der Speicherstelle zurückgegeben. Listing 8.6 zeigt die Implementierung in C++.

Listing 8.6: Fetch-and-add in C++

```
1  // Atomare Operation!
2  unsigned fetchAndAdd(unsigned *p) {
3    return (*p)++;
4  }
```

Um mit Fetch-and-add einen kritischen Bereich zu definieren, kann man sich eine Aufrufanlage vorstellen. Man zieht sich mit Hilfe der Fetch-and-add-Operation ein Warteticket. Wird das eigene Ticket aufgerufen, darf man in den kritischen Bereich. Verlässt man den kritischen Bereich, wird die aktuelle Ticketnummer inkrementiert. Listing 8.7 zeigt eine beispielhafte Implementierung.

Listing 8.7: Fetch-and-add für kritische Bereiche

```
1  unsigned *wartenummer = new unsigned(0);
2  unsigned *aktuelleNummer = new unsigned(0);
3
4  void lock() {
5    unsigned meineWartenummer = fetchAndAdd(wartenummer);
6
7    while(meineWartenummer != *aktuelleNummer) {
8      /* aktives Warten */
9    }
10 }
11
12 void unlock() {
```

```
13    (*aktuelleNummer)++;
14 }
```

In der IA-32-Architektur gibt es keinen expliziten Befehl für Fetch-and-add. Es gibt jedoch den Befehl `XADD` (Exchange and Add), mit welchem diese Methode durchgeführt werden kann. Er ist seit dem Intel 80486 implementiert. `XADD` erwartet zwei Operanden: der erste Operand (destination operand) enthält die Speicheradresse, der zweite Operand (source operand) zeigt auf ein Register, in dem der zu addierende Wert (in unserem Fall 1) enthalten ist. Wird der Befehl ausgeführt, werden die Werte der Operanden getauscht und anschließend der destination operand auf die Summe beider Operanden gesetzt. Der source operand (ein Register) enthält anschließend den alten Wert. Der destination operand enthält den neuen Wert. Dem Befehl muss das Präfix `LOCK` vorangestellt werden, damit er atomar ausgeführt wird. Er kann nicht durch Interrupts unterbrochen werden.

8.4 Entwurf

In diesem Abschnitt wird näher auf den Entwurf eingegangen der eine Synchronisation zwischen unterschiedlichen Threads ermöglichen soll. Der Entwurf ist dabei in die Klassen *Semaphore* und *Mutex* aufgeteilt. In Abbildung 8.2 sind die beiden Klassen mit den dazugehörigen Verweisen abgebildet. Zunächst werden aber die Schnittstellen, die die beiden Klassen zur Verfügung stellen und die benötigten Abhängigkeiten zu anderen Komponenten vorgestellt.

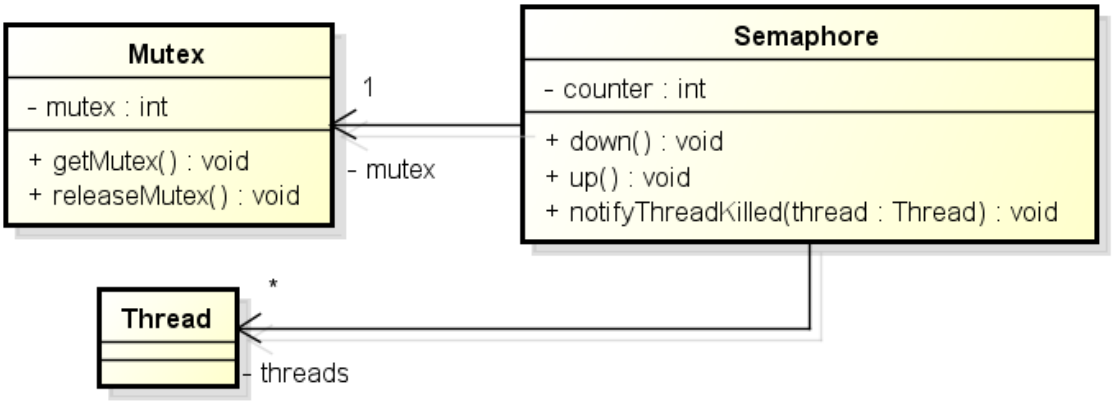


Abbildung 8.2: Klassendiagramm

8.4.1 Abhängigkeiten

Zur Realisierung der Semaphore und des Mutexes wird der Scheduler benötigt. Vom Scheduler werden drei Funktionen benötigt:

- Ermittlung des aktuellen Threads, um diesen in die Liste der wartenden Threads aufzunehmen.
- Ändern des Zustandes eines Threads. Diese Funktion wird benötigt damit die Semaphore die wartenden Threads wieder in einen ausführbaren Zustand versetzen kann.
- Einen Thread in wartenden Zustand versetzen und den zu einer Semaphore zugehörigen Mutex freigeben, wobei beide Aktionen atomar durchgeführt werden müssen. Wird benötigt wenn ein Thread an der Semaphore warten muss.

8.4.2 Mutex

Schnittstelle

Ein Mutex erlaubt die Absicherung eines kritischen Bereiches. Der kritische Bereich, der durch den Mutex geschützt wird, kann immer nur von einem Thread betreten werden. Falls sich im kritische Bereich schon ein Thread befindet, warten alle anderen Threads aktiv solang, bis der kritische Bereich wieder frei ist. Hierbei wird die Reihenfolge der Anforderung des kritischen Bereiches nicht berücksichtigt.

acquireMutex Muss aufgerufen werden, wenn der kritische Bereich betreten wird. Falls sich schon ein Thread im kritischen Bereich befindet, wird solange aktiv gewartet bis der kritische Bereich wieder frei ist.

releaseMutex Muss aufgerufen werden, wenn der kritische Bereich verlassen wird. Damit wird der kritische Bereich wieder als frei markiert und der nächste Thread kann den kritischen Bereich betreten.

Entwurf

Die Klasse Mutex ermöglicht ein aktives Warten auf einen kritischen Bereich[43] wie in Abschnitt 8.2.2 beschrieben. Um das Problem des konkurrierenden Zugriffs auf

[43] Im weiteren Verlauf als Mutex bezeichnet

die Lock-Variable *mutex* zu beheben, wird die Test-and-Set Methode[44] genutzt, da diese den Anforderungen genügt und am einfachsten zu implementieren war. Mithilfe der acquireMutex Operation wird der Mutex angefordert. Dieser Vorgang ist in Abbildung 8.3 dargestellt. Das Überprüfen und Setzen der Lock-Variable geschieht im Bereich, der mit „atomar“ gekennzeichnet ist. Da derzeit ein kooperatives Multitasking durch den Scheduler implementiert wird, gibt der Mutex an den Scheduler die Kontrolle ab, da sonst kein Threadwechsel mehr Aufgrund des aktiven Wartens im Mutex stattfinden würde. Um den Mutex wieder

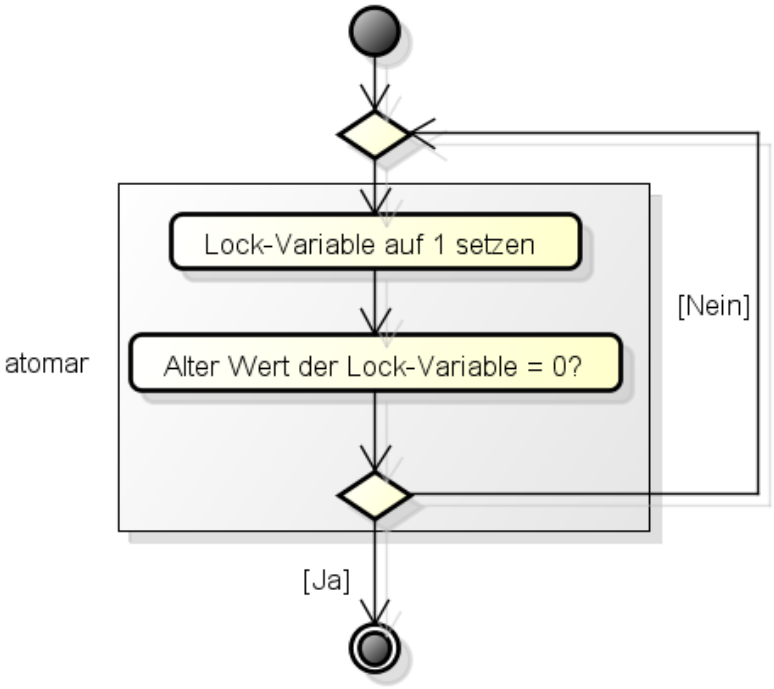


Abbildung 8.3: Anfordern des Mutexes

freizugeben wird die Operation releaseMutex genutzt. Hierbei wird die Lock-Variable auf den Zustand gesetzt, der signalisiert, dass der Mutex wieder frei ist.

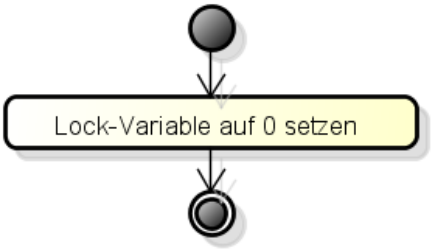


Abbildung 8.4: Mutex freigeben

[44] siehe Abschnitt 8.3.2

8.4.3 Semaphore

Schnittstelle

Eine Semaphore ermöglicht einen konkurrierenden Zugriff von mehreren Threads auf einen kritischen Bereich/Ressource. Es ist möglich, einer bestimmte Anzahl an Threads Zugriff auf den kritischen Bereich/die Ressource zu gewähren. Bei einem ausgeschöpften Kontingent warten die Threads passiv bis wieder eine Ressource zur Verfügung steht. Nachfolgend sind die Operationen beschrieben, die zum Steuern einer Semaphore zur Verfügung stehen.

down Fordert eine Ressource von der Semaphore, die den kritischen Bereich schützt, an. Falls keine Ressource mehr zur Verfügung steht, wird der aufrufende Thread in einen Wartezustand versetzt.

up Gibt eine angeforderte Ressource wieder zurück, so dass die Threads, die auf die Semaphore warten, wieder eine Ressource anfordern können.

notifyThreadKilled Mit dieser Operation wird der Semaphore mitgeteilt, dass der als Parameter übergebene Thread gestoppt wurde. Das hat zur Folge, dass der übergebene Thread nicht mehr berücksichtigt wird, falls wieder eine Ressource vorhanden sein sollte. Es ist nur nötig der Semaphore mitzuteilen, dass ein Thread gestoppt wurde, falls er auf die Semaphore wartet.

Entwurf

Die Klasse übernimmt die Funktion einer Semaphore wie sie in Abschnitt 8.2.3 beschrieben wird. Da die Semaphore intern auch einige kritische Bereiche aufweist, wird innerhalb der Semaphore ein Mutex[45] genutzt, um diese kritischen Bereiche abzusichern. In den kritischen Bereichen findet ein Zugriff auf die gemeinsamen Ressourcen statt, wie zum Beispiel der interne Zähler einer Semaphore und die Liste der Threads die auf die Semaphore warten.

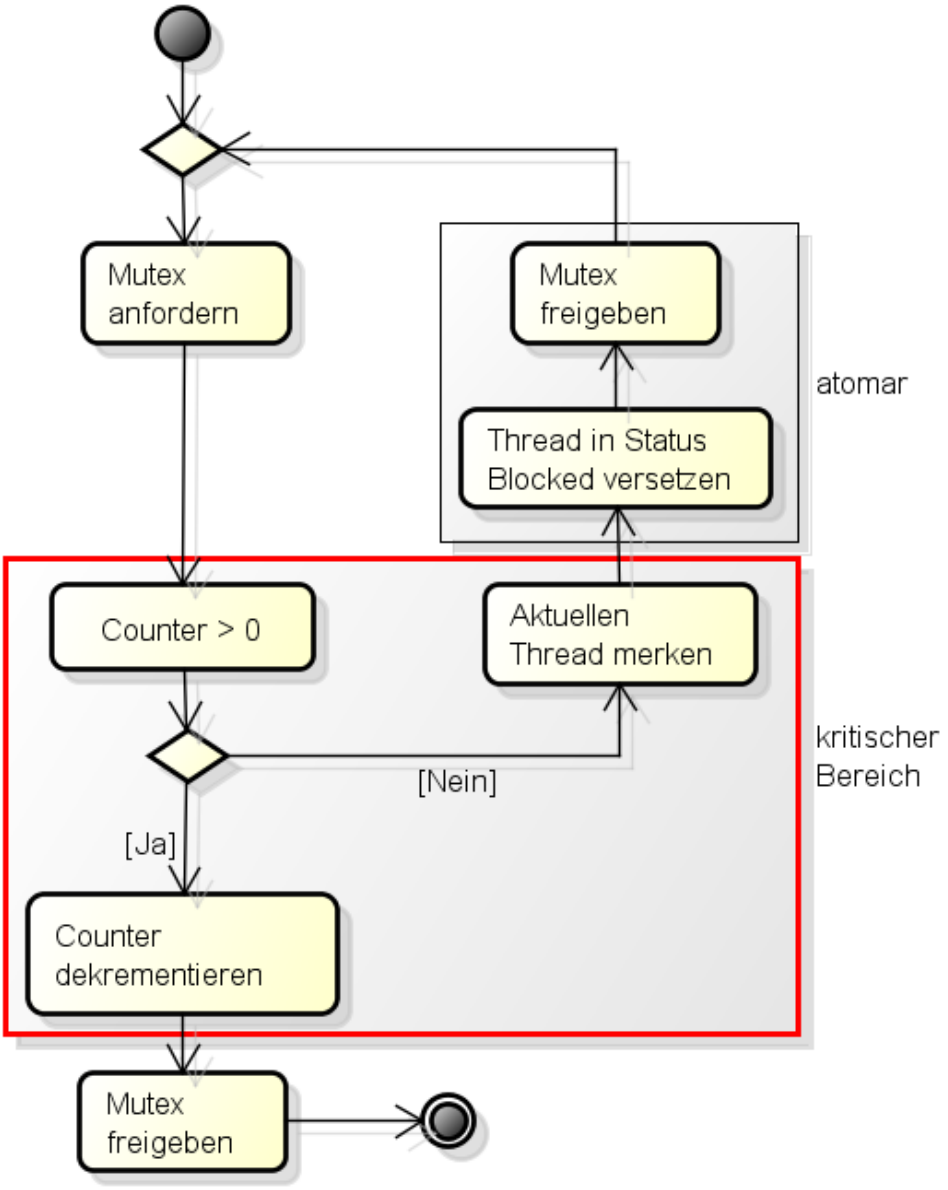


Abbildung 8.5: Methode down der Semaphore

down

Der Ablauf der down Methode ist in Abbildung 8.5 dargestellt. Der kritische Bereich ist mit einem roten Kasten gekennzeichnet. Er enthält die Operationen zum Überprüfen und Setzen des Zählers und zum Merken des aktuellen Threads. Weiterhin ist zu erwähnen, dass, wenn der aktuelle Thread aufgrund des Zählers an der Semaphore warten muss[46], das Ändern des Zustandes des aktuellen Threads und das Freigeben des internen Mutex atomar (also explizit ohne Task-Wechsel-Unterbrechungen) durchgeführt werden muss. Ansonsten könnten inkonsistente Zustände in der Semaphore auftreten. Dies wird dadurch gewährleistet, dass diese beiden Aktionen durch Aufruf einer Operationen des Scheduler durchgeführt werden, der dann intern die Atomarität dieser beiden Aktionen sicherstellt.

up

Der Ablauf der Implementierung der up-Methode ist in Abbildung 8.6 dargestellt. Der kritische Bereich ist wieder mit einem roten Kasten markiert. In diesem wird der Zähler inkrementiert und alle Threads, die auf diese Semaphore warten, wieder in Zustand RUNNABLE versetzt. Dies hat zur Folge, dass keine feste Reihenfolge existiert, in welcher die wartenden Threads eine Freigabe von der Semaphore erhalten.

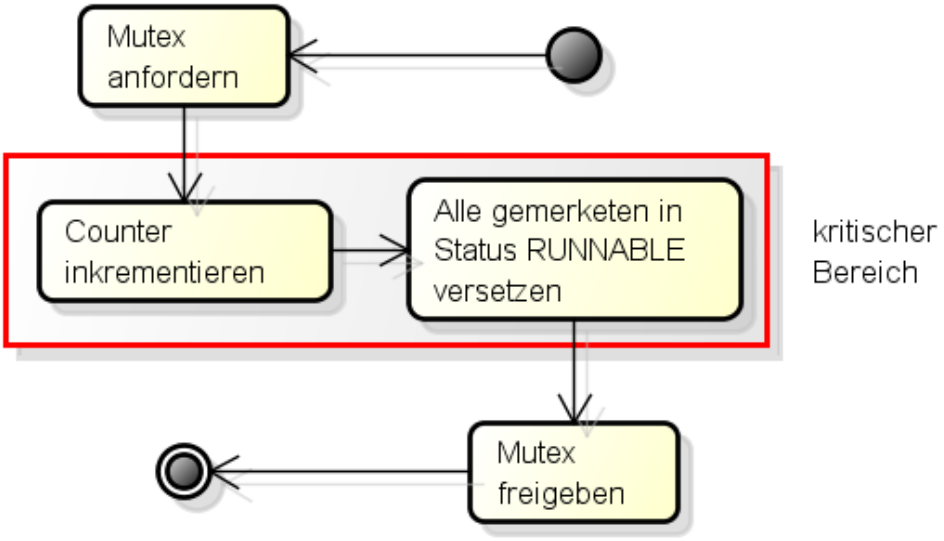


Abbildung 8.6: Methode up der Semaphore

[45] Siehe Abschnitt 8.4.2
[46] Abbildung 8.5 Kasten atomar

9 Task-Verwaltung

9.1 Analyse

9.1.1 Task State Segment

Beim Task-Wechsel möchte man den unterbrochenen Task zu einem späteren Zeitpunkt in dem Zustand[47] weiter ausführen, in dem er unterbrochen wurde. In der Intel IA32-Architektur wird dies mit Hilfe des Task State Segments (TSS) erreicht. Über das TSS können diese Werte durch den Prozessor abgerufen werden. Die Abbildung 9.1 zeigt den Aufbau des Task State Segments.

Für das Task State Segment wird ein Deskriptor in der GDT abgelegt, dieser kann über einen Selektor des Deskriptors vom Prozessor geladen werden. Der Nachteil dieser Realisierung ist, das maximal 8189 Tasks abgelegt werden können, denn die GDT kann nicht mehr Einträge enthalten. Genauere Informationen zur GDT befinden sich in Abschnitt 9.1.7.

Bei einem Task-Wechsel aktualisiert der Prozessor die dynamischen Werte, wie z. B. den Stack- oder Instruction-Pointer. Die Voraussetzung dafür ist, dass das Task-Register auf ein Task-State-Segment zeigt. Daher muss für den Initialprozess auch ein TSS erstellt werden und einmalig das Task-Register auf dieses geladen werden, damit zu anderen Prozessen gewechselt werden kann.

Das Task-State-Segment bietet zusätzlich Schutz für IO, dieser Mechanismus wird jedoch bislang nicht berücksichtigt.

9.1.2 Task-Register

Das Task-Register enthält einen 16-Bit Segmentselektor und den kompletten Segmentdeskriptor für die TSS des aktuellen Tasks. Diese Informationen sind aus

[47] Ein Zustand besteht aus Stack- und Instruction Pointer, Registerzuständen und Segmentselektoren

31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0

Reserved bits. Set to 0.

Abbildung 9.1: Task-State-Segment

dem TSS Deskriptor des aktuellen Tasks kopiert. Es hat einen sichtbaren Teil, der durch Software gelesen und geändert werden kann und einen unsichtbaren Teil, der vom Prozessor verwaltet wird.

Die Befehle LTR (load task register) und STR (store task register) laden und lesen den sichtbaren Teil des Task-Registers:
Die LTR Anweisung lädt einen Segmentselektor in das Task-Register der auf einen TSS Deskriptor in der GDT zeigt. Danach wird der unsichtbare Teil des Task-Registers mit Informationen des TSS Deskriptor befüllt. LTR ist eine privilegierte Anweisung die nur auf CPL 0 ausgeführt werden kann. Sie wird während der Systeminitialisierung benutzt, um einen Initialwert in das Task-Register zu speichern. Später ändert sich der Inhalt des Task-Registers implizit wenn ein Taskwechsel stattfindet.

Die STR Anweisung speichert den sichtbaren Teil des Task-Registers in einem Mehrzweckregister oder Speicher. Diese Anweisung kann von jedem Privileglevel ausgeführt werden, um den aktuell laufenden Task zu identifizieren.

9.1.3 Task-Gates

Ein Taskwechsel kann mithilfe eines Task-Gate vorgenommen werden. Es kann in der GDT, der LDT, oder IDT liegen. Dabei zeigt der TSS-Segmentselektor im Task-Gate auf denjenigen Segmentdeskriptor, der das TSS des Tasks definiert, zu dem gewechselt wird. Wenn durch einen Call, Jump oder Interrupt ein Task-Gate aufgerufen wird, so wird ein Task-Wechsel initiiert und der Zustand des bisher aktiven Tasks in dem TSS abgespeichert, das durch das Task-Register (TR) definiert ist.

9.1.4 Thread-Zustände

Blockiert Der Thread verweilt in einem Wartezustand und muss auf die Erfüllung (mindestens) einer Wartebedingung (z. B. Semaphore) warten. Sobald alle Bedingungen für den Thread erfüllt wurden, geht dieser in den Zustand Bereit über und erwartet die Prozessorzuteilung.

Bereit Die Threads, die in der Warteschlange stehen, werden als bereit bezeichnet. Nur einem Thread, für den keine Wartebedinungen vorliegen, kann die CPU zugeteilt werden.

Aktiv Durch Zuteilung der CPU wird der Thread in den Status „Aktiv“ versetzt. In diesem Zustand bekommt der Thread exklusiv Rechenzeit

der CPU zugeteilt. Welcher rechenwillige Thread zur Ausführung kommt, hängt vom Scheduler ab. Der Zustand kann sich ändern, wenn ein anderer Thread zur Ausführung gelangt und dem aktuellen Thread den Prozessor entzieht. Darüber hinaus können Wartebedingungen (z. B. Warten auf Eingabedaten) entstehen, die den Thread in den Zustand „Blockiert“ wechseln lassen. Ein Prozessor kann zu einem gegebenen Zeitpunkt immer nur einen Thread bearbeiten.

Die Übergänge zwischen den Zuständen zeigt Abbildung 9.2.

Abbildung 9.2: Threadzustände

9.1.5 Prozess vs. Thread

Prozesse sind die Abfolge von bestimmten Ereignissen und lösungsorientiert. Als Spezialfall: „in Betriebssystemen ist ein Prozess ein Vorgang, der durch ein Programm kontrolliert wird, welches zur Ausführung einen Prozessor benötigt“ (Duden Informatik).
Threads sind leichtgewichtigere Prozesse und erweitern das Prozessmodell. Threads bewirken, dass Teile innerhalb eines Prozesses quasi nebenläufig ausgeführt werden. Sofern das Betriebssystem dies unterstützt und die Betriebsmittel vorhanden sind, können Threads durch mehrere CPU-Kerne echt parallel ausgeführt werden. Threads die (echt) nebenläufig ausgeführt werden, teilen sich mit anderen Threads die Betriebsmittel des Rechners.

9.1.6 Einordnung von Prozessen, Threads und Tasks

In unserem Projekt unterscheiden wir zwischen Prozessen, Threads und Tasks. Nachdem die Begriffe Prozess und Thread bereits grundlegend definiert wurden, soll nun eine auf dieses Projekt bezogene Abgrenzung statt finden.

Prozesse und Threads sind Begriffe, die wir auf der Objektebene (High-Level) benutzen. Hierbei gilt, dass ein Prozess mehrere Threads beinhalten kann und ein Thread immer zu einem Prozess gehören muss. Der Begriff Task bezieht sich auf

die Ausführung innerhalb der Hardware, konkret des Prozessors (Low-Level). Der Prozessor kennt weder Prozesse noch Threads. Für ihn gibt es nur Tasks, zwischen denen er wechseln kann. Diese Tasks können untereinander kommunizieren, wenn sie über einen gemeinsamen Speicherbereich verfügen, ähnlich wie es Threads eines Prozesses untereinander möglich ist. Daher entspricht in diesem Projekt ein Prozess im ”High-Level“ in etwa einem PageContext im ”Low-Level“. Ein Thread im ”High-Level“ entspricht so einem Task im ”Low-Level“.

9.1.7 Global Descriptor Table

Da die Segmentregister mit 16 Bit etwas wenig Platz für die notwendigen Informationen anbieten, erhalten diese nur noch Selektoren – also die Nummer eines Segments, das auf eine Tabelle verweist und zusätzliche Flags. Die wirklichen Beschreibungen dieser Segmente sind in der Global Descriptor Table (GDT) zu finden. Diese besteht aus mehreren Segmentdeskriptoren zu je 64 Bit, die sich direkt hintereinander in dieser Tabelle befinden. Tabelle 9.1 beschreibt einen Verweis auf die GDT.

Bits	Wert	Bedeutung
0 - 1	0x3	RPL (Requested Privilege Level). Gibt die Privilegstufe (den Ring) an mit der versucht werden soll das Segment anzusprechen, falls diese numerisch kleiner ist als die aktuelle des aufrufenden Tasks so behält dieser seine aktuelle
2	0x4	Bestimmt, ob ein Eintrag in der GDT (Bit nicht gesetzt) oder in der LDT (Bit gesetzt) ausgewählt wird
3 - 15	0xfff8	Nummer des Eintrags in der GDT bzw. LDT (ab Null gezählt)

Tabelle 9.1: Verweis auf die GDT

Die Global Descriptor Table ist eine Tabelle, um den Speicher, das Multitasking und verschiedene Gates zu verwalten. Sie beinhaltet verschiedene Informationen über Speicherabschnitte. In dieser Tabelle können Segmente eingetragen werden, die zum Beispiel den Arbeitsspeicher in einen bestimmten Bereich adressieren und schützen. Die Struktur der Tabelleninhalte wird im folgenden Abschnitt beschrieben.

Struktur GDT

Jeder Eintrag der GDT besteht aus 8 Byte und hat die in Tabelle 9.2 dargestellte Struktur.

Byte	Name	Bits
0	Limit	0-7
1	Limit	8-15
2	Base	0-7
3	Base	8-15
4	Base	16-23
5	Accessbyte	0-7 (vollständig)
6	Limit	16-19
6	Flags	0-3 (vollständig)
7	Base	24-31

Tabelle 9.2: Struktur GDT-Eintrag

- Limit: Größe des Segments - 1 (entweder in Bytes oder in 4 KiB-Einheiten siehe Flags)
- Base: Die Adresse wo das Segment beginnt
- Accessbyte: Zugriffsinformationen (Ring, executable, etc.) (Vollständig in Byte 5)
- Flags: Definiert die Segmentgrößeneinheit und 16/32?Bit. (Vollständig in Byte 6)

Initialisierung der GDT

Die GDT muss immer mindestens einen gültigen Deskriptor enthalten. Da dies schwierig zu gewährleisten ist, ist der erste Eintrag der GDT der sogenannte Nulldeskriptor. Dieser kann auf 0 gesetzt bzw. gelassen werden. Durch diese Konvention ist es möglich eine leere GDT zu erkennen und entsprechend zu behandeln. Ab dem folgenden Eintrag beginnen die tatsächlichen Deskriptoren. Zusätzlich zum Nulldeskriptor gibt es noch zwei weitere Deskriptoren, die vorhanden sein müssen: das Code- und Datensegment für den Kernel. Diese drei Standardeinträge sollte immer in der GDT vorhanden sein.

Laden der Global Descriptor Table

Wenn sich im Speicher des Kernels eine GDT befindet, mit der er auf seine Segmente verweisen möchte, muss der Kernel noch das Register der GDT (GDTR)

ändern. In diesem Register befinden sich zwei wichtige Informationen, die Adresse und das Limit der GDT. Dieses Register wird mit dem Befehl `lgdt`geladen. Da dieses Register nur 6 Byte groß ist, kann dieser Befehl den neuen Wert nicht direkt als Operand entgegennehmen, sondern erwartet einen Zeiger auf eine Speicherstelle, die diese 6 Bytes enthält. Anschließend sollten alle Segmentregister neu geladen werden.

9.1.8 Dispatcher

Der Dispatcher ist im Rahmen der Prozessverwaltung eines Betriebssystem tätig und dient dazu bei einem Kontextwechsel dem aktiven Thread die Ressourcen (CPU) zu entziehen und diese dem darauf folgenden Thread zuzuordnen. Es wird dem rechnenden Thread der Rechenkern entzogen und dieser dem nächsten Thread zugewiesen. Die Entscheidung, welcher Thread der Nachfolger ist, wird vom Scheduler getroffen. Scheduler und Dispatcher arbeiten Hand in Hand.

9.2 Entwurf

Das für diesen Entwurf angefertigte Klassendiagramm befindet sich im Anhang. Im folgenden werden einige ausgewählte Aspekte näher beschrieben.

9.2.1 Klasse „GDTManager“

Zur Verwaltung der GDT ist die Klasse GDTManager zuständig. Diese Klasse bietet alle Funktionalitäten, die für die Verwendung der GDT notwendig sind. Diese Klasse ist nach dem Entwurfsmuster Singleton umgesetzt und daher gibt es von ihr im gesamten Betriebssystem nur eine Instanz. Der Konstruktor dieser Klasse initialisiert die zuvor beschriebenen Pflichteinträge der GDT (Nulldeskriptor, Code/Datensegment für den Kernel). Die Operation `setGDTEntry()` ermöglicht es, weitere Einträge in der GDT in lesbarer Form vorzunehmen. Listing 9.1 zeigt, wie die Parameter an die richtige Position in der Tabelle zugewiesen werden.

Listing 9.1: Laden der GDT

```
void GDTManager::loadGDT() {
        puts("Loading GDT from Kernelspace: ");
        struct GdtPtr {
                uint16_t limit;
```

```
                void * base;
        }__attribute__((packed));
        GdtPtr gdtptr;
        gdtptr.limit = GDT_ENTRYS *
                sizeof(CodeOrDataOrTaskSegmentDescriptor) - 1;
        gdtptr.base = start;
        // load the gdt
        __asm("lgdt %0" : : "m" (gdtptr));
        // load all selectors except CS for activating GDT
        uint16_t a = 0x10;
        __asm ("mov %0, ax;" : : "r"(a) : );
        __asm("mov ds, ax");
        __asm("mov es, ax");
        __asm("mov fs, ax");
        __asm("mov gs, ax");
        __asm("mov ss, ax");

        puts("OK\r\n");
}
```

Zur Verwendung von Außen bietet der GDTManger bislang zwei Methoden an:

Selector addTSS(TaskStateSegment * tss); Diese Methode erstellt einen Deskriptor für die mit tss referenzierte TSS, legt ihn in der GDT ab und gibt den Selektor dafür zurück.

void free(Selector selector); Diese Methode setzt den GDT Eintrag der mit selector referenziert wird auf nicht Present. Damit kann dieser Eintrag für andere Deskriptoren verwendet werden.

9.2.2 Threads

Jeder Thread hat einen Zustand. Im Folgenden werden die möglichen Zustände beschrieben.

RUNNABLE sind Threads, wenn sie lauffähig sind.

BLOCKED ist ein Thread, wenn er auf eine Semaphore wartet.

WAITING ist der Startzustand eines Threads. In diesem Zustand kann ein Thread noch nicht ausgeführt werden. Er wartet auf seine Initialisierung.

Abbildung 9.3: Threadzustände

TERMINATED bedeutet, dass ein Thread fertig mit der Ausführung ist.

RUNNING ist ein impliziter Zustand. Der Thread, der im Scheduler als aktuell laufender Thread ausgezeichnet ist, hat diesen Zustand.

Gegenüber der Analyse wurde noch die Zustände „Waiting“ und „Terminated“ hinzugefügt. Waiting repräsentiert einen gerade erstellten Thread. Dieser geht nicht automatisch in den Zustand Runnable über, damit die Initialisierung in jedem Fall sauber abgeschlossen werden kann. Der Zustand „Terminated“ dient dazu bereits abgearbeitete Threads zu erkennen, die nicht mehr benötigt werden.

Im Prozess werden die Threads in Listen verwaltet. Für die Zustände `RUNNABLE`, `WAITING` und `BLOCKED` gibt es jeweils eine Liste. Ist ein Thread im Zustand `TERMINATED`, so wird dieser bei der nächsten Lauffähigkeits-Überprüfung vom Scheduler gelöscht. Wird der letzte Thread eines Prozesses entfernt, so wird der Prozess ebenfalls gelöscht.

Ist ein Thread im Zustand ”TERMINATED“ so wird dieser bei der nächsten Lauffähigkeits-Überprüfung vom Scheduler gelöscht. Wird der letzte Thread eines Prozesses entfernt, so wird der Prozess ebenfalls gelöscht.

Die Übergänge zwischen den Threadzuständen zeigt Abbildung 9.3.

9.2.3 Scheduler

Der Scheduler ist dafür zuständig, alle Prozesse und deren Threads zentral zu verwalten. Dazu ist er als Singleton realisiert und hat eine Liste aller Prozesse. Über die Prozesse hat er auch Zugriff auf die zugehörigen Threads. Da es sich um ein kooperatives Multitasking handelt wird aus der Liste aller Prozesse der Erste ausgewählt und nacheinander alle Threads des Prozesses abgearbeitet bis sie fertig sind oder die Ressourcen abgeben. Die Prozesse sind nach ihrem Erstellungszeitpunkt geordnet. Falls keine Prozesse mehr vorhanden sind, läuft ein Idle-Prozess, der ständig „Dispatcher::backToScheduler“ aufruft, um nach aus-

führbaren Threads zu suchen. Für die Ausführung eines neuen Prozesses benutzt der Scheduler den Dispatcher. Der Scheduler ist dafür zuständig alle Prozess und deren Threads zentral zu verwalten. Dazu ist er als Singleton realisiert und hat Zugriff auf eine verkettete Liste aller registrierten Prozesse über deren erstes Element (registeredProcesses - Änderung in: firstRegisteredProcess später). Über die Prozesse hat er auch Zugriff auf die zugehörigen Threads. Da es sich um ein kooperatives Multitasking handelt wird, wenn ein neuer Thread ausgewählt werden soll (intern: chooseNext) immer der auf den aktuellen Thread folgende (currentThread->getNextThread()), ausgewählt. Falls der aktuelle Thread keine folgengenden Threads hat, wird der nächste Prozess ausgewählt (currentThread->getParent->getNextProcess()). Falls jedoch irgendwann auch ein Prozess keinen Nachfolger mehr hat, wird wieder beim ersten Prozess begonnen. Die Prozesse sind nach ihrem Registrierungszeitpunkt geordnet, der erste Prozess ist immer der zuletzt registrierte. Falls die Prozesse keine lauffähigen Threads mehr besitzen läuft ein Idle-Prozess der ständig “Dispatcher::backToScheduler“ aufruft, um nach ausführbaren Threads zu suchen. Für die Thread-Wechsel nutzt der Scheduler den Dispatcher.

Im den Abbildungen 9.4 und 9.5 werden die internen Algorithmen des Schedulers dargestellt. Zunächst chooseNext(), der den nächsten RUNNABLE-Thread sucht, im Anschluss loadNextProcess(), der dann ggf. den nächsten Prozess und den dazugehörigen RUNNABLE-Thread findet.

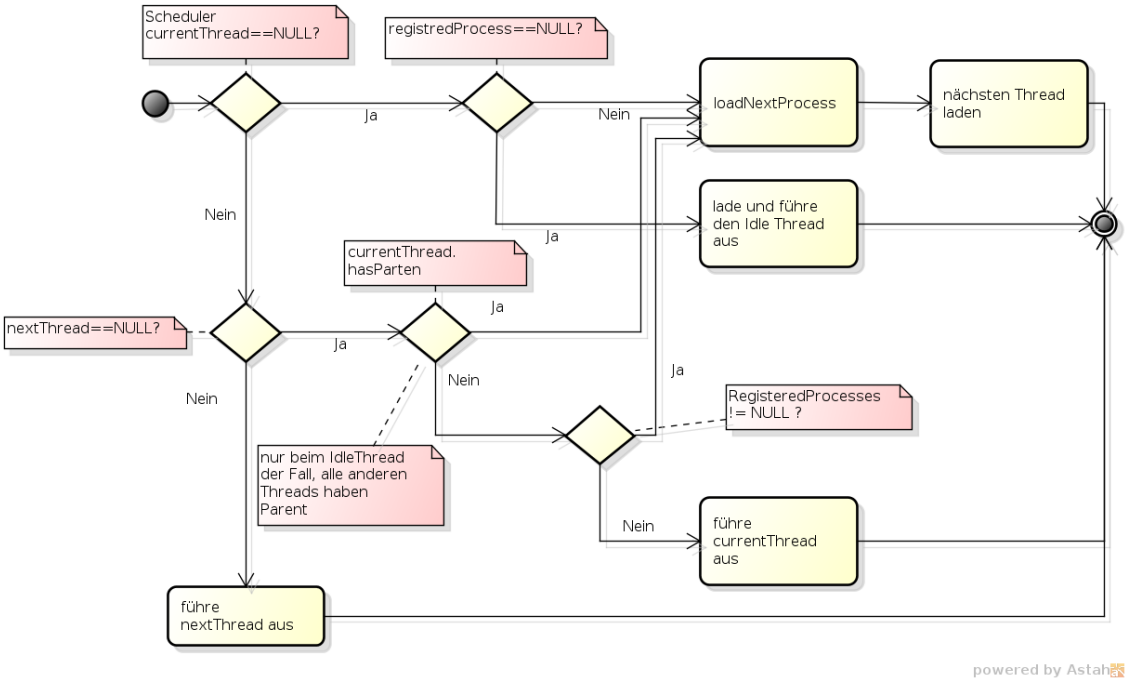


Abbildung 9.4: Scheduler

An öffentlichen Schnittstellen bietet der Scheduler die getInstance()-Methode um die Singleton-Instanz zu bekommen. Die Funktion initScheduler() bereitet

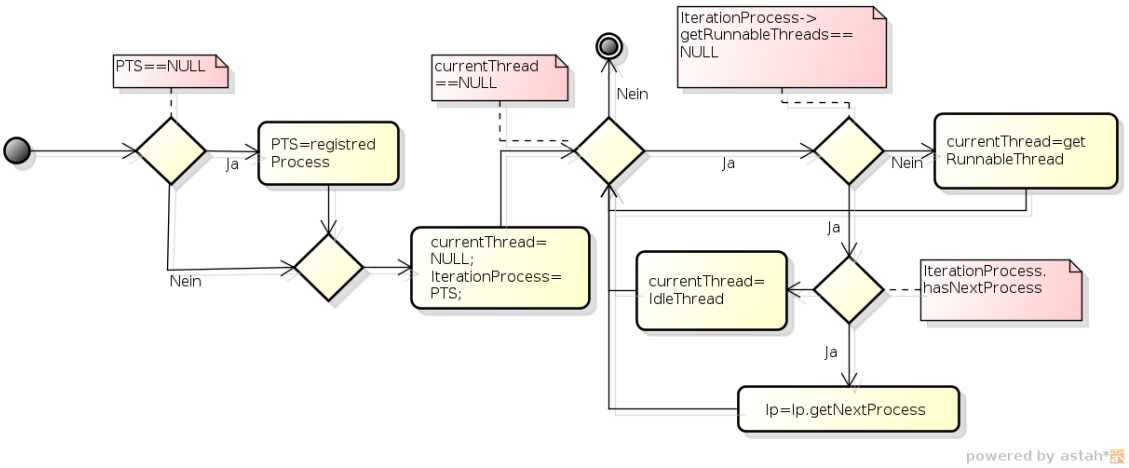


Abbildung 9.5: loadNextProcess(processToStart)

den Scheduler vor und muss nur einmalig beim Systemstart ausgeführt werden. Um den aktuellen Thread zu bekommen stellt der Scheduler die Funktion getCurrentThread() bereit. Um Prozesse zu registrieren und zu entfernen besitzt der Scheduler die beiden Methoden addProcess(Process* p) und removeProcess(Process* p). AddProcess bindet den Prozess dabei als erstes Listenelement ein, removeProcess entfernt ihn aus der Liste. Mit der Funktion blockCurrentThread(Semaphore* semaphore, Mutex* mutex) kann außerdem der aktuelle Thread blockiert werden, genaueres dazu im Teil Synchronisation. Wenn man alles vollständig initialisiert hat kann man den Scheduler mit start() bzw. start(bool testing) gestartet werden. Mit Hilfe der Angabe testing kann festgelegt werden, ob der Scheduler im Test-Modus läuft, da für Testfälle der Idle-Thread nicht gestartet wird.

9.2.4 Dispatcher

Im Kontext des FHDW-OS ist der Dispatcher dafür verantwortlich, Task-Wechsel vorzunehmen. Da es sich beim Dispatcher nur um statische Methoden handelt wurde an dieser Stelle auf ein Singleton verzichtet. Der Dispatcher wird vom Scheduler aufgerufen und leitet den Task-Wechsel ein. Zusätzlich kann er von einem Thread benutzt werden, um seine Ausführung zu unterbrechen und an den Scheduler abzugeben.
Die Klasse Dispatcher bietet dem Scheduler die Funktionen loadIdleThread, loadTss und dispatch an. Die Funktion dispatch ist für den Prozesswechsel zuständig. Wie die einzelnen Aufrufe vom Scheduler zum Dispatcher aufgebaut sind, ist dem vorangegangenen Abschnitt zu entnehmen.

Listing 9.2: Laden der TSS

```
void Dispatcher::loadTss(Selector tssSelector) {
        Selector selector = tssSelector; // needed for jump
        FarPointer farPtr;
        farPtr.offset = 0;
        farPtr.selector = selector.combined();
        __asm__ volatile("jmp␣fword␣ptr␣%0"::"m"(farPtr));
}
```

9.3 Benutzung von Prozessen und Threads

9.3.1 Threaderstellung

Um Prozesse, Threads und das zuvor vorgestellte Konzept zu nutzen, benötigt man eine Unterklasse von der abstrakten Klasse Thread in Thread.h. In dieser Unterklasse muss man die run-Methode implementieren, in der dann Aktionen ausgeführt werden können.

Um den Thread zu starten, muss man die Methode StartRunning ausführen. Dies ändert den Zustand des Threads in RUNNABLE, sodass dieser vom Scheduler als lauffähig erkannt und irgendwann ausgeführt wird.

9.3.2 Zusammenspiel mit dem Scheduler

Da das FHDW-OS auf kooperativem Multitasking basiert, müssen laufende Threads dem Scheduler melden, wenn sie ihre Ausführung unterbrechen wollen. Um diesen Vorgang durchzuführen, ruft der ausgeführte Prozess einfach die Methode `backToScheduler` vom Dispatcher auf. Diese statische Methode übergibt daraufhin an den Scheduler, der den nächsten Thread auswählt. Ist die Run-Methode eines Threads abgeschlossen, wird der Zustand des Threads in `TERMINATED` geändert. Der Scheduler erkennt dies und löscht in diesem Fall den Thread.

9.4 Fazit und Ausblick

Es wurde ein kooperatives Multitasking implementiert. Als Scheduling-Algorithmus wurde das Round-Robin Verfahren verwendet. Scheduler sowie Dispatcher dienen

dabei als zentrale Klassen, um Task-Wechsel durchzuführen. Durch die Platzierung des Idle-Thread im Kernel-Space, ist für einen Wechsel zum Idle-Thread kein Kontextwechsel nötig, da der Kernel-Space immer gemappt ist.

Es wurden Vorkehrungen getroffen, um zu einem späteren Zeitpunkt präemptives Multitasking umzusetzen. Dazu ist es notwendig, dass der Scheduler an bestimmten Punkten nicht unterbrochen werden kann. Dafür bietet es sich an, kritische Bereiche zu definieren, in denen keine Unterbrechungen gestattet sind. Dazu wurden bereits Methoden zum Betreten sowie Verlassen des kritischen Bereichs angelegt. In der Zukunft muss über eine Implementierung dieser Methoden nachgedacht werden und über Möglichkeiten Unterbrechungen zu verhindern. Denkbar wäre eine Maskierung der Interrupts. Darüber hinaus wird eine Instanz benötigt, die zeitgesteuert Prozesse bzw. Threads unterbricht und den Scheduler aufruft, der den Task-Wechsel durchführt.

Für die in diesem Kapitel vorgestellten Konzepte wurden Informationen aus folgenden Quellen verwendet:

- `www.lowlevel.eu`
- `www.wikipedia.org`

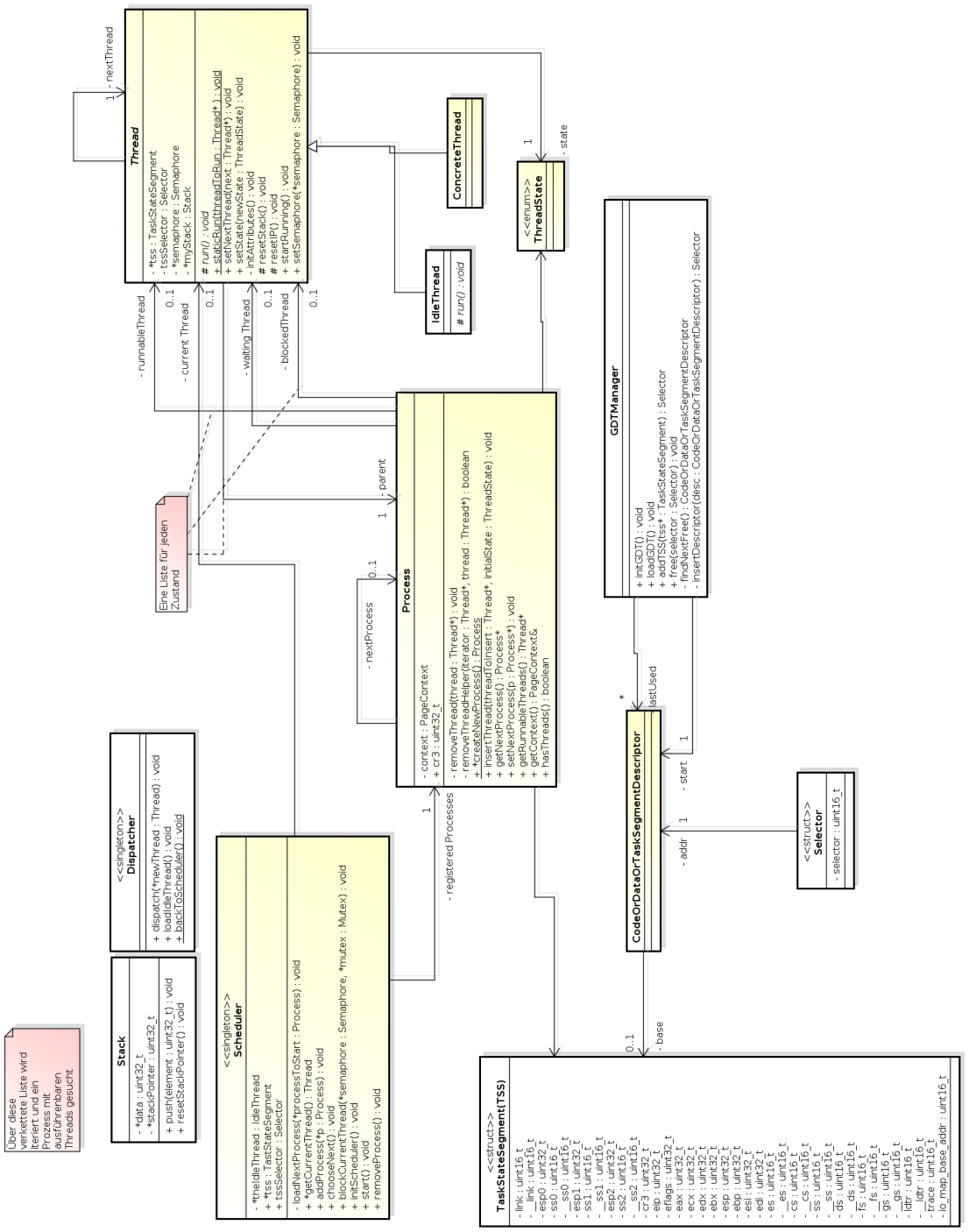


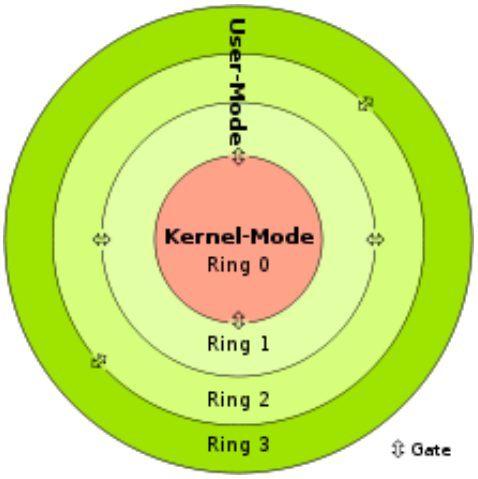
Abbildung 9.6: Klassendiagramm Tasks I

10 Interprozesskommunikation

10.1 Grundlagen

10.1.1 Sicherheitsstufen eines Prozesses

Nachdem der Zugriff auf den Speicherbereich einzelner Prozesse bereits durch Speichervirtualisierung eingeschränkt wurde, kommen im Zuge der Anforderung einer Interprozesskommunikation weitere Sicherheitsaspekte hinzu, die berücksichtigt werden müssen. Auf kritische Teile des Betriebssystems, wie dem Allozieren von Speicher, sollten nicht alle Prozesse direkten Zugriff haben.



Ein gängies Modell zur Einschränkung der Kommunikation unter den Prozessen sowie zur Einschränkung des Zugriffs auf Ressourcen, ist es die Prozesse in einzelne Privilegierungsstufen einzuteilen. Die höchste Stufe, und damit der zentrale und kritische Teil des Betriebssystems, steht ausschließlich dem Kern zur Verfügung. Nieder privilegierte Stufen haben keine Möglichkeit direkt auf eine höher privilegierte Stufe zuzugreifen. Um dennoch eine Kommunikation über diese Grenzen hinweg umzusetzen, existieren "Gates". Jeder Wechsel aus einer Schicht in eine andere erfordert einen Kontextwechsel der CPU.

10.2 Analyse

In diesem Abschnitt werden die einzelnen Aspekte der Speicheraufteilung und der Prozesskommunikation analysiert.

10.2.1 Aufgabe

In dieser Teilaufgabe soll einerseits ein Modell zur Interprozesskommunikation entwickelt und andererseits die Verwendung von Schutzmechanismen des Prozessors betrachtet und sinnvoll umgesetzt werden.

Für die Interprozesskommunikation ist es notwendig ein Konzept zur Verwendung von gemeinsamem Speicher zwischen 2,...,n Prozessen zu entwickeln. Dieser Speicherbereich muss in besonderer Weise alloziert und abgebildet werden, damit eine bestimmte Anzahl Prozesse auf ihn zugreifen kann.

Die Schutzmechanismen des Prozessors basieren im Wesentlichen auf der Verwendung der Privileglevel, um so den Zugriff auf Speicher weiter zu reglementieren. Dabei verbietet der Prozessor, dass ein Prozess auf Privileglevel 3 auf Daten im Speicher des Prozesses auf Privileglevel 0 zugreifen kann, auch nicht über Mechanismen der Interprozesskommunikation.

Das Hauptaugenmerk liegt dabei auf der Interprozesskommunikation. Hier soll ein Konzept sowie eine fertige und getestete Implementierung geliefert werden. Für den zweiten Teil der Aufgabe geht es um das Wechseln des Privileglevels. Hier wird lediglich ein Konzept sowie eine erste Implementierung geliefert. Getesteter Programmcode kann für diesen Teil noch nicht geliefert werden, da es zur Zeit der Erstellung noch nicht möglich ist einen Prozess auf einem anderen Privileglevel zu starten.

Daher ist das Erstellen von Prozessen auf anderen Privilegleveln auch nicht Bestandteil dieser Aufgabe.

10.2.2 Aufteilung des Speichers

Die Aufteilung des linearen Speichers erfolgt im FHDW OS in einen Kernel- und einen User-Bereich. Die Adressen im User-Bereich reichen von 0 - 3 GB und im Kernel-Bereich von 3 - 4 GB.

Zusätzlich unterstützt der Prozessor bis zu vier Privileglevel. Level 0 ist dabei dabei am höchsten privilegiert, während Level 3 den niedrigst-privilegierten Level

verkörpert. Für jeden Privileglevel steht je ein Stack-, Code- und Datensegment zur Verfügung.

Zurzeit werden im FHDW OS alle Prozesse auf Privileglevel 0 ausgeführt.

10.2.3 Allozieren von Speicher

Um Speicher zu allozieren, müssen im Wesentlichen zwei Schritte durchgeführt werden. Zunächst wird von dem Allokator eine Adresse im linearen Speicher des Prozesses gesucht. Dieser linearen Adresse muss durch das Physical Memory Management eine echte, physikalische Speicheradresse zugewiesen werden, die wiederum dem Paging bekannt gemacht werden muss.

Zurzeit gibt es einen Allokator für den Kernelbereich. Dieser verwaltet lediglich den Speicherbereich zwischen den Adressen 0xC0000000 und 0xFFBFF000 was dem Bereich der Adressen von 3 - 4 GB entspricht. Speicher aus dem Bereich 0 - 3 GB kann momentan nicht alloziert werden.

Der von einem Allokator allozierte Speicher ist nur in jeweils einem Pagekontext eines Prozesses gemappt, sodass kein anderer Prozess darauf zugreifen kann.

10.2.4 Kommunikation zwischen zwei Prozessen

Damit zwei Prozesse P_1 und P_2 miteinander kommunizieren können, muss es den Prozessen möglich sein Daten miteinander auszutauschen. Aus Abschnitt 10.2.3 geht bereits hervor, dass zwei Prozesse keinen gemeinsamen Speicher haben, abgesehen vom Kernel-Bereich, über den die Prozesse Daten austauschen könnten. Vom Austausch der Daten über den Kernel-Bereich wird allerdings abgesehen, da zum Beispiel beim Austauschen größerer Datenmengen zwischen Prozessen, schnell die Menge an freien linearen Adressen für den Kernel-Bereich ausgeschöpft würde.

Ein gemeinsamer Speicherbereich zwischen Prozessen muss daher entweder erzeugt werden oder die auszutauschenden Daten müssen durch einen Unterhändler in den Speicherbereich des jeweils anderen kopiert werden. Selbiges gilt auch für mehr als zwei Prozesse, daher wird in Zukunft immer nur von zwei Prozessen gesprochen.

Während beim Erzeugen eines gemeinsam genutzten Speicherbereiches nur die linearen Adressen in P_1 und P_2 auf die selben physikalischen Adressen gemappt werden müssen, ist der Aufwand beim Kopieren der Daten etwas höher. Hierzu wäre es notwendig, dass sich ein Prozess P_{copy} zunächst die Daten von

P_1 holt, also die physikalische Page in seinen linearen Adressraum abbildet, und diese anschießend in den Speicherbereich von P_2 kopiert. Auch das geht nur indem eine Art gemeinsamer Speicher mit P_2 angelegt wird. Dies erfordert neben einem höheren Programmieraufwand auch mehrere Kontextwechsel für das Austauschen dieser Daten und schließt sowieso das Erstellen gemeinsamen Speichers ein. Aus diesem Grund erscheint es sinnvoll direkt einen gemeinsamen Speicherbereich zwischen P_1 und P_2 einzurichten.

10.2.5 Wechsel des Privileglevels

Muss außerdem noch ein Wechsel des Privileglevels durchgeführt werden, ist das Einrichten eines gemeinsamen Speicherbereiches nicht in der oben beschrieben Form möglich. Der Prozessor lässt dies nicht zu, da ein Speicherblock immer nur in einem Stack-, Code- oder Datensegment liegen und sich daher nicht gleichzeitig in zwei unterschiedlichen Privilegleveln befinden kann.

Hier ist es unumgänglich einen Unterhändler zu haben, der ein Kopieren vornimmt. Dieser muss auf einem höher privilegierten Level ausgeführt werden, damit er das Kopieren übernehmen kann. Um diesen wiederum aufrufen zu können, können nur bestimmte Kernel-Routinen verwendet werden. Ein einfacher Call würde mit einer Generell-Protection Exception vom Prozessor beendet werden. Möglichkeiten dennoch einen Aufruf in einen Ring mit höherem Privilegelevel durchzuführen, sind beispielsweise Interrupts oder Call-Gates. Bezüglich der Unterschiede zwischen den beiden Möglichkeiten sei hier auf den System Programming Guide von Intel verwiesen. In Abbildung 10.1 wird der Aufbau eines Call Gate Descriptors in der Intel IA-32 Architektur dargestellt. Wenn von Call Gates gesprochen wird ist das als Kurzform für Call Gate Deskriptoren zu verstehen. Ein Call Gate Deskriptor kann in der GDT oder in der LDT liegen.

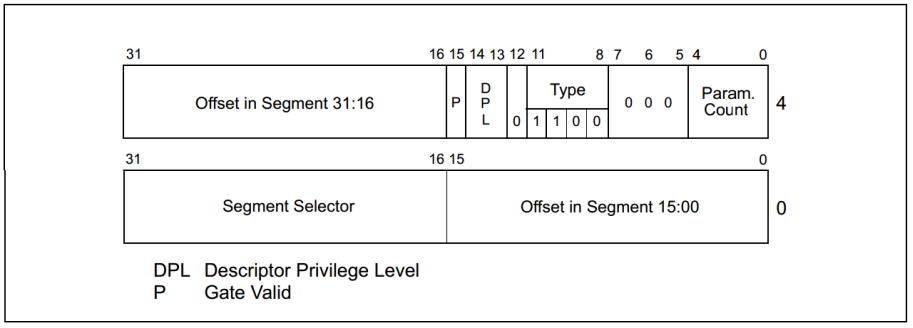


Abbildung 10.1: Aufbau eines Call Gates

Wie in der Abbildung zu sehen, ist ein Call Gate 8 Byte groß. Ein Call Gate stellt damit insgesamt 6 Funktionen bereit:

- Spezifiziert das Codesegment, auf das zugegriffen wird.
- Definiert einen Eintrittspunkt für eine Prozedur in dem spezifizierten Codesegment.
- Spezifiziert das Privileglevel, das von einem Aufrufer benötigt wird, der versucht auf die Prozedur zuzugreifen.
- Wenn ein Stackwechsel vorkommt, spezifiziert es die Anzahl optionaler Parameter, die zwischen den Stacks kopiert werden.
- Definiert die Größe (16-Bit oder 32-Bit) für die Werte, die auf den Ziel Stack gelegt werden. (In unserem Fall immer 32Bit)
- Spezifiziert ob der Call Gate Deskriptor valide ist.

10.3 Entwurf

In diesem Abschnitt wird der Entwurf für die Interprozesskommunikation erläutert. Zunächst wird der allgemeine Ablauf der Interprozesskommunikation und die Prozess-Kernel Kommunikation beschrieben. Danach wird auf einzelne spezielle Teilaspekte eingegangen.

10.3.1 Allgemein

Um die Kommunikation aus einem Prozess zum Kernel oder zu einem anderen Prozess möglichst einfach zu gestalten, werden die beiden Kommunikationen nahezu identisch gehalten. Dabei wird jedem Prozess ein Objekt vom Typ CommandRelay zur Verfügung gestellt, über das die gesamte Kommunikation aus einem Prozess heraus durchgeführt wird. Dieses Objekt wird Ring 3 Prozessen bei der Erzeugung eines neuen Thread auf den Stack gelegt. Die Kommunikation erfolgt mithilfe von objektifizierten Nachrichten, die dem CommandRelay übergeben werden.

Wenn einem CommandRelay eine Nachricht übergeben wird, werden zwei Schritte durchgeführt.

- Zunächst wird ein Wechsel der Privelegstufen mithilfe eines Call Gates durchgeführt.
- Darauffolgend wird die Nachricht an den Zielprozess zur weiteren Verarbeitung übermittelt. Der zweite Schritt wird nur ausgeführt, wenn die Nachricht

für einen anderen Prozess bestimmt ist. Wenn die Nachricht für den Kernel bestimmt ist, wird diese direkt nach dem Wechsel der Privelegstufen abgearbeitet. Im anderen Fall wird die Nachricht zur Abarbeitung an den anderen Prozess übergeben.

In Abbildung 10.2 sind die in diesem Abschnitt erwähnten Klassen dargestellt.

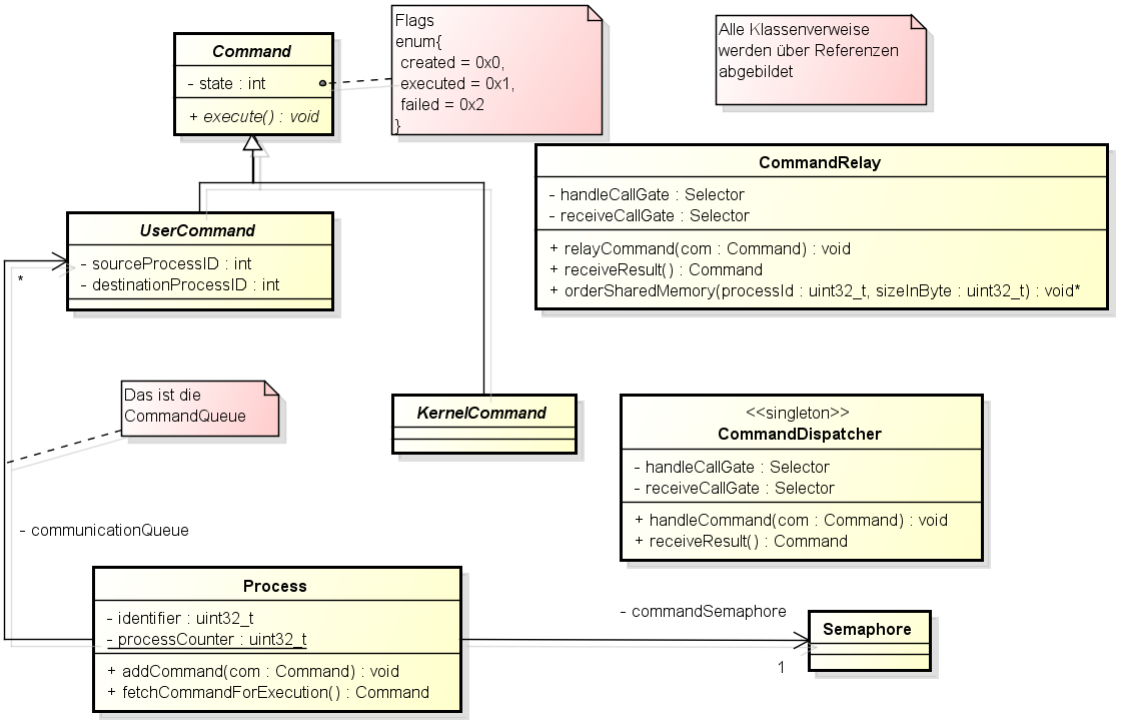


Abbildung 10.2: Kapselung von Nachrichten in Commands

10.3.2 Objektifizierung der Nachrichten

Die Objektifizierung der Nachrichten wird mithilfe eines Command-Patterns durchgeführt. Die Commands haben mehrere Flags, die den Status der jeweiligen Nachricht anzeigen, wie zum Beispiel abgearbeitet oder ausstehend. Des Weiteren findet eine Unterteilung mithilfe von Vererbung zwischen Nachrichten für den Kernel mit der Klasse KernelCommand und Nachrichten für andere Prozesse mit der Klasse UserCommand statt.

Nachrichten können die Zustände ”erstellt“, ”ausgeführt“ und ”fehlgeschlagen“ annehmen. Beim Erstellen einer Nachricht bekommt diese den Zustand “erstellt” zugewiesen. Je nachdem, ob die Ausführung erfolgreich war oder nicht, wechselt der Zustand in “ausgeführt” oder “fehlgeschlagen”.

10.3.3 Kommunikation über Privileglevel-Grenzen

Für den Wechsel der Privilegstufen werden Call Gates genutzt. In dem niedrig previligierten Code steht das CommandRelay zur Verfügung. Dieser kapselt den Zugriff auf die Call Gates. Das Gegenstück im Ring 0 Code ist der Command-Dispatcher. Die Call Gate Aufrufe, im CommandRelay, rufen die entsprechenden Gegenstücke im CommandDispatcher auf.

Die benötigten Call Gates dafür werden bei der Initialisierung des Command-Dispatchers in der GDT erzeugt. Aufrufe die im CommandRelay durchgeführt werden, werden immer über Call Gates an den CommandDispatcher übergeben, auch wenn der Aufruf in Ring 0 durchgeführt wird.

Commands, die an den Kernel gerichtet sind, können so direkt vom Command-Dispatcher ausgeführt werden. Bei der Bearbeitung von Commands für andere Prozesse sind weitere Schritte nötig, die im folgenden Abschnitt erläutert werden.

10.3.4 Interprozesskommunikation

Wie in Abschnitt 10.2.4 beschrieben, gibt es zwei Möglichkeiten einen Austausch von Daten zwischen Prozessen zu realisieren, was eine Voraussetzung für die Kommunikation zwischen zwei Prozessen ist. Aufgrund der wenigen Wechsel des Pagekontext und des nicht vorhandenen Kopieraufwands, wird in diesem Entwurf zugunsten des gemeinsam verwendeten Speichers entschieden.

Nachfolgend wird zunächst die Verwaltung vom gemeinsamen Speicher beschrieben. Darauf erfolgt die Erläuterung des Austauschs der Nachrichten.

Gemeinsamer Speicher

In Abbildung 10.3 wird schematisch dargestellt, wie das Erstellen eines gemeinsamen Speichers durchgeführt wird.

Die Prozesse P_1 und P_2 verfügen je über einen eigenen linearen Adressbereich von 0GB bis 3GB (blau dargestellt), die den Adressen des User-Bereichs (unterer, roter Kasten) entsprechen. Allerdings werden die beiden linearen Adressbereiche der Prozesse auf unterschiedlichen physikalischen Speicher (schwarz) gemappt. Soll nun ein gemeinsamer Speicherbereich für P_1 und P_2 eingerichtet werden, werden die linearen Adressen S1.1 und S1.2 aus dem jeweiligen linearen Adressräumen der Prozesse auf dieselbe physikalische Adresse gemappt.

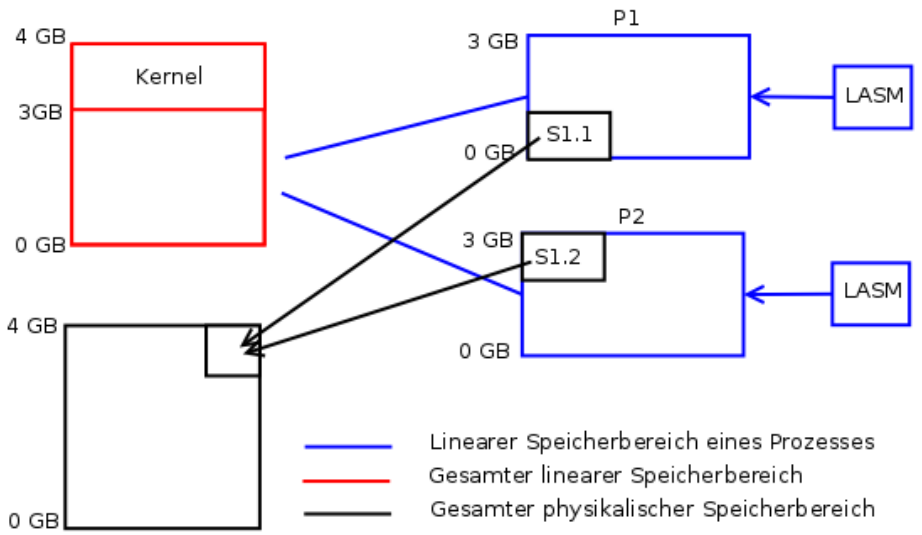


Abbildung 10.3: Aufbau gemeinsamen Speichers

Damit zunächst ein freier Bereich im linearen Adressbereich eines Prozesses ermittelt werden kann, ist eine Verwaltung von diesem nötig. Die Verwaltung des linearen Adressraumes eines Prozesses übernimmt dabei ein eigener LinearAddressSpaceManager (LASM), der in Abbildung 10.3 rechts in blau dargestellt ist. In Abbildung 10.4 ist zu sehen, dass der LinearAddressSpaceManager eine Liste mit Adressraumreservierungen besitzt und so weiß, welche Bereiche im linearen Adressraum noch ungenutzt sind. Auch bei der Erzeugung eines Allokators im User-Bereich muss zunächst linearer Adressraum vom LASM reserviert werden. Dieser reservierte, lineare Adressraum wird dem Allokator zugewiesen, der diesen dann verwaltet. Nicht zur Aufgabe des LASM gehört es, die reservierten Bereiche auf physikalischen Speicher zu mappen und im Paging einzutragen.

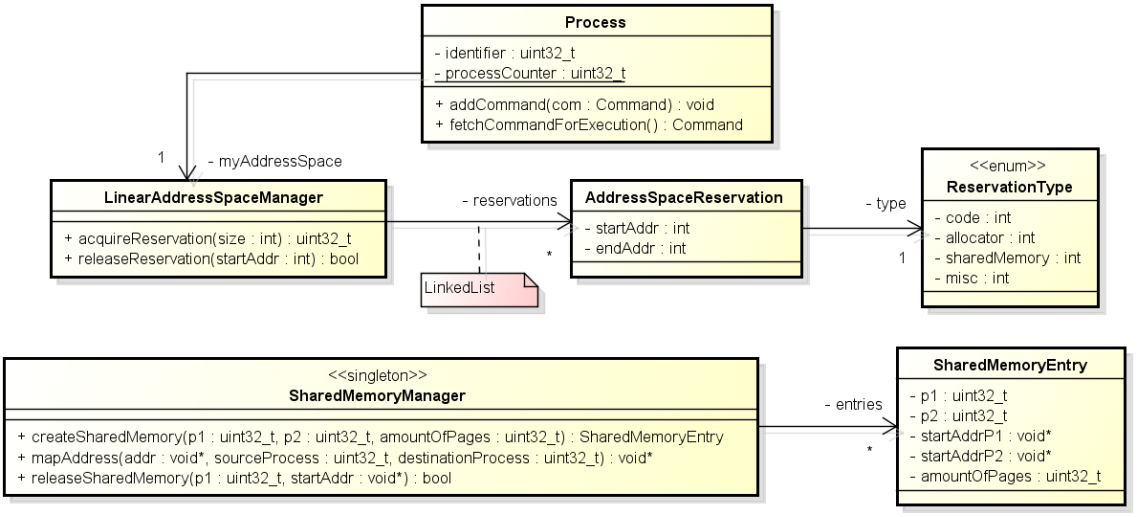


Abbildung 10.4: SMM und LASM

Der SharedMemoryManager (SMM) erstellt und verwaltet die gemeinsamen Speicherbereiche zwischen den Prozessen. Wie in Abbildung 10.4 gezeigt, verwendet

er dafür eine Liste von SharedMemoryEntrys. In denen sind die wesentlichen Informationen zu einem gemeinsamen Speicher enthalten. Dabei kann es mehrere solcher Einträge für ein Prozesspaar P_1 und P_2 geben. Dies hat den Grund, dass nicht garantiert werden kann, dass ein reservierter Speicherblock für die gesamte zukünftige Kommunikation reichen wird. Ebenso wenig kann garantiert werden, dass ein vorhandener Speicherblock erweitert werden kann, falls mehr Speicher benötigt wird. Aus diesem Grund wird für jede Kommunikation zwischen P_1 und P_2 ein neuer, gemeinsamer Speicherbereich eingerichtet.

Kommunikation

Nun verfügen P_1 und P_2 über einen gemeinsamen Speicher in dem sie beide lesen und schreiben können. Jetzt muss P_1 noch P_2 mitteilen, dass dieser etwas mit den Daten in dem gemeinsamen Speicher tun soll. Da P_1 allerdings keine Operation von P_2 aufrufen kann, wird ein Umweg mithilfe der Commands über den CommandDispatcher im Kernel in Kauf genommen.

Um das Command an den CommandDispatcher weiterzuleiten wird das Objekt CommandRelay genutzt, welches in Abbschnitt 10.3.3 vorgestellt wurde. Hierfür ist wichtig, dass zuvor ein gemeinsamem Speicher angefordert wurde, was ebenfalls mithilfe eines Commands, welches an den Kernel gerichtet wird, durchgeführt werden kann. In diesem gemeinsame Speicher muss das Command Objekt, welches an Prozess P_2 geschickt werden soll, erstellt werden.

Bevor das Command durch den CommandDispatcher an den Prozess P_2 weitergeleitet wird, wird zunächst überprüft ob ein gemeinsamen Speicher zwischen P_1 und P_2 existiert und ob sich das Command in diesem befindet. Darauf wird die lineare Adresse des Commands von P_1 in die lineare Adresse für P_2 übersetzt, da sich der gemeinsame Speicher nicht zwangsläufig bei beiden Prozessen an der selben lineare Adresse befinden muss.

Nun kann das Command an Prozess P_2 weitergeleitet und abgearbeitet werden. Hierfür standen zwei Ansätze zur Auswahl:

- Der erste Ansatz war der synchrone Ansatz. Bei diesem wird das Command direkt an den Prozess P_2 überstellt und es wird ein sofortiger Prozesswechsel eingeleitet und P_2 beginnt mit der Abarbeitung. Nach der Fertigstellung folgt ein sofortiger Wechsel zurück zu P_1 mit dem Ergebnis.
- Im zweiten Ansatz wurde eine asynchrone Abarbeitung bevorzugt. Hierbei erhält jeder Prozess eine Warteschlange in der die Commands eingereiht werden, die der Prozess abzuarbeiten hat. Ein zusätzlicher Thread in je-

dem Prozess übernimmt die Aufgabe diese Warteschlange abzuarbeiten, wenn der Thread durch den Scheduler aktiviert wird. Die Ergebnisse der Abarbeitung werden in die Warteschlange des Quellprozesses eingereiht.

Von den beiden Ansätzen wurde der asynchrone Ansatz ausgewählt, da dieser keinen Eingriff in das Scheduling des Betriebssystemes erfordert. Weitere Punkte die diese Auswahl begünstigen sind, dass aus einem asynchronen Ansatz simpel eine synchrone Kommunikation mithilfe von Sperren erstellt werden kann, als auch die einfache Umsetzung im Gegensatz zum synchronen Ansatz.

Um den ausgewählten Ansatz umzusetzen erhält jeder Prozess sowohl eine Queue in der Commands, die abgearbeitet werden sollen, eingereiht werden, als auch eine Semaphore, die Freigaben für den Zugriff auf die Queue erteilt. Des Weiteren wird mithilfe von CommandRelay und CommandDispatcher ein Zugriff aus Ring–3–Code auf die Queue geschaffen, da diese sich im Kernel-Bereich befindet.

Somit wird um das Command weiterzuleiten, das Command durch den CommandDispatcher in die Queue des Prozesses P_2 gelegt. Wenn dies gesehen ist, ist der Ablauf auf der Seite des Prozesses P_1 abgeschlossen.

Nun muss der Prozess P_2 einen Thread besitzen, der die in der Queue vorhandenen Commands abarbeitet. Da die Queue sich im Kernel–Bereich befindet, muss auch hier der Umweg über eine Operation des CommandRelays und CommandDispatcher vollzogen werden. Um etwaige Ergebnisse der Abarbeitung des Commands an den Prozess P_1 zurückzumelden, muss das spezielle Command entsprechende Attribute vorsehen. In diese werden die Ergebnisse der Abarbeitung eingetragen. Um nun die Abarbeitung an den Prozess P_1 zurückzumelden, wird das abgearbeitete Command, in dem die Ergebnisse eingetragen wurden, wieder an den CommandDispatcher mithilfe des CommandRelay übergeben. Dieser reiht es nun auch in die Queue des Prozess P_1 der wie P_2 nun das Command mithilfe des CommandRelay abfragen muss.

10.4 Implementierung

Die Implementierung spiegelt den Entwurf in allen Punkten wieder. So wurden alle Klassen gemäß Abbildung 10.4 und Abbildung 10.2 codiert bzw. erweitert.

10.4.1 Linear Memory Management

Der LinearAddressSpaceManager benötigt eine sortierte Liste, um eine performante Suchen über seine Reservierungen durchführen zu können. Um eine Sortierung zu vereinfachen, wurde die Klasse LinkedList als Implementierung der List-Klasse hinzugefügt. Sie erweitert das Interface um eine Methode zum Einfügen eines Elements nach einem Gegebenen.

Listing 10.1: Einfügen eines Elements nach einem bestimmten

```
1  template<class T>
2  bool LinkedList<T>::add(T* insert, T* after){
3    LinkedListEntry<T> * i = this->firstElement;
4    while (i->getEntry() != after){
5      i = i->getNextItem();
6      if(!i){
7        return false;
8      }
9    }
10   LinkedListEntry<T> * theNew = new LinkedListEntry<T>(insert);
11   theNew->setNextItem(i->getNextItem());
12   i->setNextItem(theNew);
13   this->currentSize++;
14   return true;
15 }
```

Der Algorithmus zum Suchen eines freien Speicherbereichs prüft, ob der Bereich zwischen Endadresse einer Reservierung und der Startadresse des nächsten Elements ausreichend groß ist. Falls ja, wird an dieser Stelle eine neue Reservierung eingefügt, andernfalls die nächste Reservierung geprüft. Ist ein Platz gefunden, wird die lineare Adresse zurück gegeben, an der die erste Page zu finden ist.

Für den Fall, dass keine Speicher mehr vorhanden ist, wird die Adresse 0x00000000 zurück gegeben. Die allererste Reservierung in einem linearen Adressraum erhält daher immer die Page mit der linearen Adresse 0x00000100.

Zu beachten ist auch, dass der LinearAddressSpaceManager lediglich den linearen Adressraum verwaltet. Er führt keine Allokation und kein Paging durch!

Alle Klassen, die zum LinearAddressSpaceManager gehören, wurden in dem Namensraum memory::lmm definiert.

10.4.2 Identifikation eines Prozesses

Zur Identifikation von Prozessen wurden Prozess–IDs eingeführt. Diese werden im Konstruktor des Prozesses mithilfe eine statische Counter Variable in der Prozessklasse gesetzt, die nach jeder Erzeugung eines Prozesses automatisch erhöht wird. In Quelltext 10.2 und Quelltext 10.3 wird dieser Mechanismus kurz dargestellt.

Listing 10.2: Konstruktor von Prozess

```
Process::Process(PageContext * theProcessContext, uint32_t identifier){
  this->identifier = identifier;
}
```

Listing 10.3: Aufruf des Konstruktors der Klasse Prozess

```
Process* p = new Process((PageContext*) &context, processCounter++);
```

10.4.3 Shared Memory Management

Der SharedMemoryManager erstellt und verwaltet gemeinsame Speicherbereich zwischen zwei Prozessen. Zur Identifikation eines Prozesses wird die in Abschnitt 10.4.2 eingeführte Prozess–ID genutzt.

Zur Verwaltung verwendet der SharedMemoryManager eine ArrayList von SharedMemoryEntrys. Bei der Erstellung eines gemeinsamen Speichers werden folgende Tätigkeiten durchgeführt.

- Es wird für jeden Prozess ein Bereich im linearen Adressraum reserviert.
- Es wird physikalischer Speicher reserviert.
- Die beiden linearen Adressräume der Prozesse werden auf den gleichen physikalischen Speicher gemappt.

Beim Freigeben eines SharedMemory Bereiches wird dieser über die Prozess ID, einer der beiden Prozesse des gemeinsamen Speichers, und der zugehörigen Startadresse des Speicherbereichs identifiziert. Wichtig ist, dass Prozess ID und Startadresse zueinander gehören.

10.4.4 Kommunikation zwischen Prozessen

Um die Kommunikation zu realisieren, wurde das Command-Pattern aus Abbildung 10.2 umgesetzt.

Dabei spielen die Klassen CommandRelay und CommandDispatcher eine zentrale Rolle. Im CommandRelay wird über die Methode relayCommand(Command* command) ein Command-Objekt an den Dispatcher geschickt. Quelltext 10.4 zeigt den dafür notwendigen Programmcode.

Listing 10.4: CommandRelay::relayCommand

```
memory::FarPointer farPtr;
farPtr.selector = handleCallGate.combined();
scheduler::Scheduler::getInstance().getCurrentThread()->
    pushToStack((uint32_t)command);
__asm__ volatile("call␣fword␣ptr␣%0"::"m"(farPtr));
```

Um Call Gates in der Objektstruktur von C++ einfacher zu verwenden, wurde ein Struct als dessen Repräsentation verwendet und in der GDT abgelegt. Die Call Gates werden im Zuge der Initialisierung des CommandDispatchers erstellt.

Hierfür wurde der GDTManager um die Methode addCallGateDescriptor(CallGateDescriptor * cgd) erweitert. Diese wiederum nutzt die bereits vorhandene, private Methode insertDescriptor um den Call Gate Descriptor in die GDT zu legen.

10.5 Tests

Zum Testen wurde das bisher vorhandene Testframework verwendet.

Automatisierte Tests für die erstellten Klassen sind nur bedingt möglich, da der Scheduler dafür verwendet werden muss. Dies führt zu einer „unsauberen“ Umgebung für die Tests, da während des Tests auch andere Threads durch den Scheduler gestartet und so die Erwartungswerte verfälscht werden könnten.

Dennoch wurde eine Methode zum Testen angelegt, die die zwei Prozesse erstellt und diese dem Scheduler hinzufügt. Einer der beiden Prozess besitzt einen Thread, der ein Command absetzt, der Andere einen zum Abarbeiten von Commands.

Bei dieser Kommunikation werden alle erstellten Klassen verwendet und so getestet.

Wie bereits erwähnt konnte Kommunikation zwischen Prozessen unterschiedlicher Privileglevel aufgrund der fehlenden Voraussetzungen[48] nicht getestet werden.

[48] siehe Abschnitt 10.2.1

10.6 Verwendung

Die Verwendung des Frameworks zur Interprozesskommunikation wird anhand eines Beispiels demonstriert.

Für dieses Beispiel wird angenommen, dass die Mechanismen zur Ausgabe von Text auf der Konsole nicht im Kernel enthalten sind, sondern in einen eigenen Treiber in einem eigenem Prozess $P_{console}$ ausgelagert sind.

Ein Thread T_1 eines Prozesses P_1 möchte nun den Text „Hallo Welt“ auf der Konsole ausgeben.

Hierfür wird eine entsprechende Command-Klasse benötigt. In diesem Beispiel erbt daher die Klasse „ConsoleCommand“ von „UserCommand“. Die Klasse bekommt ein Attribut textForOutput, in das der auszugebende Text gespeichert werden kann. In $P_{console}$ muss es nun eine Instanz geben, die ständig ein down() auf der Liste der auszuführenden Commands macht und ein dort abgelegtes Command ausführen kann. Code, der ausgeführt werden soll, steht dabei in der execute-Methode des ConsoleCommand-Objekts.

Möchte nun T_1 seinen Text ausgeben, erwirbt dieser über sein CommandRelay einen gemeinsamen Speicherbereich mit $P_{console}$. T_1 erhält eine Adresse aus seinem linearen Speicherraum, an der er mit einem placement new nun das ConsoleCommand erstellen und das Attribut textForOutput mit seinem Text „Hallo Welt“ füllen kann. Nun kann er die Nachricht, wieder über sein CommandRelay, abschicken. Den Programmcode für diesen Vorgang Quelltext 10.5.

Listing 10.5: Abschicken des ConsoleCommands

```
1 uint32_t addr = CommandRelay::orderSharedMemory(
2                         Pconsole, sizeof(ConsoleCommand));
3 ConsoleCommand * command = new(addr) ConsoleCommand();
4 command->setTextForOutput("Hallo Welt");
5 CommandRelay::relayCommand(command);
```

Möchte T_1 eine Antwort abwarten, kann es die Methode receiveResult() des CommandRelays aufrufen. Dort wartet er solange, bis es eine Antwort gibt.

10.7 Fazit

Die vorliegende Arbeit liefert ein Modell und in weiten Teilen eine Implementierung zur Interprozesskommunikation. Die Kommunikation über gemeinsam

genutzte Speicherbereiche sowie Privilelevel als Schutzmechanismus für den Prozessor bilden eine mögliche Umsetzung der Interprozesskommunikation. Die Objektifizierung der Nachrichten ermöglicht es weitere Kommandos in Zukunft leicht zu ergänzen.

Eine fehlende ausformulierte Aufgabenstellung sorgte für einen großen Spielraum bei der Umsetzung. Dies führte unter anderem dazu, dass an Bereichen gearbeitet wurde, die außerhalb unseres Zuständigkeitsbereich waren. Weiterhin kollidierte die Umstrukturierung des Testframeworks mit unserer Aufgabe. Dies erschwerte es erste Implementierungen zu testen. Darüber hinaus wurden im Zuge des Testens Fehler im Paging des User-Spaces gefunden, die von uns behoben wurden.

10.8 Ausblick

Abschließend ist festzustellen, dass es noch einige offene Punkte gibt, die sich direkt oder indirekt aus den umgesetzten Anforderungen ergeben.

Zunächst ist eine Möglichkeit zu schaffen, um Prozesse in anderen Privilegleveln erstellen zu können. Daraus ergibt sich, dass auch die Treiber auf Privileglevel 1 ausgeführt werden sollen.

Weiterhin ist denkbar das Anlegen des SharedMemory sowie des CommandObjekts in dem CommandRelay zu kapseln. Auch das Erstellen und Versenden des CommandObjekts an den CommandDispatcher könnte in dieser Methode im CommandRelay ausgelagert werden.

Ebenso sind noch weitere Schnittstellen der Interprozesskommunikation zu ermitteln und die entsprechenden Command-Klassen dafür zu entwickeln.

Letztendlich fehlt noch eine Möglichkeit einzelne Prozesse im Falle eines Fehlers zu terminieren, ohne dass dabei das gesamte Betriebssystem funktionsunfähig wird.

11 Scheduler

11.1 Einleitung

Das Multitasking in der derzeitigen Entwicklungsstufe des FHDW-OS wurde als kooperatives Multitasking umgesetzt. Im Verlauf dieses Projektes soll der Mechanismus auf preemptives Multitasking umgestellt werden. Dazu wird der jetzige Scheduling-Mechanismus überarbeitet. Weiterhin müssen verschiedene Level eingeführt werden, in denen unterschieden wird, ob Ausführungen unterbrochen werden können. Außerdem muss eine Kommunikation zwischen diesen Ebenen geschaffen werden, sodass sich verschiedene Ebenen eine Ressource teilen können. Desweiteren sollen Tasklets umgesetzt werden, damit Code schnell nach dem Ausführen eines Interrupts durchgeführt werden kann, ohne das Betriebssystem lange zu behindern.

11.2 Analyse

Im nachfolgendem Abschnitt wird analysiert, wie Ausführungslevel in anderen Betriebssystemen umgesetzt sind und wie eine Synchronisation zwischen den einzelnen Ausführungsleveln aussehen kann. Desweiteren wird aufgezeigt, wie ein preemptives Multitasking mit den Leveln zusammen spielen kann und wie Tasklets sinnvoll umgesetzt werden können.

11.2.1 Ausführungslevel

Bestimmte Code-Segmente dürfen nicht unterbrochen werden, bzw. sind nicht fortsetzbar. Somit muss gewährleistet werden, dass ein Unterbrechen an bestimmten Stellen nicht auftreten kann. Gängige Betriebssysteme, wie Linux oder Windows, verwenden dazu verschiedene Level. Diese signalisieren, ob der ausgeführte Quelltext derzeit unterbrochen werden darf oder nicht. Im wesentlichen werden dazu zwei Level benötigt: Ein Level in dem Code ausgeführt wird, der reentrant ist und verdrängt werden kann und ein Level auf dem keine Verdrängung

jedoch eine Unterbrechung stattfinden darf. Zu diesen zwei Stufen wird noch eine höhere Stufe benötigt. In diesem Level werden die eingehenden Hardware-Interrupts behandelt. Sie hat die höchste Priorität. Innerhalb dieser Stufe können noch weitere Abstufungen vorgenommen werden. Den Interrupts werden Prioritäten zugeordnet. Der Timer-Interrupt hat die höchste Priorität. Dieser darf durch niemanden unterbrochen werden. Bei niedrigstufigen Interrupts können jedoch Unterbrechungen durch höhere Interrupts auftreten.

Verhalten der Ausführungslevel beim Threadwechsel

Abbildung 11.1 zeigt, wie Level erhöht und erniedrigt werden, wenn es zu einem Threadwechsel durch den Scheduler kommt.

TimerInt	Dispatch	Passive	Thread A	ISR (Kontext A)	Thread B	ISR (Kontext B)
		X	doSomething			
		X	TimerInterrupt			
	X	X				
X	X			maskiere Interrupts		
X				End of Interrupt		
X				Interrupt Handler		
X	X			Maskierung aufheben		
	X	X		Tasklet ausführen		
	X	X		neuen Thread wählen		
		X		Dispatch Thread B		
		X			doSomething	
		X			TimerInterrupt	
	X	X				
X	X					maskiere Interrupts
X						End of Interrupt
X						Interrupt Handler
X	X					Maskierung aufheben
	X	X				Tasklet ausführen
	X	X				neuen Thread wählen
		X				Dispatch Thread A
		X		RETI		
		X	doSomething			

Abbildung 11.1: Verhalten der Ausführungslevel beim Threadwechsel

Der Ablauf beginnt damit, dass Thread A dabei ist auf dem Passive-Level Code auszuführen. Dabei wird er durch den Timer-Interrupt unterbrochen. Zuerst wird das Level auf Dispatch-Level angehoben und danach auf das Timer-Level. Auf dem Weg dorthin werden alle anderen Interrupts (inklusive dem Timer-Interrupt) maskiert.

Sobald das geschehen ist, wird der „End of Interrupt“ gesendet und die Interrupts werden wieder aktiviert. Auf diesem Level wird nun die Standard Interrupt bearbeitet und durch das „InterruptSubSystem“ mit seinen „InterruptHandlern“ durchgeführt. Sobald diese Handler fertig sind, wird das Level schrittweise herab gesenkt, wobei die Interrupts demaskiert werden.

Beim Wechsel von „Dispatch“ auf „Passive“ werden zuerst die Tasklets ausgeführt. Danach wird der Scheduler benachrichtigt. Dieser wählt einen neuen Thread und dispatched ihn, sobald das Passive-Level erreicht wurde. Anzumerken ist, dass dies alles im Kontext von Thread A passiert.

Der zweite Thread B führt nun seinen Code auf dem Passive-Level durch. Ein erneuter Timer-Interrupt lässt die Level wie beim ersten Mal gleichermaßen ansteigen und absinken. Hierbei ist anzumerken, dass beim Dispatchen von Thred A erst wieder in den ISR Kontext von A gesprungen wird – dieser wurde schließlich auch beim Dispatchen von Thread B verlassen – und ein RETI(Return from Interrupt) erst wieder den normalen Thread A weiterlaufen lässt.

Verhalten der Ausführungslevel bei mehreren Interrupts

Abbildung 11.2 zeigt, wie Level erhöht und verringert werden, wenn ein Interrupt durch einen anderen Interrupt unterbrochen wird.

TimerInt	Keyboard	Dispatch	Passive	Thread A	ISR Keyboard (Kontext A)	ISR Timer (Kontext A)
			X	doSomething		
			X	KeyboardInterrupt		
		X	X			
	X	X			Interrupts maskieren	
	X				End of Interrupt	
	X				KeyboardHandler	
	X				Timer Interrupt	
X	X					Interrupts maskieren
X						End of Interrupt
X						Timer Interrupt Handler
X	X					Timer Interrupt desmaskieren
	X					RETI
	X	X			Keyboard Interrupt demaskieren	
		X				
		X	X		Tasklets ausführen	
			X		RETI	
			X	doSomething		

Abbildung 11.2: Verhalten der Ausführungslevel bei mehreren Interrupts

Der Ablauf beginnt damit, dass Thread dabei ist auf dem Passive-Level Code auszuführen. Dabei wird er durch den KeyboardInterrupt unterbrochen. Zuerst wird das Level auf Dispatch-Level angehoben und danach auf das Keyboardlevel. Auf dem Weg dorthin werden alle anderen Level – außer dem Timer-InterruptLevel – maskiert.

Sobald das geschehen ist, wird der „End of Interrupt“ gesendet und die Interrupts werden wieder aktiviert. Auf diesem Level wird nun die Standard Interruptbearbeitung durch das „InterruptSubSystem“ mit seinen „InterruptHandlern“ durchgeführt. So kann der KeyboardHandler eingegangene Tastendrücke auslesen.

Wird dieser bei seiner Arbeit durch den Timer-Interrupt unterbrochen, wird das Level in der neuen ISR (Interrupt Service Routine)erhöht. Dabei wird zusätzlich

zu den bisherigen Maskierungen der Timerinterrupt maskiert und ein End of Interrupt wird gesendet. Nun wird der TimerInterruptHandler ausgeführt und der TimerInterrupt wieder demaskiert, sobald wieder das KeyboardInterruptLevel erreicht wurde.

Mit einem „RETI“ wird in den Keyboard ISR zurückgesprungen, die nun ungehindert weiterarbeiten kann. So demaskiert sie beim Levelabstieg erneut alle Interrupts und springt mit dem RETI zurück in die Bearbeitung von Thread A.

11.2.2 Synchronisation

In einem Multitaskingbetriebssystem können schwer auffindbare Programmfehler durch unbeabsichtichtige Wettlaufsituationen entstehen. Ein sogenannter kritischer Wettlauf (Race Condition) entsteht, wenn das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängig ist. Die Bereiche in denen kritische Wettläufe stattfinden können, heißen kritische Bereiche.

Auch in einem Singletaskingbetriebssystem kann es zu Problemen kommen, da beim preemptiven Multitasking der Programmablauf in kritischen Bereichen unterbrochen werden kann und somit beim Ausführen von anderem Code Race Conditions entstehen können.

Als Lösung für dieses Problem dienen Sperren, die eingesetzt werden, um sicherzustellen, dass nur ein Task den kritischen Bereich zur gleichen Zeit betreten kann. Dabei unterscheidet man hauptsächlich zwei Typen von Sperren, die Spinlocks und die Semaphoren.

Um Datenstrukturen zu Synchronisieren werden momentan ausschließlich Semaphoren eingesetzt. Falls irgendwann Mehrkern-Architekturen vom FHDWOS unterstützt werden, können auch Spinlocks für weitere Anwendungsfälle eingesetzt werden.

Bei der Erstellung einer Semaphore muss klar sein, von welchen Levels aus auf die Semaphore zugegriffen wird. Das höchste Level, dass die Semaphore nutzt wird in dieser gespeichert. Falls nur Threads auf Passive-Level die Semaphore benutzen muss das Level der Semaphore auf ein Level gesetzt werden, welches nicht verdrängt werden kann.

Bisher wurden für die Absicherung der kritischen Abschnitte innerhalb der Semaphoren sogenannte Mutex-Elemente in Form von Spinlocks eingesetzt. Eine solche Absicherung eines kritischen Bereiches soll jetzt durch das Anheben/-Absenken des Ausführungslevels geschehen. Beim betreten eines kritischen Abschnittes wird das Level auf das höchste Level, dass die Semaphore benutzt

angehoben. Dabei werden die Interrupts auf dem Weg maskiert, sodass keine Interrupts des gleichen, oder eines niedrigeren Levels, die Anweisungen innerhalb der Semaphore unterbrechen können. Nachdem die Semaphore alle kritischen Anweisungen ausgeführt hat, wird das Level auf das ursprüngliche Level herab gesenkt. Hierbei werden die Interrupts wieder freigegeben.

11.2.3 Preemptives Multitasking

Da preemptives Multitasking umgesetzt werden soll, muss ein Mechanismus entworfen werden, der unterbrechbare reentrante Tasks unterbricht und ein Scheduling veranlasst. Da dieser Mechanismus zyklisch ablaufen soll, wird ein Timer benötigt, der diesen Zyklus anstößt. Es bietet sich an, das preemptive Multitasking in Abhängigkeit zu einem Levelabstieg von einem nicht verdrängbaren zu einem verdrängbaren Level durchzuführen, da nur verdrängbarer Code durch den Scheduler beeinflusst werden kann.

11.2.4 Tasklets

Die Behandlung von Hardware-Interrupts wird in modernen Betriebssystemen wie Microsoft Windows oder Linux in zwei Teile aufgeteilt. In einen Bereich, der den eigentlichen Interrupt bildet (ISR) und eine sogenannte *untere Hälfte* (bottom half Konzept). Möglichst wenig Code steht in der eigentlichen Interrupt Service Routine (ISR). So können die Interrupts schnell abgearbeitet werden und möglichst viele Interrupts insgesamt bearbeitet werden. Die ISR kommuniziert beispielsweise nur direkt mit der Hardware und stellt die Daten der unteren Hälfte zur Verfügung. Diese behandelt den Rest des Interrupt im Dispatch- oder höheren Level um nicht von Tasks unterbrochen werden zu können. Von daher muss hier genau auf Sperren geachtet werden.

Unter Linux werden diese Instanzen ab der Version 2.3 Tasklets bezeichnet. Vor der Version 2.3 des Linux Kernels wurden Tasklets als Bottom-Half bezeichnet und wurden von der ISR des Interrupts immer wieder neu in eine Warteschlange hinzugefügt. Tasklets verbleiben im Gegensatz zu Bottom-Halfs im Speicher und werden nur aktiviert wenn sie eine Aufgabe erhalten. Ein Tasklet kann mehrmals vor seiner Abarbeitung aktiviert werden, wird aber nur einmal ausgeführt. Dieses beschleunigt die komplette Behandlung von Interrupts.

Als Beispiel kann hier das Drücken einer Tastaturtaste genannt werden. Bei einem Tastendruck löst die Tastatur einen Hardware-Interrupt aus. Die dafür geschriebene ISR holt den Scancode der gedrückten Taste aus dem Port und

legt diesen in einem Puffer ab, aktiviert das dazu gehörige Tasklet und aktiviert die normale Interrupt Behandlung, damit weitere Interrupts behandeln werden können. Von daher ist dieser Teil schnell abgearbeitet. Das Tasklet sorgt für die weitere Behandlung, wie in diesem Fall für die Umsetzung des Scancodes in den dazugehörigen Keycode.

11.3 Entwurf

11.3.1 Ausführungslevel

Grundsätzlich wird zwischen zwei verschiedenen Levels (Passive und Dispatch) und je einen für jeden *Interrupt Request (IRQ)* unterschieden. Somit existieren insgesamt 17 Level.

Eine Übersicht über die existierenden Ausführungslevel befindet sich in Abbildung 11.3.

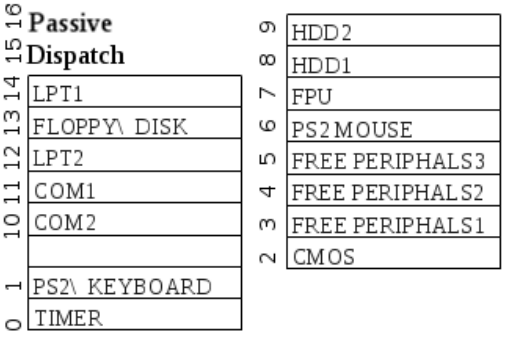


Abbildung 11.3: Übersicht über die verschiedenen Ausführungslevel

Interrupt Request Level

Der Interruptkontext ist der Zeitraum, indem der Interruptcontroller keine weiteren Interrupts sendet. Dieser Zeitraum sollte so kurz wie möglich sein, damit keine weiteren eintreffenden Interrupts verloren gehen und das System nicht zulange unterbrochen wird. Um dieses Kritierium zu erfüllen, wird der *Interrupt Request Level (IRQL)* eingeführt. Er orientiert sich an den Interrupt Prioritäten, womit genau 15 Level entstehen. Nachfolgend sind die möglichen IRQ Level mit aufsteigender Priorität dargestellt:

- IRQL_LPT1
- IRQL_FLOPPY_DISK

- IRQL_LPT2
- IRQL_COM1
- IRQL_COM2
- IRQL_HDD2
- IRQL_HDD1
- IRQL_FPU
- IRQL_PS2_MOUSE
- IRQL_FREE_PERIPHALS3
- IRQL_FREE_PERIPHALS2
- IRQL_FREE_PERIPHALS1
- IRQL_CMOS
- IRQL_PS2_KEYBOARD
- IRQL_TIMER

In jedem IRQ-Level werden Interrupts, die unterhalb des aktuellen Levels liegen maskiert, sodass diese die Ausführung des IRQs nicht unterbrechen können. So werden die in dieser Zeit eintreffenden Interrupts im Interrupt Controller aufbewahrt und weiter gegeben, sobald sie demaskiert werden.

Passive-Level

Das niedrigste Level ist das sogenannte Passive-Level. In diesem Level laufen die normalen Threads ab. Diese Threads können jederzeit unterbrochen und verdrängt werden und ihr Code ist reentrant, d.h. er kann nach dem Unterbrechen (auch von einem anderen Thread) wieder durchlaufen werden. Tritt im Passive-Level ein Interrupt auf, so wird nach dessen Ausführung der Scheduler aufgerufen, um einen Taskwechsel durchzuführen.

Dispatch-Level

Das nächst höhere Level ist das Dispatch-Level. Auf diesem Level läuft der Scheduler. Außerdem existiert hier eine "Taskletqueue". In dieser Queue werden die Verarbeitungen von IRQs durchgeführt. Dadurch wird die Zeit in der eine Ausführung auf einem IRQ-Level stattfindet, minimiert. Der Ausführungen im Dispatch-Level kann vom Timer-Interrupt unterbrochen werden, dieser springt

jedoch lediglich wieder zurück zur aktuellen Ausführung und veranlasst kein Scheduling. Programmcode auf dem Dispatch-Level kann so zwar unterbrochen werden, ist jedoch nicht reentrant. Des weiteren können Threads auf Dispatch-Level nicht verdrängt werden.

Klassendiagramm

In Abbildung 11.4 ist ein Klassendiagramm dargestellt, dass die Levelstruktur wiedergibt.

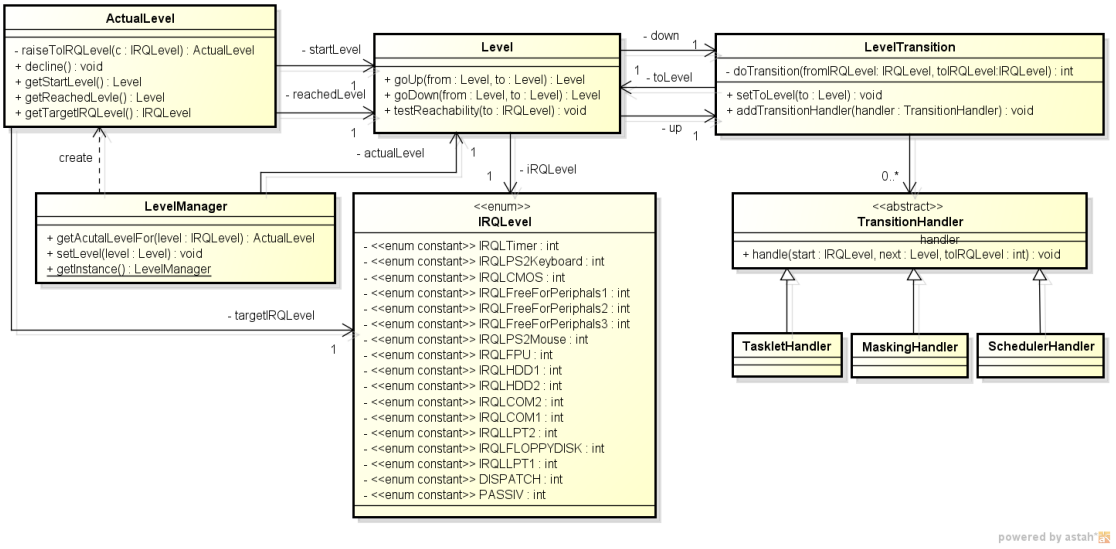


Abbildung 11.4: Klassendiagramm der Ausführungslevel

Das Enum *IRQLevel* steht für die verschiedenen Ausführungslevel die dem Programmierer zur Verfügung stehen.

Ein Level kapselt einen IRQLevel als Objekt und bietet Operationen an, mit denen entlang der Level traversiert werden kann. Die gesamte Modellierung der Ausführungslevel wird durch die Verkettung von Levels mit LevelTransitionen durchgeführt. An den LevelTransitionen können TransitionHandler registriert werden, durch die Code bei Levelübergängen ausgeführt werden kann.

So kann bei einer Transition von dem IRQLevel IRQL_PS_KEYBOARD zu IRQL_TIMER der Keyboardinterrupt maskiert werden. Ein Wechsel von IRQL_DISPATCH zu IRQL_PASSIV führt zum Aufruf des Schedulers, falls der Abstieg bei Level IRQL_TIMER begonnen hat. In jedem Fall werden bei diesem Wechsel die Tasklets durch den TaskletHandler ausgeführt.

Der LevelManager verwaltet das aktuelle global gültige Level und stellt Operationen zur Verfügung mit denen das aktuelle Level gesetzt werden kann. Desweiteren bietet er die Operation getActualLevelFor mit dem ein Objekt zur Klasse

ActualLevel zurückgeliefert wird, die das aktuelle Level kapselt und den Levelauf- und Abstieg verwaltet.

Beim Aufruf des Konstruktors der Klasse ActualLevel wird das globale Level auf den übergebenen Level gehoben. Das ursprüngliche Level und bei jedem Schritt das aktuelle Level werden in dem ActualLevel-Objekt zwischengespeichert. Dies ist z.B. dafür nötig, dass das ActualLevel den internen Level beim Aufruf der Operation decline wieder auf den Startlevel absenkt.

Erhöhen der Level

In Abbildung 11.5 ist ein Sequenzlevel dargestellt, dass aufzeigt, wie die Levelstruktur auf einen Timerinterrupt reagiert.

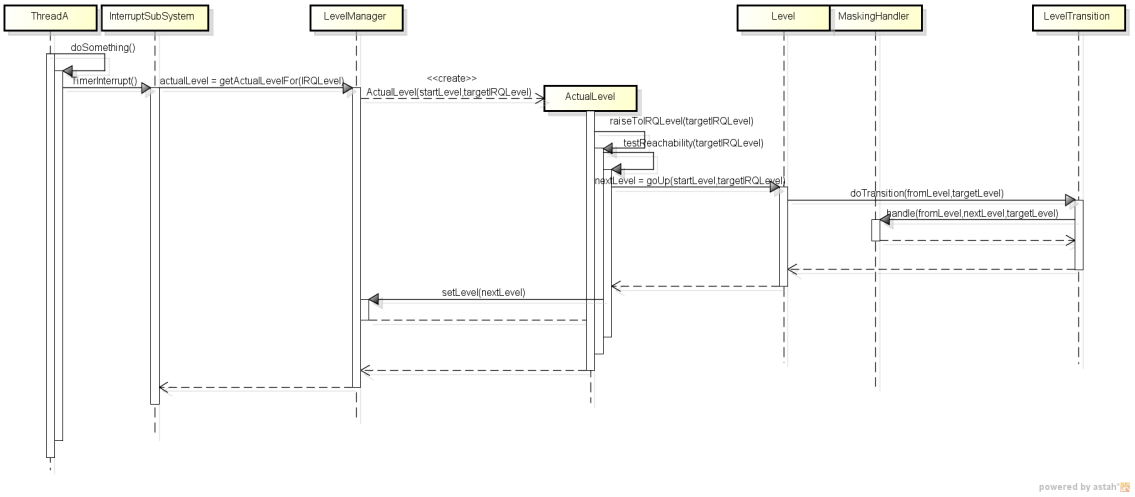


Abbildung 11.5: Klassendiagramm der Ausführungslevel

Zuerst wird der aktuell ausgeführte Code von dem Timer-Interrupt unterbrochen. Durch diesen wird in das InterruptSubSystem gesprungen, dass den aktuellen Level vom LevelManager anfordert. Dieser erzeugt ein ActualLevelObject und übergibt den angeforderten Level.

Beim Aufruf des Konstruktors von ActualLevel wird gleich eine Erhöhung des Level auf das übergebene Ziellevel durchgeführt. Dazu wird zuerst geprüft, ob das übergeben Level erreicht werden kann. Danach wird das Level schrittweise erhöht, wobei bei jeder Erhöhung die LevelTransition ihre entsprechenden Handler ausführt. Bei jedem erreichten höherer Level wird dieses Level als aktuelles Level im LevelManager gesetzt. Ist das entsprechende Level erreicht, wird an das InterruptSubSystem zurückgegeben.

11.3.2 Synchronisation

Wie in der Analyse beschrieben muss beim Erstellen der Semaphore klar sein, von welchen Ausführungsleveln aus auf sie zugegriffen wird. Die Semaphore bietet zwei verschiedene Konstruktoren. Einen, der ein IRQLevel zusätzlich zum Startwert der Semaphore benötigt (*Semaphore::Semaphore(unit32_t permits, IRQLevel level)*) und einen der wie bisher nur einen Startwert benötigt (*Semaphore::Semaphore(unit32_t permits)*).

Der erste Konstruktor muss eingesetzt werden, wenn von verschiedenen Levels aus auf die Semaphore zugegriffen wird. Das höchste Ausführungslevel muss bei der Erstellung im Konstruktor angegeben werden (*level*). Der zweite Konstruktor darf nur eingesetzt werden, wenn normale Threads auf Passive-Level auf die Semaphore zugreifen. Der zweite Konstruktor setzt die interne Variable (die das höchste Level speichert) auf Dispatch-Level.

Die Semaphore bietet außerdem die schon bekannten Operationen *up()* und *down()*. Innerhalb dieser Operationen werden die beiden privaten Operationen *ActualLevel enterCriticalArea()* und *void leaveCriticalArea(ActualLevel ac)* eingesetzt.

Diese beiden Operationen, die den kritischen Bereich sichern, sind dafür zuständig das Level anzuheben (dabei wird ein ActualLevel erzeugt) und dieses später wieder abzusenken

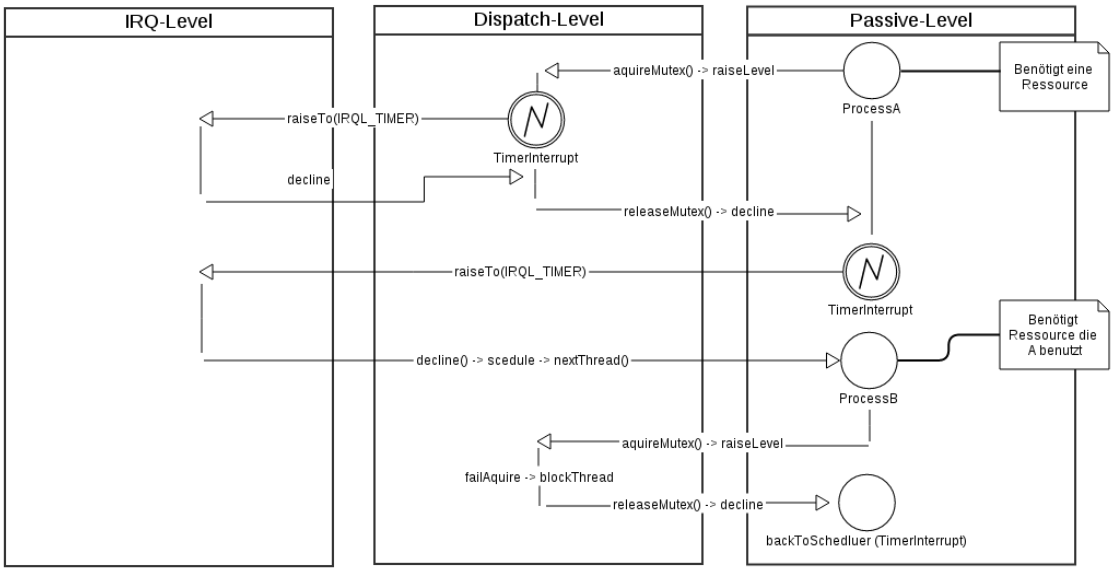


Abbildung 11.6: Synchronisations Beispiel

11.3.3 Preemptives Multitasking

Um preemptives Multitasking umzusetzen wird der "Programmable Interrupt Timer"(kurz PIT) verwendet. Dieser löst nach ca. 55 ms einen Interrupt aus. Nach jedem fünften Auftreten dieses Interrupts wechselt der Scheduler den Task. Der Taskwechsel geschieht, wenn alle Tasklets ausgeführt wurden und der Level in den Passive-Level abgesenkt wird. Der "TransitionHandler"für diesen Übergang greift auf den Scheduler zu, der den Wechsel durchführt.

11.3.4 Tasklets

Die Behandlung der Hardware-Interrupts wurde in Anlehnung an modernen Betriebssysteme entworfen und implementiert. Es wurden Tasklets als *untere Hälfte* der ISRs eingeführt. Tasklets werden von den Gerätetreibern erzeugt und den Interrupt-Handlern zur Verfügung gestellt.

Ein Tasklet kann drei verschiedene Zuständen annehmen: WORKLESS, HASWORK und TOKILL.

Tasklets im WORKLESS Zustand werden beim Wechsel zum Passive-Level nicht ausgeführt. Tasklets im HASWORK Zustand werden beim Wechsel zum Passive-Level ausgeführt. Tasklets im TOKILL Zustand werden vom TaskletManager aus seiner Liste entfernt und gelöscht. Aus diesem Zustand kann in keinen anderen Zustand gewechselt werden. Dies dient dem Schutz der Tasklets vor der Löschung des Tasklets, während die work() Operation ausgeführt wird.

Die work() Operation des Tasklets gibt den Folgezustand des Tasklets zurück. Somit können Tasklets bestimmen, ob sie beim nächsten Durchlauf aller Tasklets nochmal ausgeführt werden sollen. Dies dient dazu die Arbeit der Tasklets in mehrere Schritte aufteilen zu können um Tasks öfter auszuführen.

Für die Verwaltung und Ausführung der Tasklets wird ein Manager (TaskletManager) eingesetzt. Bei einem SMP(Symmetrischen Multiprozessorsystem) muss pro Kern ein Manager erstellt werden. Der Manager enthält die work() Operation die sämtliche Tasklets ausführt die im Zustand HASWORK sind, die in der Liste des Managers enthalten ist. Ausgeführt wird diese vom TransitionHandler beim Wechsel von dem Dispatch- zu dem Passive-Level. Von daher werden die Tasklets im Dispatch-Level ausgeführt.

Die Abbildung 11.7 zeigt das Klassenmodel der Tasklets und des TaskletManagers.

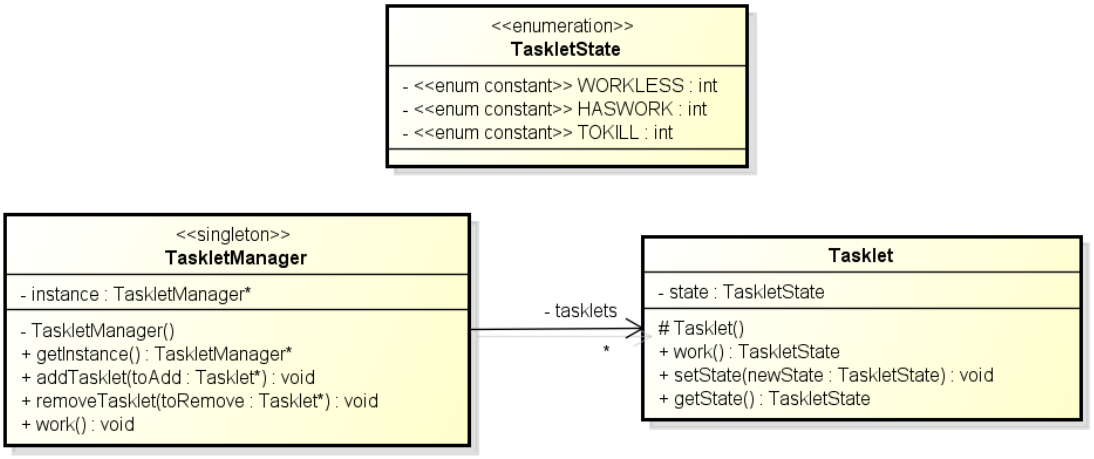


Abbildung 11.7: Tasklets

11.4 Fazit

In diesem Projekt wurde das bestehende Scheduling-System von kooperativ auf preemptiv umgestellt. Dazu wurde analysiert, wie gängige Betriebssysteme dieses Multitasking realisieren. Es wurden verschiedene Ausführungslevel eingeführt, die unter anderem signalisieren ob der Code unterbrechbar ist oder nicht. Der Scheduler wurde angepasst und das Interrupt-System umgestellt. Die Interrupts werden als Anfragen betrachtet und ein Interrupt-Handler arbeitet diese so schnell wie möglich ab, um weitere Interrupts nicht zu blockieren. Für die weitere Ausführung wurde ein *bottem half* Konzept implementiert. Diese sogenannten Tasklets arbeiten auf Dispatch-Level. Weiterhin wurde eine Möglichkeit geschaffen um zwischen verschiedenen Levels zu kommunizieren. Zuletzt wurden bisherige Treiberthreads (Keycode und Charlayout) auf Tasklets umgestellt.

Teil IV

Anwendungen

12 Infrastruktur

12.1 Einleitung

Diese Teilaufgabe (Tasks III) für das FHDW-OS beschäftigt sich mit eigenständigen Programmen für das FHDW-OS. Es sollen verschiedene Aufgaben erledigt werden, die in diesem Abschnitt kurz erläutert werden:

12.1.1 Toolchain Anpassung

Die aktuelle Toolchain (der Build-Prozess) soll angepasst werden, sodass eigene Programm separat entwickelt werden können. In diesem Zusammenhang müssen auch Schnittstellen zu den Kernel-Funktionen geschaffen werden.

12.1.2 Reimplementation der bisherigen Treiber

Alle bisherigen Treiber sollen in der neuen Struktur als Programme reimplementiert werden und an die neuen Kommunikationsmittel der letzten Aufgaben – genauer: an die Interprozesskommunikation – angepasst werden.

12.1.3 Programmlader

Es soll ein einfacher Programmlader entwickelt werden, der dafür zuständig ist ein Programm in den Speicher einzulesen und die nötige Umgebung (Heap, etc.) zu erstellen. Dabei soll auch eine Argumentübergabe in Form einer beliebigen Zeichenkette unterstützt werden.

12.2 Analyse

In diesem Kapitel werden die aktuellen Gegebenheiten des FHDW-OS analysiert, die für die Bearbeitung der einzelnen Aufgaben wichtig sind:

Bisher existiert im FHDW-OS ein großes Eclipse-Projekt für den gesamten Code. Der Build-Prozess basiert auf einem Hauptmakefile, dass sowohl das kernel-Projekt, als auch die loader-Projekte (bootloader/osloader) miteinbezieht und diese zu den einzelnen Images (.img Dateien) zusammen „gießt“. Von diesen Images kann Bochs dann das Betriebssystem starten. Zusätzlich zu dem Hauptmakefile existieren in den anderen Projekten (kernel, loader) jeweils eigene Makefiles, die die Einzelheiten des Bauens regeln.

Um jetzt eigenständige Programme entwickeln und einbinden zu können sollte es ein Projekt parallel zum kernel-Projekt geben, dass alle einzelnen Programme miteinbezieht. In diesem Projekt (im folgenden apps genannt) könnten dann auch Bibliotheken (beispielsweise zum Zugriff auf kernel-Funktionen) für alle Programm-Entwickler bereitstehen.

Wichtig ist, dass in der Anwendung keine im Kernel bereitgestellten Methoden verwendet werden können.

Möglichkeiten zur Interprozesskommunikation stehen in dem FHDW OS allerdings bereits zur Verfügung. Diese können auch für die Kommunikation zwischen Kernel und Anwendung genutzt werden.

12.3 Entwurf

Unter Berücksichtigung der Analyseergebnisse soll ein neues Verzeichnis „apps“ angelegt werden, in dem die einzelnen Programme abgelegt werden können. In jedem dieser Programm-Ordner wird ein Makefile benötigt, das den Namen und Pfad des jeweiligen Programmes definiert. Die fertig gebauten Programme werden mithilfe des Hauptmakefiles in das Gesamtsystem integriert.

12.3.1 Programmlader

Um die *apps* während der Laufzeit auszuführen (und zu laden) wird ein Programmlader benötigt. Dieser hat verschiedene Aufgaben:

- ein Prozessobjekt für jede *app* erzeugen (inkl. Stack(im User-Adressraum), etc.)
- linearen Speicher für den Programmcode über den zum Prozess gehörenden LinearAddressSpaceManager reservieren
- kopieren des Codes in diesen Adressraum

- initialen Thread erzeugen und von dort den Einsprungspunkt der Anwendung laden

Ein erster Entwurf ist in **??** zu sehen.

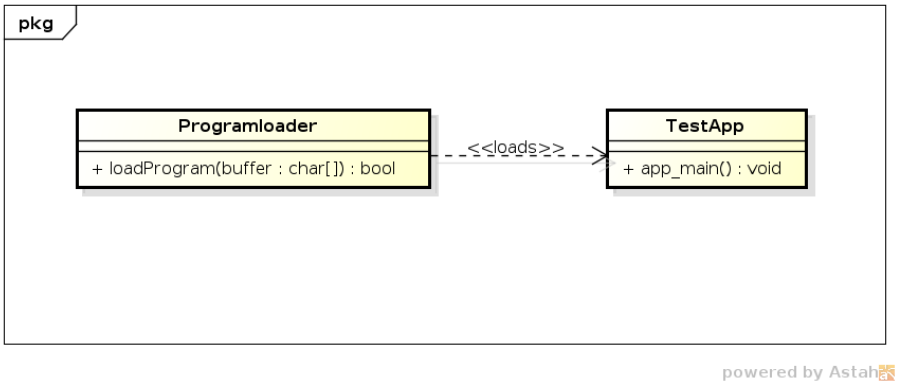


Abbildung 12.1: Klasse ProgramLoader

Der ProgramLoader besitzt eine Methode, die einen Puffer übergeben bekommt und den dort enthaltenen Code lädt.

12.3.2 Bibliothek

Unterhalb des *apps*-Ordners soll eine Bibliothek angelegt werden, die mithilfe der Interprozesskommunikation den Zugriff zu kernel-Methoden regelt. Dazu werden zu alle benötigten Klassen Proxy/Wrapper-Klassen angelegt. Diese Wrapper ermöglichen mithilfe der Kommandos der Interprozesskommunikation dann einen Zugriff auf die gewünschten Methoden.

Die angestrebten Klassen sind in Abbildung 12.2 zu sehen.

Hier ist gut zu erkennen, dass viele Klassen Requests nutzen, um mit dem OS Kernel zu kommunizieren. Für diese *Requests* muss es entsprechende *Command-Klassen* geben. Diese sind stets gleichnamig bis auf das Request durch Command ersetzt ist. Alle diese Commands haben eine einzige execute()-Methode. Da ein grafisches Modell hier kein neues Wissen aufzeigen kann, wird an dieser Stelle darauf verzichtet.

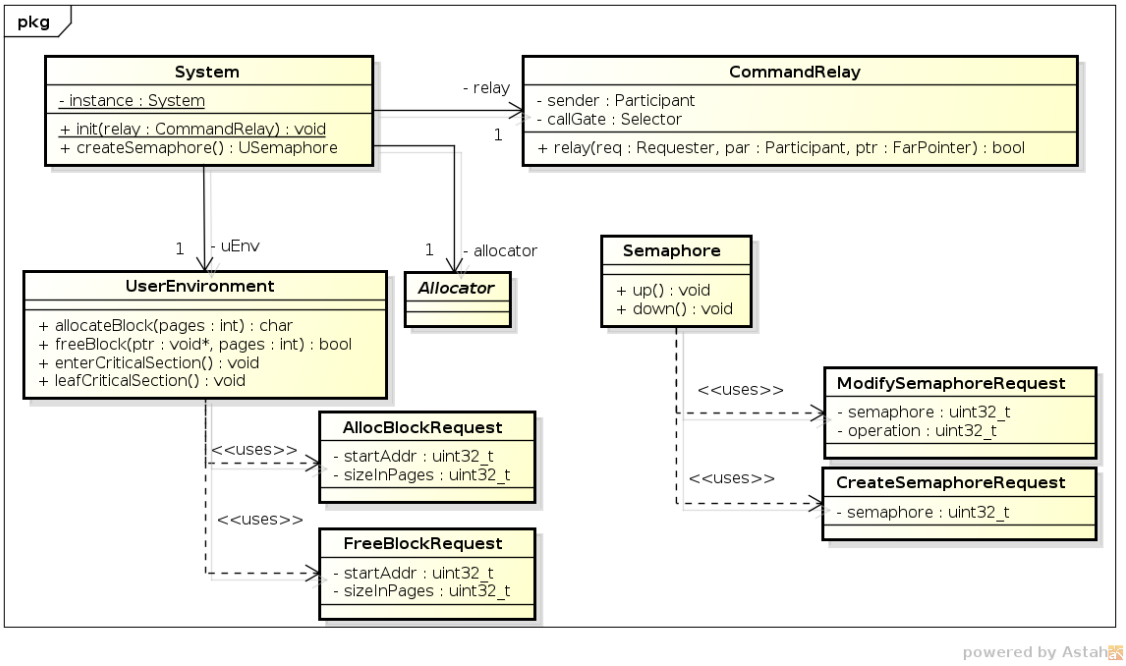


Abbildung 12.2: Entwurf der Bibliothek

12.4 Implementierung

12.4.1 Programmlader

Für den Programmlader wurde eine Klasse *ProgramLoader* im Namensraum *task* angelegt, die es ermöglicht ein Programm zu laden.

Dazu wird die Methode

```
bool loadProgram(void* ptr, uint32_t sizeInByte);
```

eingesetzt. Im Entwurf war vorgesehen, einen Puffer zu übergeben. Es hat sich allerdings herausgestellt, dass dies nicht nötig ist. Lediglich die Startadresse sowie die Speichergröße sind ausreichend um ein Programm laden zu können. Dies hat den Vorteil, dass ein Aufrufer nicht erst einen Puffer erstellen, bzw. seinen Code in diesen Kopieren muss.

Wie im Entwurf besprochen werden hier außerdem verschiedene Vorbereitungen durchgeführt. Unter anderem wird ein *UserModeThread* für jeden *Process* erstellt, dessen run-Methode dann das Programm ausführt.

12.4.2 Bibliothek

Wir haben in der Bibliothek unter (*apps/library/*) verschiedene Wrapper-Klassen implementiert, die den Zugriff zum kernel managen.

Die dort implementierten Klassen liegen im Namensraum *lib*, bzw. *lib::runtime* im Falle der Klasse UserEnvironment.

Das gesamte System wird über die Klasse *System* initialisiert. Dies übernimmt die Methode

```
static void init(ipc::CommandRelay &relay);
```

welche als erstes in jedem Prozess aufgerufen werden muss. Andernfalls kann ein korrektes Funktionieren nicht gewährleistet werden.

Es erscheint daher sinnvoll, zuerst diese Methode aufzurufen und danach erst die eigentliche main() der Anwendung. Beim Versuch dies umzusetzen, stellte sich allerdings heraus, dass dafür bereits zum Zeitpunkt der Programmierung feststehen müsste, welche Adresse main() und init() haben. Es ist allerdings lediglich bekannt, an welcher Speicherstelle die main() liegt, nämlich an 0x100000 des linearen Adressraums.

Das übergeben CommandRelay dient der Kommunikation mit dem OS Kernel.

Neben den in **??** aufgeführten *Request* und *Command* Klassen wurde außerdem ein Request/Command Paar für das Laden eines Programms geschrieben. Dies nutzt ebenfalls Startadresse und Größe um ein Laden durchzuführen.

Zur Zeit wird dies noch zu einer **Pagefault Exception** führen, da höchst wahrscheinlich die als Startadresse angegeben Adresse nicht im Kernel-Bereich gemappt ist. Hier besteht noch Handlungsbedarf!

Des Weiteren wurden einige Klassen aus dem Kernelbereich in eine Art Shared Library ausgegliedert. Diese liegen parallel zum Apps- und Kernelverzeichnis in einem Ordner *library*. Durch Anpassung des Makefiles ist es so möglich, Klassen in diesem Ordner sowohl in die Binärdatei des Kernels auch in die jeder Anwendung zu kompilieren. Auf diese Weise können Objekte dieser Klassen sowohl im Kernel als auch im Anwendungsbereich genutzt werden. Dies ist gerade für die Klassen um den Allokator, sowie für die bereits vorhanden Typen und Listenstrukturen sinnvoll.

12.5 Test

Das Testen des umgestellten Built-Prozesses geschah manuell. Es wurden die Dateigrößen der einzelnen Binärdateien betrachtet und so ermittelt, ob beim Built etwas produziert wurde. Die Funktionsfähigkeit wurde überprüft, indem nach einem Clean-Built das Betriebssystem gestartet wurde und die vorhandenen Testfälle ausgeführt wurden.

Das Testscenario für die Mechanismen der User-Kernel-Kommunikation, sowie der Initialisierung der Speicherstrukturen und notwendigen Klassen im Anwendungsbereich sah einen großen, manuellen Test vor. Hier wurde lediglich eine simple Anwendung geschrieben, die einen Text auf der Konsole ausgibt. Dieser Vorgang nutzt alle implementierten Methoden und konnte so zur Verifikation verwendet werden.

12.6 Fazit

Mithilfe des entwickelten Programmladers können nun vom Betriebssystemkernel unabhängig erstellte Programme geladen werden. Dazu wurden notwendige Änderungen am Build-Prozess vorgenommen.

Des Weiteren wurden auf Basis des zuvor entwickelten Mechanismus der Interprozesskommunikation einige *Commands* implementiert, die nun von anderen Programmteilen verwendet werden können.

Die Treiber wurden in diesem Teilprojekt jedoch aus Zeitnot nicht als eigenständige Programme entwickelt.

12.7 Ausblick

Da die Entwicklung geeigneter Zugriffsmethoden auf eine Festplatte parallel zu den in diesem Dokument beschriebenen Entwicklungen erfolgte, ist ein weiterer logischer Schritt das Laden eines Programms von der Festplatte zur Laufzeit des Betriebssystems.

Wie bereits im Abschnitt *Implementierung* erwähnt muss noch sichergestellt werden, dass die Startadressen bereits im Kernel-Bereich gemappt sind, bevor ein Programm über Request/Command geladen wird.

13 Kommandozeile

13.1 Analyse

Die Aufgabe ist es einen Kommandozeilen Interpreter zu entwickeln, der möglichst flexibel erweitert werden kann. Damit neue Funktionen des FHDW–OS aufgerufen werden können.

Die Aufgabe des Kommandozeilen Interpreter ist es einen Eingabe–Prompt darzustellen, um den Anwender mitzuteilen, dass nun ein Befehl eingegeben werden kann. Darauf kann der Anwender einen Befehl und weitere Parameter eingegeben werden. Der Befehl kann dann durch drücken von Return ausgeführt werden. Des Weiteren ist es notwendig, dass der Kommandozeilen Interpreter dem Benutzer eine Rückmeldung gibt, falls kein passender Befehl gefunden wurde.

13.2 Entwurf

Das Command Line Interface (CLI) besteht im Wesentlichen aus drei Teilen. Die Tastatureingaben werden von einem Scanner eingelesen und anhand von Seperatoren (z. B. Leerzeichen) in eine Folge von Elementen eingeteilt. Diese werden an einen Parser übergeben, der die Eingaben interpretiert und daraus einen ausführbaren Befehl erstellt. Dieser wird schließlich ausgeführt und ggf. eine Bestätigung auf die Konsole geschrieben. Dieser Vorgang ist in **??** dargestellt.

13.2.1 Scanner

Die Aufgabe des Scanners ist es die einzelnen Zeichen die der Benutzer eingibt zu analysieren und in sinnvolle Gruppen einzuordnen. Um dies umzusetzen wird für den Scanner ein Zustandsmuster benutzt. Nachfolgend wird zunächst näher auf die Gruppierung eingegangen. Darauf wird genauer auf das Zustandsmuster

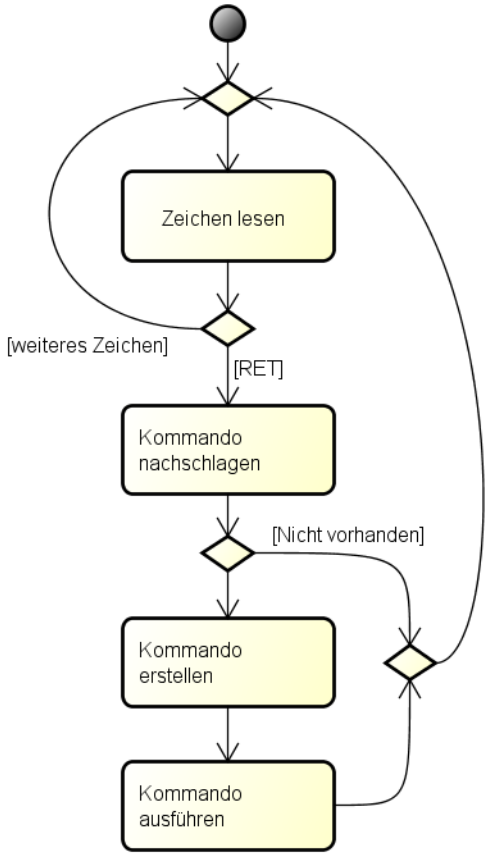


Abbildung 13.1: Ablauf des CLI

des Scanners eingegangen. Die in diesem Abschnitt erwähnten Klassen sind in **??** dargestellt.

Scanner Elemente

Die sinnvolle Gruppierung werden durch die FosCliElemente repräsentiert. Sinnvolle Gruppierungen sind zum Beispiel:

- Wörter die durch Leerzeichen getrennt sind. Diese können sowohl zu einem das Kommando sein als auch Parameter für ein Kommando.
- Sonderzeichen, die die Ausführung eines Befehls modifizieren wie zum Beispiel die Standardausgabe umzuleiten.

Von den Möglichkeiten der Gruppierung wurde nur die Leerzeichen getrennte Wörter umgesetzt. Die durch die Spezialisierung FosCliStringElement repräsentiert werden. Außerdem wurde für die Spezialisierungshierarchie ein Visitor-Pattern implementiert, damit später im Parser die einzelnen Elemente typisiert werden können.

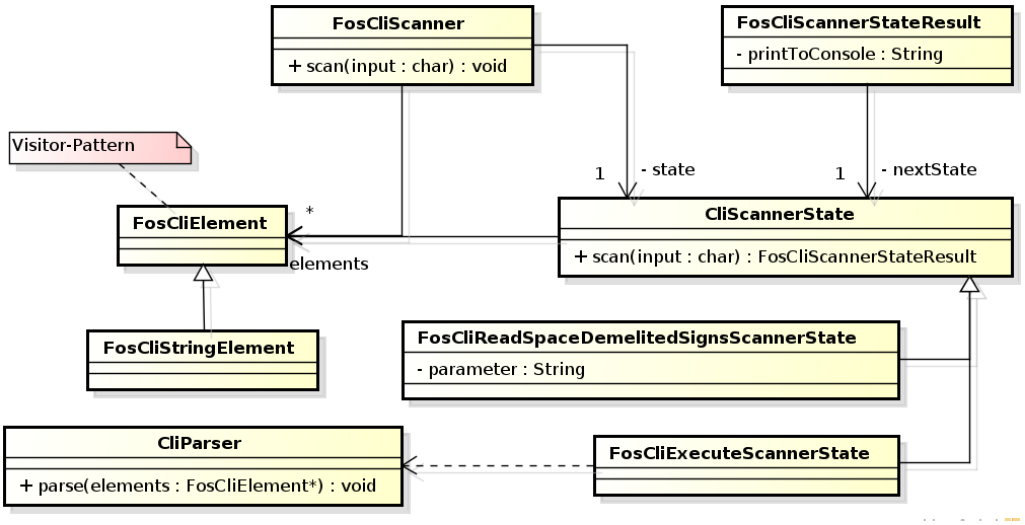


Abbildung 13.2: Klassendiagramm Scanner

Scanner Zustandmuster

Der Scanner, welcher durch die Klasse FosCliScanner repräsentiert wird, verarbeitet jedes Zeichen einzeln und ordnet diese einer bestimmten Gruppierung zu und fügt falls nötig einzelne Zeichen zusammen. So wie es für die Leerzeichen getrennten Wörter nötig ist. Damit möglichst flexibel neue Gruppierungen hinzugefügt werden können wurde im Scanner ein Zustandsmuster implementiert. Derzeit existieren zwei Zustände:

1. Der erste Zustand, mit den Namen FosCliReadSpaceDelimitedSignsScannerStat, ist für die Leerzeichen getrennten Wörter zuständig. Er fügt die einzelnen Zeichen zu einem String zusammen. Bei einem Leerzeichen wird ein neuer String begonnen.

2. Der zweite Zustand, mit dem Namen FosCliExecuteScannerState, sendet alle Elemente, die bis zur Eingabe eines Returns gesammelt wurden, an den Parser.

Die Elemente, die in den einzelnen Zuständen erzeugt werden, werden in einer Liste, die durch den FosCliScanner verwaltet wird, gesammelt. Bei jedem gescannten Zeichen kann eine beliebig Lange Zeichenkette zurückgegeben werden, die auf dem Terminal ausgegeben wird. Somit kann zum Beispiel nach dem senden der Elemente zum Parser ein neuer Prompt ausgegeben werden.

13.2.2 Parser

Der Parser koordiniert die Interpretation der erhaltenen CLI-Elemente. Dabei wird das erste Element stets als Befehl interpretiert. Dieser Befehl wird in einem

Verzeichnis nachgeschlagen und die weiteren CLI-Elemente als Parameter zur Erstellung des Befehls weitergereicht. Das Klassenmodell in Abbildung 13.3 zeigt die dafür notwendigen Klassen.

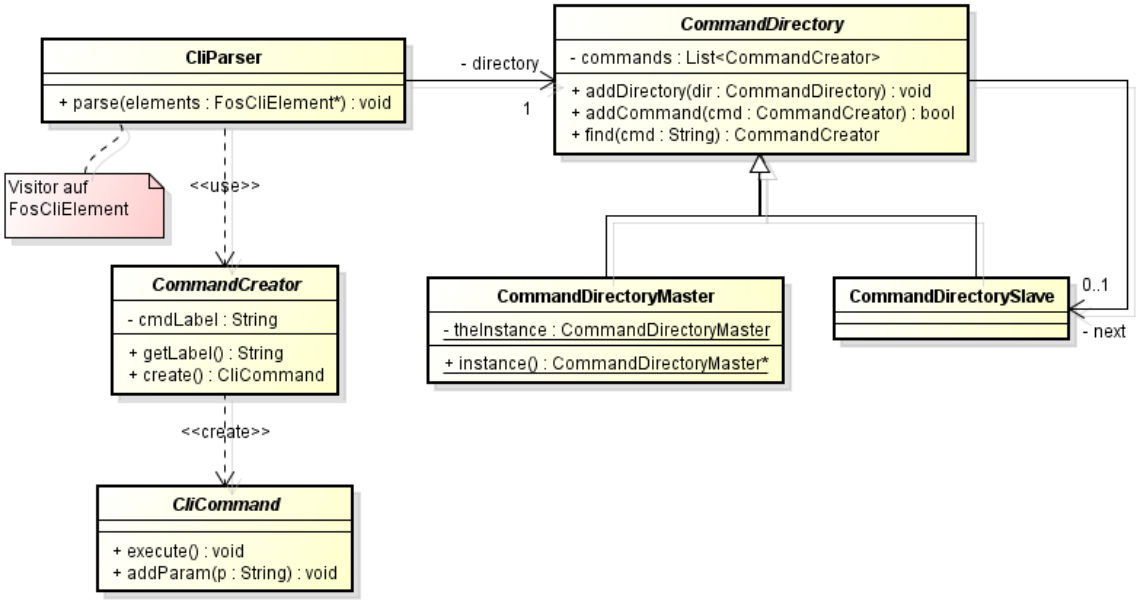


Abbildung 13.3: Parser für CLI

Als Verzeichnis für Befehle dient die Klasse `CommandDirectory`, die aus einem Master- und mehreren potentiellen Slave-Verzeichnissen zusammengesetzt ist. Das Master-Verzeichnis dient als Einstiegspunkt in die Befehlssuche und leitet Anfragen ggf. weiter an seine Unterverzeichnisse. Wird in diese Struktur ein neues Verzeichnis hinzugefügt, so wird dieses bei der Suche nach einem Befehl erst nach den bereits vorhandenen Verzeichnissen berücksichtigt. Dies geschieht aus Sicherheitsgründen, damit bereits definierte Befehle nicht überschrieben werden.

Wurde ein Befehl gefunden, so wird der passende `CommandCreator` an den Parser zurückgegeben. Dieser erstellt einen ausführbaren Befehl (`CliCommand`), an welchen die Parameter übergeben werden. Der erstellte Befehl wird schließlich ausgeführt.

13.3 Implementierte Kommandos

Als Beispiele wurden bereits mehrere Kommandos implementiert. Folgende Befehle sind bereits verfügbar:

help Ermittelt die verfügbaren Befehle aus den Verzeichnissen und listet diese auf.

clear Leert die Konsolenausgabe.

version Keine Parameter, gibt die aktuelle Version des FHDW-OS aus.

echo Gibt die Parameter aus, getrennt durch Leerzeichen.

Time Gibt die Uptime des FHDW-OS (in Millisekunden) sowie die aktuelle Systemzeit (in Sekunden) aus.

Teil V

Gerätetreiber

14 Treiber-Framework

14.1 Was ist ein Treiber Framework

Die Aufgabe eines Treiber Frameworks für Gerätetreiber ist es, für die angeschlossenen Geräte verschiedene Treiber zur Verfügung zu stellen und diese zu verwalten. Ein Treiber ist in unserem Fall für die Vorbereitung eines Gerätes zuständig. Auch die Aufräumarbeiten, wenn ein Gerät wieder entfernt wird, werden von einem Treiber ausgeführt. Die einzelnen Funktionen eines Gerätes werden direkt über das Gerät angesprochen. Das Framework muss die Möglichkeit bieten verschiedene Treiber hinzuzufügen und auch wieder zu entfernen.

14.2 Treiberinitialisierung

Für die Initialisierung der Treiber müssen Abhängigkeiten erkannt und Berücksichtigt werden. Solche Abhängigkeiten können zum Beispiel zum Zeitgeber oder den Interrupts bestehen.
Zur Berücksichtigung solcher Abhängigkeiten bei der Initialisierung der Treiber gibt es unterschiedliche Vorgehensweisen, die im Folgenden beschrieben werden sollen. Im Anschluss werden diese bewertet und es wird erläutert, welcher dieser Mechanismen von uns verwendet wurde.

14.2.1 Mehrstufige Initialisierung

Der Initialisierung-Prozess der Gerätetreiber wird in mehrere Stufen unterteilt. Bei der mehrstufigen Initialisierung wird mit einem unbestimmten Treiber begonnen. Sofern dieser Abhängigkeiten zu noch nicht gestarteten Geräten hat, wird der entsprechende Treiber in die jeweils nächste Stufe eingereiht. Diese Strategie kann zu einer Endlosschleife bei der Initialisierung führen, wenn zum Beispiel zwei Treiber Abhängigkeiten zum jeweils anderen haben.

14.2.2 Bedarfsinitialisierung

Die Bedarfsinitialisierung lädt die Treiber nach Bedarf. Begonnen wird mit einem unbestimmten Treiber. Sollten für diesen Abhängigkeiten vorhanden sein, wird zunächst versucht eine dieser Abhängigkeiten durch Laden des entsprechenden Treibers aufzulösen. Auch bei dieser Vorgehensweise können gegensätzliche Abhängigkeiten zu einem Endlosschleife führen.

14.2.3 Gewählte Strategie

Für das FHDW OS wurde eine mehrstufige Initialisierung gewählt. Die Funktionsweise ist in Abbildung 14.1 visualisiert.

Zu Beginn liegt eine Liste mit zu initialisierenden Treibern vor. Aus dieser Liste wird der erste Treiber genommen, um ihn zu initialisieren. Sofern für diesen Treiber Abhängigkeiten vorliegen, wird der Treiber in die Liste der nächsten Stufe aufgenommen und dort initialisiert. Diese Strategie garantiert eine mehrstufige Initialisierung, bei der Treiber mit Abhängigkeiten solange in die nächste Stufe verschoben werden, bis für diese alle benötigen Treiber geladen sind.

Ein Wermutstropfen ist die Tatsche, dass dieser Mechanismus bei Zyklen in einen Endlosschleife läuft. Gegenseite Abhängigkeiten müssen im Vorfeld analysiert werden.

14.3 Aufbau

Den Aufbau des Frameworks zeigt Abbildung 14.2.

Das Framework sieht eine Trennung von Treiber und Gerät in die Klassen Driver und Device vor. Die Klasse Device repräsentiert dabei die Hardware und stellt die Funktionen für deren Verwendung bereit. Die Klasse Driver dient zum Abbilden der Abhängigkeiten zu anderen Geräten und der Initialisierung eines Gerätes.

Treiber werden Geräten über eine ClassID zugeordnet. Die IDs für jede Klasse sind in der Datei ClassConstants.h zu finden.

Die Klasse DriverManager ist sowohl für das Verwalten von Treibern als auch für das Initialisieren von Geräten zuständig. Die Methode „initializeDevices“ prüft für alle dem DriverManager bekannten Treiber die eingetragenen Abhängigkeiten und ruft für den Fall, dass alle Abhängigkeiten erfüllt sind, die Methode „check-

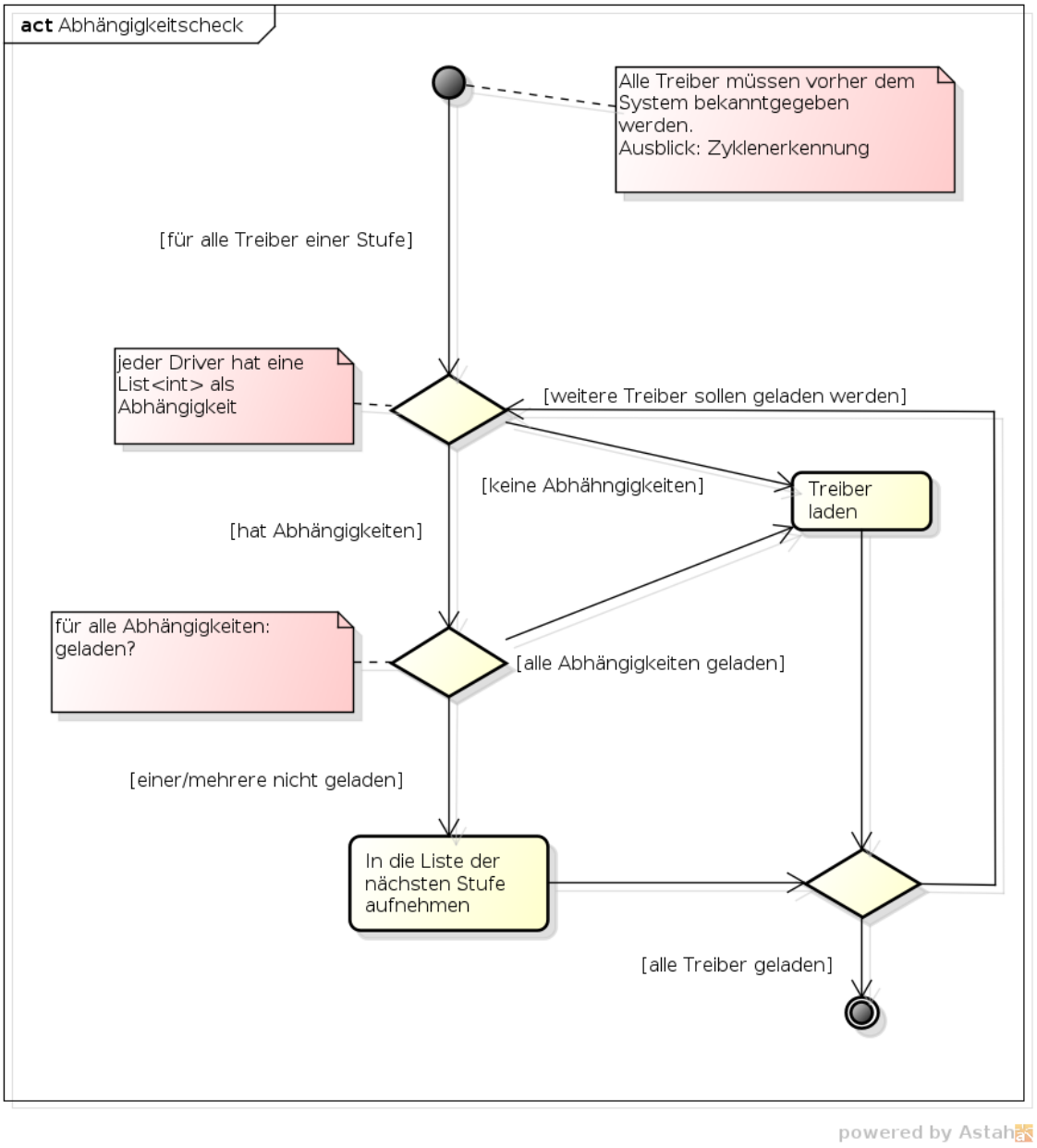


Abbildung 14.1: Ablauf der Geräteinitialisierung

Dev“ des Treibers auf. Hier muss die Initialisierung des zugehörigen Device durchgeführt werden.

Die Klasse DeviceManager verwaltet alle Geräte. Ein Gerät wird bereits beim Initiieren im Konstruktor der Klasse Device dem DeviceManager hinzugefügt. Über die „get“ Methode kann ein Device für eine bestimmte Device- Klasse erhalten werden. Dieses „get“ ist eine Template-Methode, die mit einer Klassenangabe annotiert werden kann.

```
1        T* get<T>(int DeviceClassID)
```

Die DeviceClassID wird zusätzlich benötigt, um ein Objekt der entsprechenden Klasse zu zuordnen.

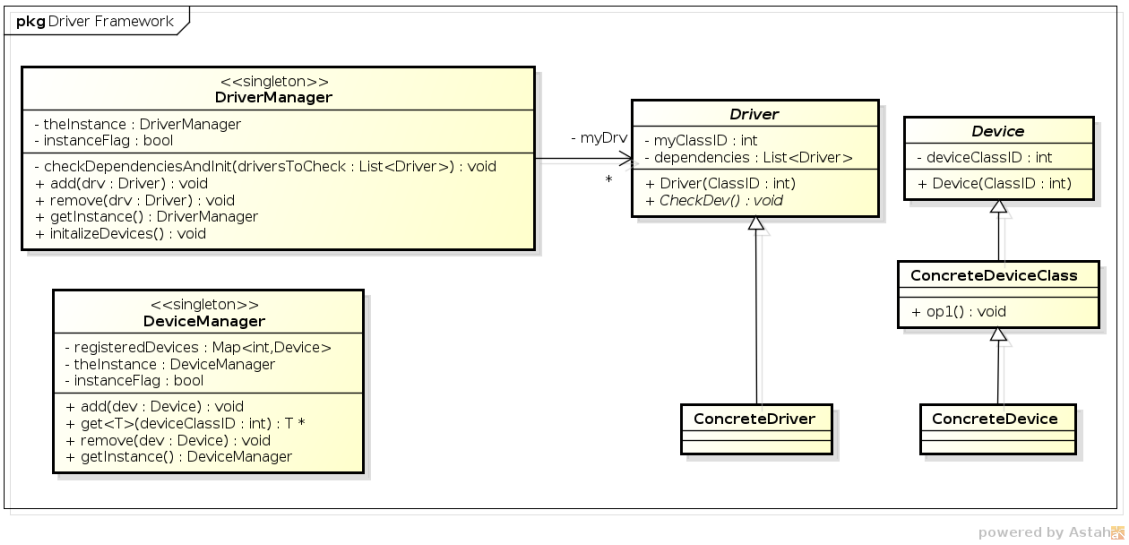


Abbildung 14.2: Aufbau des Treiberframeworks

Für jede konkrete Unterklasse von Device sollte es eine Device Klasse geben, die als Schnittstelle für diese Unterklasse fungiert.

14.4 Geräte Klassen

Um die Funktionalitäten verschiedener Klassen von Geräten zusammen zu fassen, wurden die sogenannten DeviceClasses eingeführt. Bei diesen Klassen handelt es sich um Schnittstellen, die die Spezifikation der einzelnen Geräte, die sie implementieren darstellen. Sie enthalten die Operationen, die alle Geräte der jeweiligen Klasse unterstützen. Somit wird ein Entwickler, der ein neues Gerät einer Klasse implementiert, gezwungen, bestimmte Methoden zu implementieren. Eingeführt wurden zwei Geräte Klassen. Die Grafik Klasse, die Operationen für Grafik Geräte spezifiziert und die Audio Klasse, die Operationen für Audio Geräte spezifiziert. Die Tabelle 14.1 zeigt die Operationen die in den beiden Klassen definiert wurden.

Für die ebenfalls in dieser Phase des Projektes implementierten Zeitgeber und Tastatur bzw. Eingabegeräte wurden keine Klassen implementiert.

Audio Klasse:	void on()
	void off()
	void setFrequency(int freq)
	void beep()
	void beep(int freq)
	void beep(int freq, int duration)
Grafik Klasse:	void write(char* c, int x, int y)
	void moveCursor(int x, int y)
	void moveCursorByOffset(int xOffset, int yOffset)
	VGAChar read(int x, int y)

Tabelle 14.1: Geräte Klassen und ihre Operationen

14.5 Verwendung

14.5.1 Implementierung von Treibern

Um das Treiber Framework zu nutzen, muss von einem Programmierer nicht viel getan werden. Es muss lediglich eine Unterklasse zu Driver und eine zu Device bzw. der DeviceClass Klasse erstellt werden.

Für jede konkrete Klasse von Device sollte eine abstrakte Klasse existieren. Gibt es noch keine, muss diese angelegt werden. Diese DeviceClass- Klassen spiegeln die Art des Treibers wider und bilden so ein Interface für andere Programme oder alternative Implementierungen. Hier sollten die wesentlichen Operationen definiert werden.

In allen Unterklassen von Driver und Device muss eine DeviceClassID für die Art des Gerätes vergeben werden. Diese ID wird direkt über den Konstruktor der jeweiligen Oberklasse gesetzt. Für die IDs gibt es Konstanten in der Datei „ClassConstants.h“.

Für Driver muss außerdem die Operation checkDev implementiert werden. Diese dient dem Erkennen und Initialisieren eines noch nicht vorhandenen Gerätes. Besitzt ein Treiber Abhängigkeiten zu anderen Treibern, so müssen diese dem Attribut „dependencies“ der Klasse Driver zugeordnet werden. Hierfür werden ebenfalls die DeviceClassIDs verwendet.

14.5.2 Benutzen von Treibern

Die Treiber selbst dienen lediglich dem Initialisieren von Geräten. Die eigentliche Verwendung eines Gerätes ist in einer Unterklasse von Device implementiert.

Wird ein Driver-Objekt initialisiert, muss es dem DriverManager bekannt gemacht

werden. Dies muss geschehen, bevor die Methode „initializeDevices“ aufgerufen wird, da in dieser Methode die Initialisierung der zugehörigen Device-Objekte erfolgt.

Wird ein Gerät für die Ausführung eines Programmes benötigt, kann die get()-Methode des DeviceManagers verwendet werden, um nach einem Device einer bestimmten Klasse zu suchen.

Um beispielsweise eine Referenz auf ein Device-Objekt für den Speacker zu bekommen, kann folgendes Beispiel angewandt werden.

```
1 Device *d = DeviceManager::getInstance()->get<Speacker>(AUDIODEVICE);
```

15 Tastaturtreiber

15.1 Grundlagen

Die Tastatur ist im Gegensatz zu allgemeinen Meinung ein komplexes Gerät. Denn diese besitzt neben einer Scanmatrix auch einen Tastaturchip der viele Aufgaben übernimmt. Bei der x86 Architektur gibt es drei Generationen von Tastaturen: XT-, AT- und die MF[49] II-Tastaturen. Diese sind alle abwärts kompatibel. Somit können die heutzutage am häufigsten verwendeten MF II-Tastaturen auch an älteren Geräten verwendet werden. Ab der AT-Tastatur ist es auch möglich Befehle an die Tastatur zu senden. Damit kann der Zustand der Tastatur abgefragt werden und die Steuerung der LEDs und Wiederholrate der Tastatur ermöglicht.

15.1.1 Aufbau von Tastaturen

Die technische Erkennung der Tastananschläge geschieht im Wesentlichen über zwei Komponenten, der *Scanmatrix* und dem *Tastaturchip*

Scanmatrix

Die Scanmatrix der Tastatur besteht im Wesentlichen aus sich schneidenden Leitungen an deren Kreuzungen ein Schalter verbaut ist. Dieser wird beim Drücken einer Taste geschlossen und beim Loslassen der Taste wieder geöffnet.

Tastaturchip

Der Tastaturchip ist für mehre Aufgaben einer Tastatur verantwortlich. Das Prellen[50] der Tasten wird vom Tastaturchip abgefangen. Der Tastaturchip ist intelligent genug das Prellen vom schnellen Tippen des Benutzers zu unterscheiden.

[49] Multifunktions-Tastatur

[50] Unter Prellen versteht man das schnelle schließen, öffnen und wieder schließen des Schalters. Dieses tritt wegen der federnden Rückstellkraft der Tastenschalter auf.

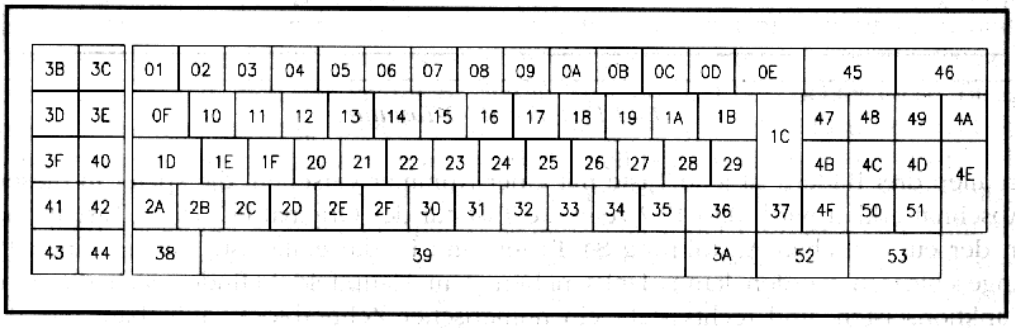


Abbildung 15.1: PC Tastatur (PC/XT)

Für das immerwährende Senden des *Scancodes* der Taste bei längerem Drücken einer Taste, ist ebenfalls der Tastaturchip verantwortlich. Hier kann auch erst ab der AT Generation die Wiederholrate des Sendens eingestellt werden.

15.1.2 Tastaturmodelle

Im Laufe der Zeit wurden verschiedene Tastaturmodelle entwickelt, welche zueinander kompatibel gehalten wurden.

PC Tastatur (PC/XT)

Die erste Genetation von PC Tastaturen (PC/XT) in Abbildung 15.1 hatte 83 Tasten. Die Treiber die zu dieser Zeit entstanden sind, können von daher auch nur 83 Scancodes interpretieren.

AT-Tastatur

Bei der AT-Tastatur in Abbildung 15.2 ist gegenüber der XT-Tastatur nur die „SysReq“-Taste mehr hinzu gekommen, die auch einen eigenen Scancode (54h) erhalten hat. Die Codebelegungen wurden z.T. umgestellt. Bspw. liefert die Leertaste nun den Keycode 3Dh anstatt 39h wie bei der XT-Tastatur.

MF II-Tastatur

Die heutigen MF II-Tastaturen in Abbildung 15.3 haben 101 (US) bzw. 102 (EU) Tasten. Dabei sind die Tasten „F11“- und „F12“ hinzugekommen die einen eigenen Scancode erhalten haben. Zusätzlich sind hier drei LEDs, die den Status der Umschalttasten (Num Lock, Caps und Rollen) anzeigen. Für die Abwärtskompatibilität sind für die neuen Funktionstasten keine Scancodes hinzugekommen,

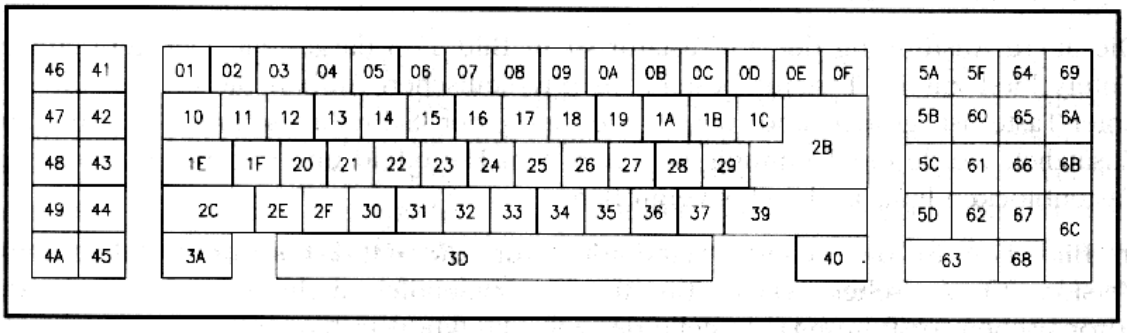


Abbildung 15.2: AT-Tastatur

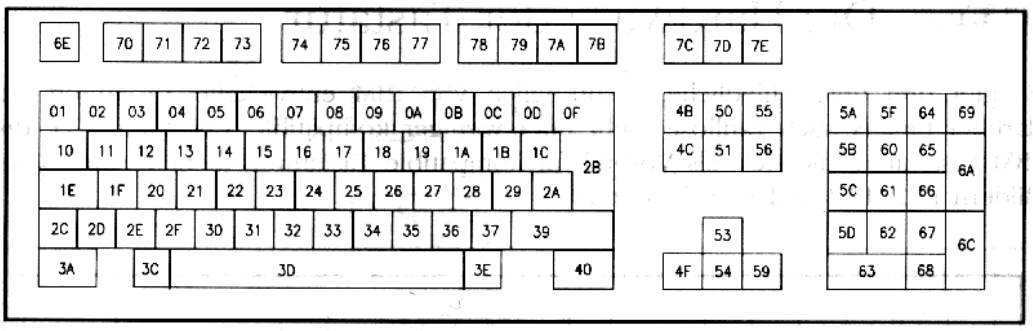


Abbildung 15.3: MF II-Tastatur

stattdessen sendet die Tastatur (Tastaturchip) hier mehrere auf einander folgende Scancodes, für die neu hinzugekommenen Tasten.

15.1.3 Funktionsweise von Tastaturen

Der Tastaturchip überprüft in regelmäßigen Abständen den Zustand der Schalter der Scanmatrix. Dazu aktiviert er nacheinander einzeln die X-Leitungen und erfasst an welchen Y-Anschluß ein Signal anliegt. Über die X und Y Koordinaten ist die gedrückte oder losgelassene Taste eindeutig durch einen Code identifizierbar.

Zu diesem Code sucht der Tastaturchip, bei *MF-Tastaturen*, in seinem derzeit verwendeten *Code Set* den zugehörigen sogenannten Scancode. AT- und XT-Tastaturen verwenden den Code direkt als Scancode. Zu diesem Scancode wird noch die Information hinzugefügt ob die Taste gedrückt oder losgelassen worden ist. Der Code der Taste + Information ergibt den endgültigen Scancode.

Den Scancode legt der Tastaturchip in einem internen Puffer der Tastatur ab und sendet über die serielle Schnittstelle, an der die Tastatur angeschlossen ist, an die Tastaturschnittstelle[51] des PCs eine Nachricht, dass ein Ereignis stattgefunden hat. Diese Nachricht beinhaltet auch den Scancode. Dadurch wird im PC der Hardware-Interrupt 1 (INT 21h) ausgelöst. Die dafür zuständige ISR muss nun

[51] Bei PC/XT ist es nur eine serielle Schnittstelle, ab AT ist hier ein Kontroller verbaut.

dafür sorgen, dass der Scancode aus dem Puffer der Tastatur ausgelesen wird und von dort auch entfernt wird. Denn der Puffer der Tastatur ist recht klein (20 Byte) und wird nicht automatisch geleert oder überschrieben wenn er voll ist.

15.1.4 Scancodes

Scancodes sind Codewörter die Informationen über die gredrückte Taste auf einer Tastatur enthalten.

Scancodesets

Code Sets sind Tabellen bei MF-Tastaturen, die für die Umsetzung von Codes zu Scancodes benutzt werden: Scan Code Set 1: entspricht der XT-Tastatur Scan Code Set 2: entspricht der AT-Tastatur (heutiger Standard bei Tastaturen) Scan Code Set 3: entspricht der MFII-Tastatur (hat sich nie durchgesetzt)

Welches Code Set die Tastatur benutzen soll kann für MF-Tastaturen eingestellt werden. Standardmäßig ist es der Code Set 2.

Das PC-BIOS kann durch Einstellung den Code Set 2 zum Code 1 (AT zu XT) umsetzen. Diese Einstellung ist standardmäßig aktiviert, kann aber von modernen Betriebssystemen übergangen werden.

Scanmatrix: Code → Tastaturchip: Scancode für AT → BIOS: Scancode für XT → Treiber: Zeichen (ASCII Code)

Break- und Make-Codes

Für die Unterscheidung ob eine Taste gedrückt oder losgelassen worden ist, wird der Scancode in Make- und Break-Code unterteilt. Der Make-Code entsteht beim Drücken der Taste, der Break-Code beim Loslassen dieser. Bei jedem Tastendruck werden also zwei Scancodes erzeugt. Beispielsweise die ’ESC’-Taste:

1. Make-Code 01h
2. Break-Code 81h (1 + 128).

geschützte Scan-Codes

Aus Kompatibilitätsgründen wurden die schützenden (escape) Scancodes e0 und e1 eingefügt um weitere Tasten zukodieren. Dabei wird erst ein schützender

Scancode (e0) gesendet auf den dann der eigentliche Scancode folgt. Der Hintergrund dabei ist, dass alte Programme den schützenden Scancode ignorieren und der zweite Scancode sinnvoll interpretiert werden kann.

Auf den schützenden Scancode (e1) folgen weitere zwei Scancodes, welche dann z.B. zur Pause Taste korrespondieren.

Fake Shifts

Die Tasten Insert, Home, PgUp, Entf, End, PgDn, Hoch, Links, Runter und Rechts sollen unabhängig von dem Zustand von Shiftlock und Numlock funktionieren. Aber auf alten AT Tastaturen würden diese Tasten Zahlen produzieren wenn Numlock aktiv oden Shift gedrückt wäre. Deshalb werden falsche Shifts gesendet um alte Programme „auszutricksen“. Wenn z.B. LShift gedrückt ist und Einfügen gedrückt wird, wird e0 aa e0 52 gesendet. Beim loslassen wird e0 d2 e0 2a gesendet. Das heißt, dass ein falscher LShift-up und LShift-down eingefügt wird.[52]

Interpretation der Scancodes

Die Tastatur sendet an den PC nur einen Scancode der Taste die gedrückt oder losgelassen worden ist. Für die Zuordnung des Scancodes zu einem Zeichen ist der Tastaturtreiber verantwortlich. Dabei hält der Tastaturtreiber eine Tabelle vor, die die Information enthält zu welchem Scancode welches Zeichen gehört. Erst durch diese Gegebenheit sind verschiedene Tastaturlayouts möglich ohne die Elektronik der Tastatur ändern zu müssen. Beispielsweise wird der Scancode 15h beim deutschen Layout zu ’z’ übersetzt, beim englischen Tastaturlayout zu ’y’.

Nicht jede Taste kann zu einem Zeichen zugeordnet werden, wie bspw. die Umschalt oder STRG- Taste. Solche Tasten werden als „stumme“ Tasten bezeichnet. Manche dieser Tasten, wie die Umschalt- oder „Alt Gr“-Taste haben aber einen Einfluss auf die Umsetzung der Scancodes zu Zeichen. Der Tastaturtreiber muss sich also merken ob z.B. die Umschalttaste gedrückt ist während eine andere Taste gedrückt wird, um diesen Tastendruck richtig umzusetzen. Für diese Umsetzung muss ich der Treiber kümmern.

[52] Näheres unter `http://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html`.

15.1.5 PS/2 Kontroller und Tastatur

Der PS/2 Kontroller ist ein Baustein auf der Hauptplantine, der früher nur für die Kommunikation mit den PS/2 Geräten eingesetzt wurde und heutzutage Teil eines Multichips (AIP) ist, mit dem viele verschiedene Geräte angesprochen werden können.

Der PS/2 Kontroller hat zwei (ein byte-große) Datenbuffer. Einer in dem Daten gespeichert werden die vom Betriebssystem gelesen werden sollen und einer für Daten die an das PS/2 Gerät gesendet werden sollen.

Ferner verwendet der PS/2 Kontroller zwei verschiedene Ports zur Kommunikation, wie in Tabelle 15.1 zusehen ist.

Tabelle 15.1: IO Ports

IOPort	Zugriffsart	Einsatzzweck
0x60	Lesen/Schreiben	Daten Port
0x64	Lesen	Status Register
0x64	Schreiben	Kommando Register

Dabei bietet das Status Regisert verschiedene Flags an die über den Zustand des PS/2 Kontroller auskunft geben. Über das Kommando Register können Befehle an den PS/2 Kontroller gesendet werden. Einen Überblick über mögliche Zustände und Befehle findet sich in [**?**].

Das PS/2 Keyboard ist ein Gerät das über den PS/2 Kontroller kommuniziert. Dabei sendet und akzeptiert es verschiedene Kommandos und stellt die Scancods beim Drücken und Loslassen von Tasten bereit.

Eine Übersicht über mögliche Befehle ist in [**?**] zu sehen.

15.2 Entwurf

Um einen modularen Aufbau zuerhalten und das Treiberframework optimal auszunutzen, werden einzelne Teile des Tastaturtreibers als eigen Treiber implementiert.

- Der *PS/2 Device* bietet einen Zugriff auf die zwei PS/2 Ports an.
- Über die *PS/2 Tastatur Device* werden Scancodes bereit gestellt und Tastatureinstellungen wie das Leuchten von LEDs oder Wiederholungsraten vorgenommen.

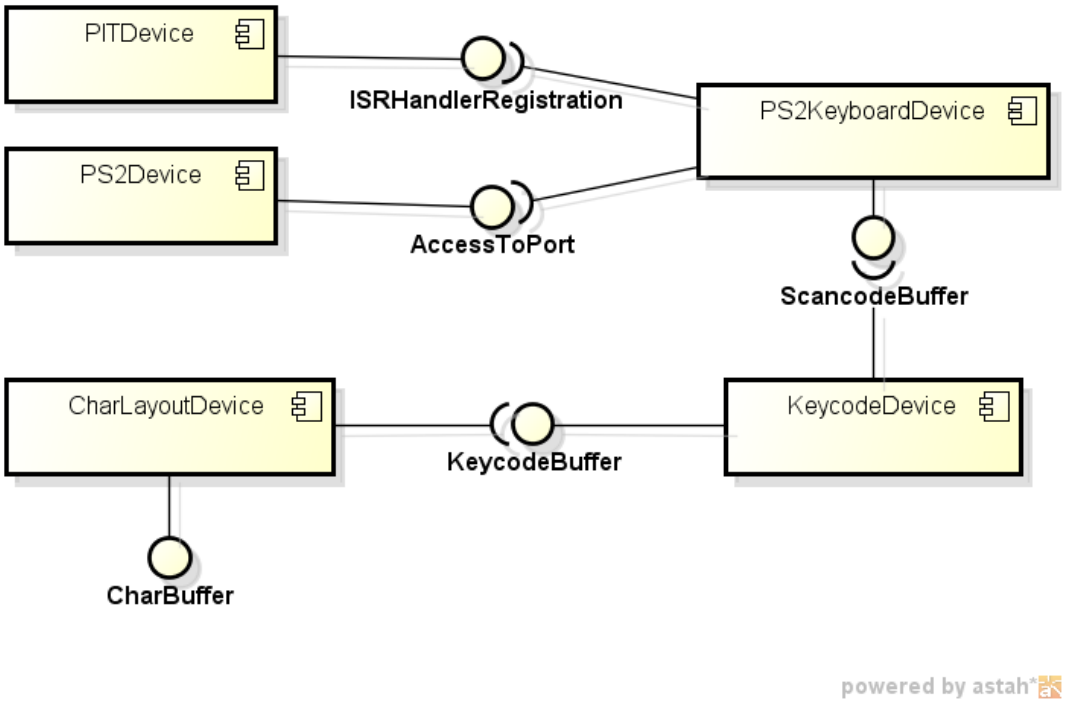


Abbildung 15.4: Zusammenspiel der einzelnen Devices

- Über den *KeycodDevice* werden Scancodes zu Keycodes übersetzt, um vom eigentlichen Tastaturlayout zu abstrahieren und einen einheitlichen Zugriff für verschiedene Anwendungsprogramme zur Verfügung zu stellen.
- Das *CharLayoutDevice* bietet eine erst beispielhafte Implementierung einer deutschen Tastatur an, indem sie Keycodes zu Zeichen übersetzt.

15.2.1 Zusammenspiel der einzelnen Devices

Das Zusammspiel und die Abhängigkeiten der einzelnen Devices ist in Abbildung 15.4 zusehen.

Der CharLayoutDevice nutzt den KeycodeBuffer des KeycodeDevices um selbst einen CharBuffer zur Verfügung zustellen. Der KeycodeDevice nutzt den ScancodeBuffer des PS2KeyboardDevices. Dieses benutzt die Interrupthandlerregistrierung des PITDevices und die Portansteuerung des PS2Devices.

15.2.2 PS/2 Device

Der PS/2 Device dient der Ansteuerung des PS/2 Kontrollers, um die PS/2 Ports und Interrupts zu aktivieren. Dabei ist das PS/2 Device wie in Abbildung 15.5 zusehen aufgebaut.

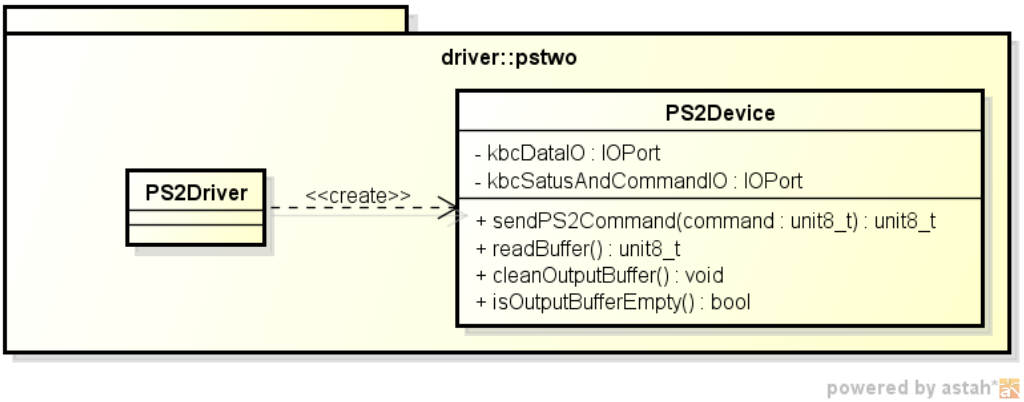


Abbildung 15.5: PS/2 Device

Die Initialisierung des PS/2 Device beinhaltet die folgenden Schritte:

1. PS/2 Ports ausschalten,
2. Ausgabebuffer leeren,
3. PS/2 Interrupts und Scancodewandlung abschalten
4. PS/2 Kontroller und PS/2 Port Selbsttests durchführen
5. PS/2 Ports einschalten und
6. PS/2 Interrupts und Scancodewandlung einschalten.

15.2.3 PS/2 Tastatur Device

Der PS/2 Tastatur Device dient dem Konfigurieren der Tastatur und dem Bereitstellen der Scancodes in einem Scancodebuffer. Dazu registriert er einen entsprechenden Interrupthandler, der beim Drücken einer Taste, die Scancodes in den Scancodebuffer übermittelt.

Der Aufbau des PS/2 Tastatur Device ist in Abbildung 15.6 zu sehen.

15.2.4 Keycode Device

Der Keycode Device führt eine Abstraktionsschicht – die sogenannten *Keycodes* – ein, um vom Tastaturlayout zu abstrahieren und um den Anwendungsprogramme einen einheitlichen Zugriff zu ermöglichen.

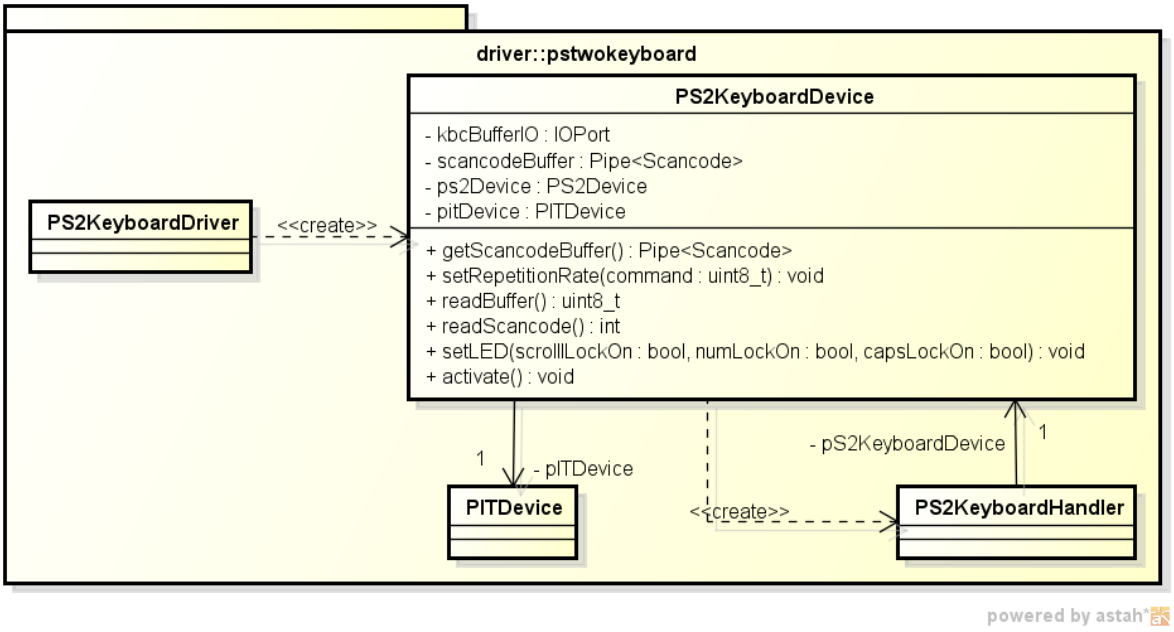


Abbildung 15.6: PS/2 Tastatur Device

Keycode

Da für das Drücken und Loslassen einer Taste (maximal 6) Scancodes erzugt werden und sich daher schlecht mit ihnen arbeiten lässt, werden die Scancodes auf Keycodes abgebildet. Tabelle 15.2 zeigt die erkennbaren Tastendrücke und deren Keycode Zuordnung des Treibers.

Aufbau und Funktionsweise des Keycode Device

Der Keycode Device ist mithilfe des *Pipes und Filters*[53] Entwurfsmusters implementiert. Dazu wird beim Initialisieren des Keycode Devices ein Workerthread gestartet, der aus dem Scancodebuffer ausliest und in einen Keycodebuffer schreibt, sobald er Scancodes nach Keycodes übersetzt hat. Die Pipe ist mithilfe einer Semaphore implementiert, die die Thread blockiert, falls sich keine Elemente mehr in der Pipe befinden. Beim Hinzufügen neuer Elemente wird der Thread wieder „aufgeweckt“, kann also wieder vom Scheduler ausgewählt werden.

Um die in Abschnitt 15.1.4 beschriebenen geschützten Scancodes zu behandeln, wird der Zustandsautomat in Abbildung 15.7 verwendet. Der *InitialScanCodeConverterState* verarbeitet die gewöhnlichen Scancodes indem er sie in Keycodes übersetzt und zurückgibt. Wird der schützende Scancode $E0$ respektive $E1$ übergeben, wird in den *EZeroScanCodeConverterState* respektive den *EOnePhaseOneScanCodeConverterState* gewechselt. Im *EZeroScanCodeConverterState* wird

[53] Nähere Informationen in [**?**]

Tabelle 15.2: Keycodes

KC	Taste	KC	Taste	KC	Taste	KC	Taste
1	ESC	2	1	3	2	4	3
5	4	6	5	7	6	8	7
9	8	10	9	11	0	12	Hyphen
13	Plus	14	Backsapce	15	Umschalt	16	Q
17	W	18	E	19	R	20	T
21	Y	22	U	23	I	24	O
25	P	26		27		28	Enter
29	STRG links	30	A	31	S	32	D
33	F	34	G	35	H	36	J
37	K	38	L	39		40	
41		42	linkes Shift	43		44	Z
45	X	46	C	47	V	48	B
49	N	50	M	51	Pipe	52	Peint
53		54	rechtes Shift	55		56	ALT
57		58	GShift	59	F1	60	F2
61	F3	62	F4	63	F5	64	F6
65	F7	66	F8	67	F9	68	F10
69		70	Scroll lock	71	KP7	72	KP8
73	KP9	74	KP Minus	75	KP4	76	KP5
77	KP6	78	KP Plus	79	KP1	80	KP2
81	KP3	82	KP0	83	KP Decimal	84	
85		86		87	F11	88	F12
89		90		91		92	
93		94		95		96	KP Enter
97	STRG rechts	98		99	KP Div	100	ALT GR
101		102	Home	103	Pfeil hoch	104	Seite hoch
105	Pfeil links	106	Pfeil rechts	107	Ende	108	Pfeil runter
109	Seite runter	110	Einfügen	111	Entfernen	112	
113		114		115		116	
117		118		119	Pause	120	

dann aus dem übergebenen Scancode der entsprechende Keycode bestimmt und zurück in den *InitialScanCodeConverterState* gewechselt. Der *EOnePhaseOneScanCodeConverterState* empfängt den übergebenen Scancode und gibt ihn an den *EOnePhaseTwoScanCodeConverterState* weiter. Dieser verrechnet ihn mit dem nächsten übergebenden Scancode und bestimmt den Keycode. Danach wird wieder in den *InitialScanCodeConverterState* gewechselt.

Der gesamte Aufbau des Keycode Devices ist in Abbildung 15.8 zusehen.

Durch den Austausch der einzelnen Pipes lässt sich ein gutes Testszenario erzeugen, indem erzeugte Inhalte mit erwarteten Inhalten innerhalb der Buffer verglichen werden.

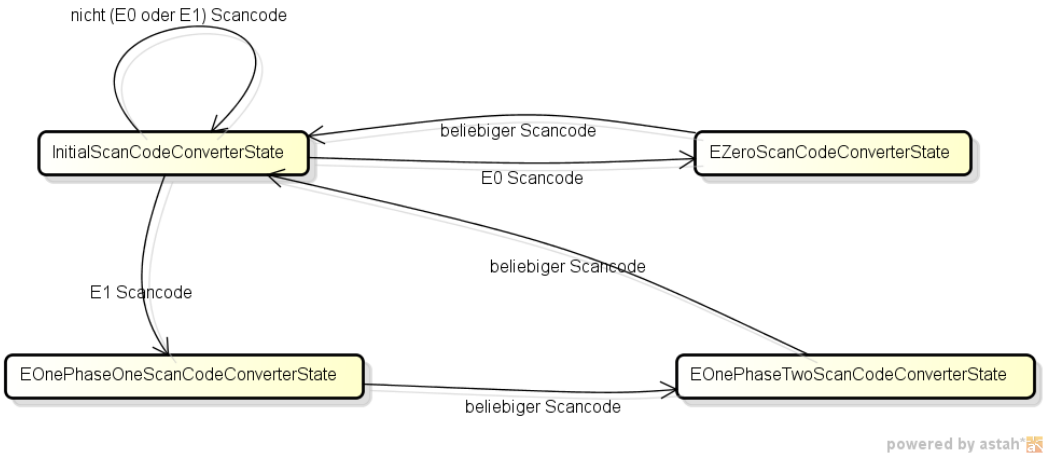


Abbildung 15.7: Zustandsdiagramm zur Behandlung geschützer Scan-Codes

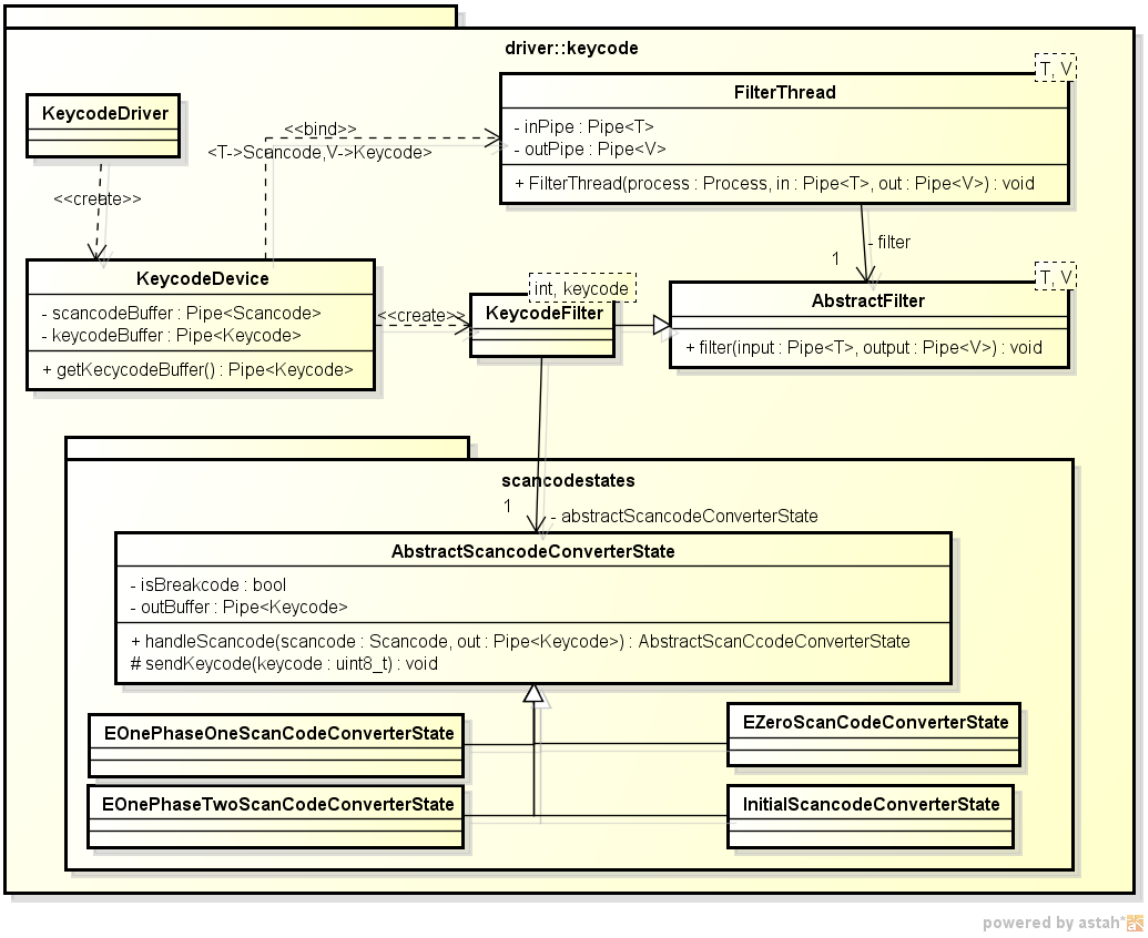


Abbildung 15.8: Keycode Device

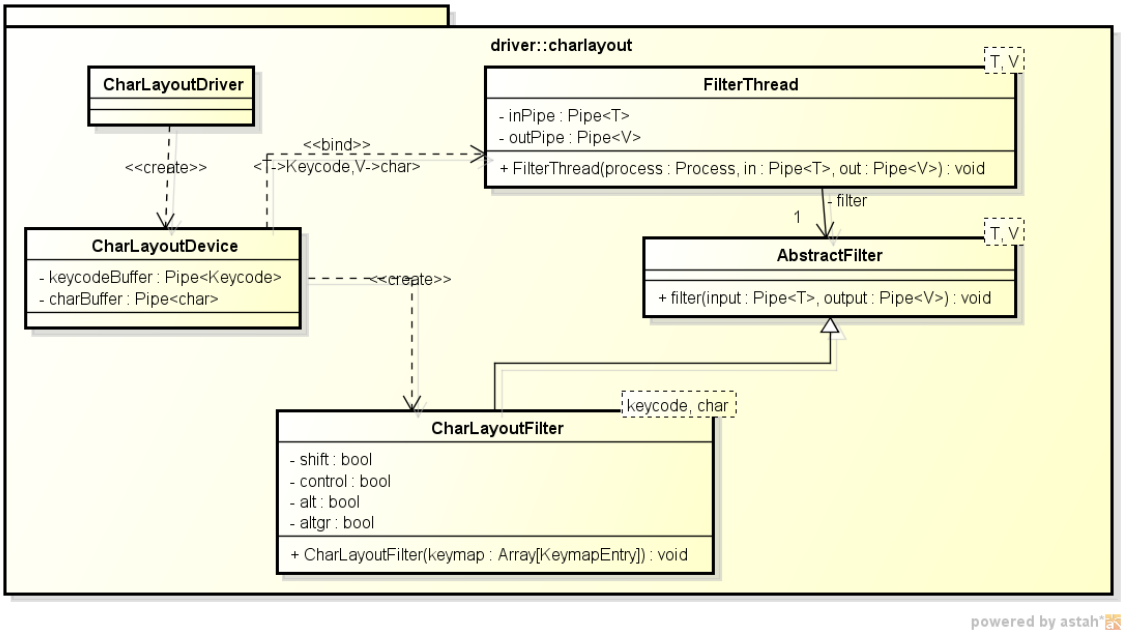


Abbildung 15.9: Char Layout Device

15.2.5 Char Layout Device

Der Char Layout Device ist eine vorläufige Implementierung eines Tastaturlayouts. Es ist ebenso mithilfe des *Pipes und Filters* Entwurfsmusters implementiert, indem es aus dem Keycodebuffer liest und in einen Charbuffer schreibt. Andere Threads können dann aus dem Charbuffer die Tastatureingaben lesen. Dabei bietet der Char Layout Device Unterstützung für Groß- und Kleinschreibung durch das gerdrückt Halten der Shift-Taste. Ferner sind einige Sonderzeichen implementiert.

Der gesamte Aufbau des Char Layout Devices ist in Abbildung 15.9 zusehen.

15.3 Fazit und Ausblick

Der Tastaturtreiber wurde durch die Entwicklung vier unterschiedlicher Treiber umgesetzt. Dabei wurden der PS/2 und PS/2 Tastatur Device definiert um die PS/2 Tastatur zu verwalten und Scancodes zur Verfügung zustellen. Dafür wurde ein ISRHandler definiert der die Scancodes bei den auftretenden Tastaturinterrupts in einem Buffer zur Verfügung stellt.

Der KeycodeDevice arbeitet nach dem Pipes und Filters Entwurfsmuster und stellt entsprechend Keycodes in einem Buffer bereit. Die Keycodes dienen dabei als Abstraktionsschicht für Anwendungsentwickler, die dadurch nicht mehr mit Scancodes arbeiten müssen.

Ein Beispiel für so eine Anwendung ist der CharLayoutDevice der Keycodes enstsprechend eines deutschen Tastaturlayouts übersetzt und einen Buffer mit Zeichen zur Verfügung stellt.

Insgesamt erkennt der Treiber die meisten Scancodes und stellt Keycodes bereit. Durch die Modularisierung wird eine gute Wiederverwendbarkeit ermöglicht. Das Pipes and Filters Entwurfsmuster garantiert bei der Transformation von Codes, dass Threads nur aus den Buffern auslesen, wenn sie Elemente enthalten. Andernfalls werden die Threads blockiert und erst wieder aufgerufen sobald neue Elemente hinzugefügt wurden.

16 Konsole im VGA-Textmodus

16.1 Video-RAM

Im Video-RAM sind alle Bildinformationen gespeichert. Im Adressraum befindet sich der Video-RAM im Speichersegment B ab der Segmentadresse $B000h$ und umfasst $64KB$. Im Grafikmodus gibt es teilweise große Unterschiede in der Verwaltung des Video-RAM. Im Textmodus ist der Aufbau des Video-RAM jedoch einheitlich. Ausgehend von der Startadresse des VGA-RAM enthalten die nächsten 4000 Byte alle Bildinformationen für den Textmodus. Dieser Bereich beschreibt eine Konsole mit 25 Zeilen mit je 80 Zeichen. Da für jedes Zeichen zwei Byte verwendet werden, lässt sich das Offset eines Zeichens wie folgt berechnen:

$$Offset = 2 \cdot (80 \cdot Zeile + Spalte)$$

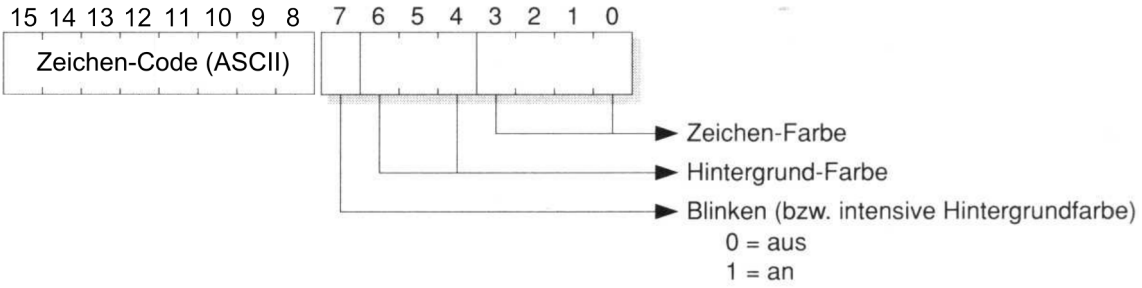


Abbildung 16.1: Struktur eines Zeichens im Video-RAM

Jedes Zeichen besteht aus zwei Byte: einem für die ASCII-Codierung des Zeichens und einem zweiten für zusätzliche Attribute. Wie **??** zeigt, sind im zweiten Byte die Text- und Hintergrundfarbe des Zeichens gespeichert. Während für die Textfarbe 4 Bit (16 Farben) zur Verfügung stehen, sind für die Hintergrundfarbe nur 3 Bit verfügbar. Aus der Farbtabelle in **??** sind dies die 8 Farben auf der linken Seite. Ist das Bit 7 des Attribut-Byte gesetzt, so blinkt das Zeichen – es wird also abwechselnd ein- und ausgeblendet.

#000000	#555555
#0000AA	#5555FF
#00AA00	#55FF55
#00AAAA	#55FFFF
#AA0000	#FF5555
#AA00AA	#FF55FF
#AA5500	#FFFF55
#AAAAAA	#FFFFFF

Abbildung 16.2: 4-Bit-Farbpalette im VGA-Textmodus

16.2 CRT-Controller

Der CRT-Controller steuert maßgeblich den Bildschirmaufbau. Dazu zählt z. B. die Koordination des Elektronenstrahls von Röhrenmonitoren. Denn ist dieser am unteren Bildschirmrand angelangt, so erfolgt sogenannter vertikaler Rücklauf (engl. vertical retrace) nach oben. Änderungen am Video-RAM sollten nur innerhalb dieser Zeitspanne erfolgen, da sonst *Schnee* auf dem Bildschirm entstehen kann. Jedoch sollte die Programmierung der Register des CRT-Controllers weitgehend dem BIOS überlassen werden. Lediglich eine kleine Auswahl dieser Register wird nicht direkt vom BIOS verwendet und ist für den Programmierer relevant. Dazu gehören insbesondere die Register $0Eh$ (Cursor Location High) und $0Fh$ (Cursor Location Low), die die Cursor-Adresse beinhalten. Diese gibt den Offset des Cursors innerhalb des Video-RAM an. Die Höhe des Cursor lässt sich durch Manipulieren der Register $0Ah$ und $0Bh$ realisieren.

16.3 Entwurf

Um den VGA–Treiber zu implementieren wurden für die zwei Klassen Driver und Device des Treiber–Frameworks VGA-Spezialisierungen erstellt. Hierbei wurde noch eine weitere abstraktere Spezialisierungsebene eingefügt, um später weitere Textanzeigegeräte hinzufügen zu können wie zum Beispiel für eine ältere CGA-Garfikkarte. Das abstrakte Textanzeigegerät wird durch die Klasse TextDevice repräsentiert.

Des Weiteren wurde eine Unterteilung von Funktionen, die das reine Setzen und Lesen von Zeichen, Attributen und des Cursors, von Funktionen, die eine Kom-

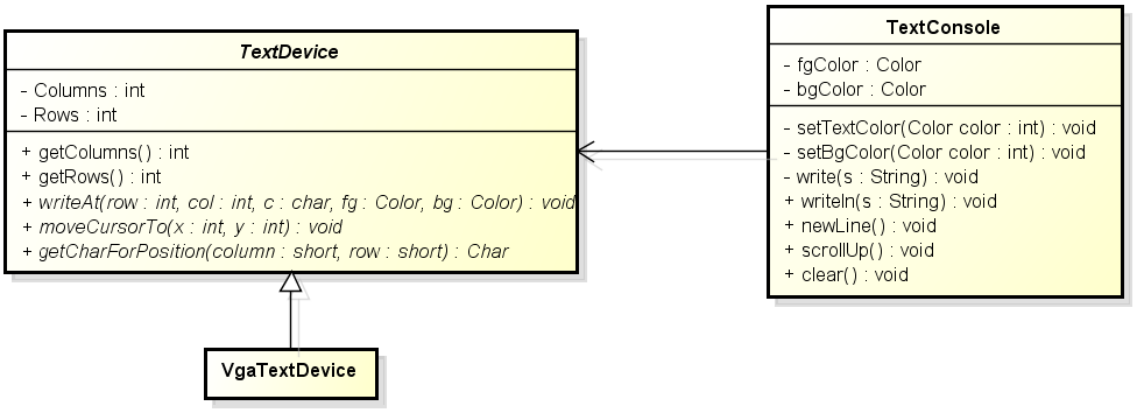


Abbildung 16.3: Klassendiagramm des TextDevice und der TextConsole

bination aus den vorher genannten Funktionen sind, durchgeführt. Ein Beispiel für eine kombinierte Funktion ist das Schreiben eines Zeichen und gleichzeitiges setzen des Cursors auf die nächsten Position an der geschrieben werden soll. Die Funktionen zum Setzen und Lesen werden durch das TextDevice bereitgestellt. Die kombinierten Funktionen befinden sich in der zusätzlichen Klasse TextConsole die ein TextDevice benutzt. In Abbildung 16.3 sind die beiden erwähnten Klassen dargestellt. Nachfolgend wird zunächst das TextDevice und die Spezialisierung VgaTextDevice die die Implementierung für den VGA-Textmodus enthält, erklärt. Darauf wird auf den Entwurf der TextConsole eingegangen.

16.3.1 TextDevice

Die Organisierung der einzelnen Zeichen erfolgt im TextDevice in Spalten und Zeilen deren Anzahl bei der Erstellung des TextDevice festgelegt werden müssen. Beim VGA Modus ist die auf eine Spaltenanzahl von 80 und einer Zeilenanzahl von 25 fest eingestellt. Die Position für ein Zeichen wird mithilfe dieser Spalten und Zeilen in Form von Koordinaten ermittelt. Wobei die Spalte die x–Koordinate und die Zeile die y–Koordinate ist. Der Nullpunkt des Koordinatensystems ist auf dem Bildschirm links oben.

Um ein Zeichen zu schreiben wird die Operation writeAt() genutzt. Dieser schreibt das angegebene Zeichen an die übergebene Position mit den angegebenen Vordergrund und Hintergrundfarbe. Außerdem kann für eine Position das Zeichen und seine Attribute abgefragt werden. Dazu wird ein Objekt des Typ TextChar zurückgeliefert. Die TextChar Schnittstelle biete sämtliche Operationen zum Setzen und Lesen des Zeichen und der Attribute an. Die Implementierung für den VGA-Textmodus wurde in der Klasse VgaChar durchgeführt. Dabei wird beim Aufrufen der Operationen direkt auf den VGA Videospeicher zugegriffen und die Werte entsprechend angepasst.

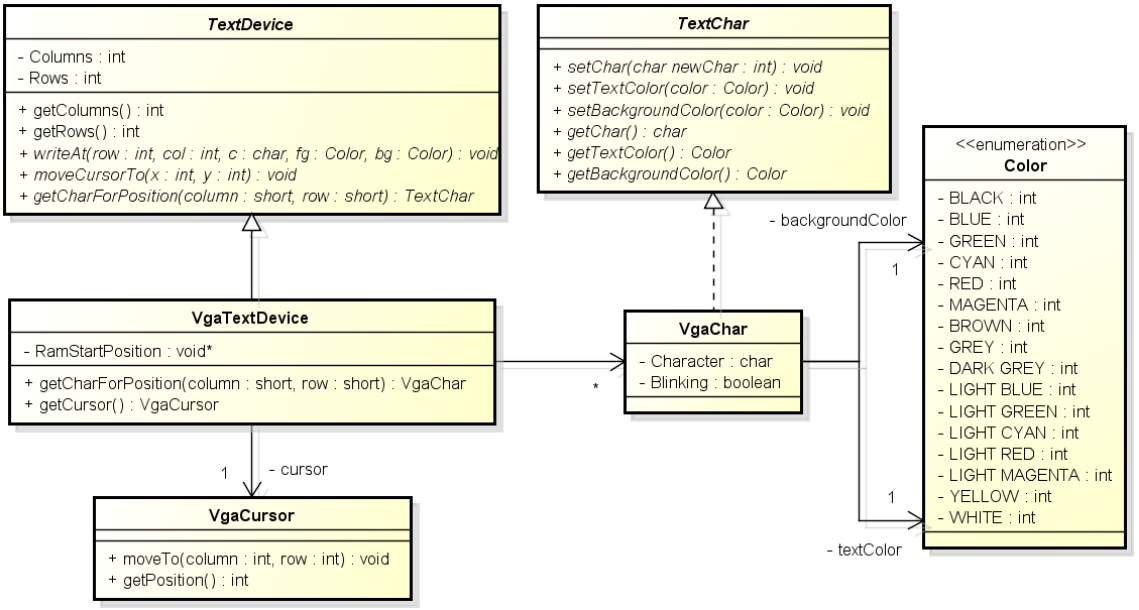


Abbildung 16.4: Klassendiagramm des VGA-Texttreibers

Zum Verschieben des Cursor dient die Operation moveCursor(). Bei dieser wird durch Angabe einer Position der Cursor verschoben. Beim VGA-Textmodus wird der Cursor durch Änderung der CRT-Register Cursor Location High und Cursor Location Low. In diesem wird die Cursor Position als Offset gespeichert. Die Umwandlung der Position bestehend aus x und x Koordinate in das Offset findet in der Klasse VgaCursor statt. In dieser ist auch die Logik enthalten um die genannten Register zu beschreiben.

Die in diesem Abschnitt erläuterten Klassen sind in der Abbildung 16.4 dargestellt.

16.3.2 TextConsole

Die TextConsole übernimmt Funktionen die aus Kombination mehrerer einzelner Funktionen des TextDevice bestehen. Beim write() wird zum einen das Zeichen geschrieben und zum anderen der Cursor um eine Position weiter gesetzt. Außerdem werden auch Steuerzeichen wie zum Beispiel \r interpretiert und in ihrer entsprechenden Funktion umgesetzt. Wie bei \r zum Beispiel das Setzen des Cursor an den Anfang der Zeile.

Des Weiteren werden bei allen Funktionen die durch die TextConsole angeboten werden jeweils immer die gesetzten Text und Hintergrundfarben berücksichtigt. Sollten die Text und Hintergrundfarben während des Betriebs der TextConsole geändert werden, werden die neuen Farben nicht für die schon bestehenden Zeichen übernommen, sondern nur für neu ausgeführte Funktionen. Um sämtliche

Farben zu übernehmen müsste ein clear() ausgeführt werden.

Wenn eine neue TextConsole erstellt wird, wird der Bildschirm immer in eine definierte Ausgangszustand versetzt. Dieser löscht sämtliche Zeichen des Bildschirms und setzt die Hintergrundfarbe auf die gewünschte Farbe. Der Cursor wird auf die Position (0,0) gesetzt.

16.4 Offene Punkte

Die Konsole wird bereits sehr früh im System verwendet. So werden neben Statusmeldungen auch Fehlermeldungen ausgegeben. Da diese Verwendung eventuell bereits vor der Treiberinitialisierung stattfindet, wurde die bisherige Konsolenimplementierung (`console.h`) beibehalten. Eine automatische Delegation der Konsole an den Treiber kann implementiert werden, dennoch sollte `console` nur verwendet werden, solange der VGA-Treiber nicht initialisiert wurde.

17 Zeitgeber Intel 8254

17.1 Analyse

Es soll ein Treiber für den programmierbaren Zeitgeber (engl. *programmable interval timer*, PIT) Intel 8254 entwickelt werden. Der Intel 8254 wird bereits seit Mitte der 80iger in PCs verbaut und befindet sich aus Kompatibilitätsgründen (neben weiteren neueren PITs) noch heute in jedem IBM kompatiblen PC[54]. Verwendung findet er bei allen Timing- und Zählfunktionen.

Der 8254 verfügt über drei unabhängige runterzählende 16 Bit Zähler (*Counter 0*, *Counter 1* und *Counter 2*). Jeder dieser Zähler hat drei Hardwarepins. Zwei Eingänge – **CLK** (Takt) und **GATE** (wird je nach Zählermodus verwendet, s. u.) – und einen Ausgang, **OUT**, als Ausgabe. Der Takt der Zähler liegt bei 1,193180 MHz. In Abbildung 17.1 ist die Hardware als Schaubild dargestellt.

In älteren Systemen waren die Funktionen der einzelnen Zähler immer gleich belegt. Der erste Zähler (angebunden an Interrupt 0 des Interrupt-Controllers) wird vom Betriebssystem genutzt, um eine Software-Uhr zu implementieren. Der zweite Zähler wird zum Auffrischen des DRAMs eingesetzt und der letzte Zähler wird für die Tonerzeugung des PC-Speakers eingesetzt.

[54] Heutzutage wird der Intel 8254, bzw. ein äquivalentes Schaltnetz, in größere Chips integriert.

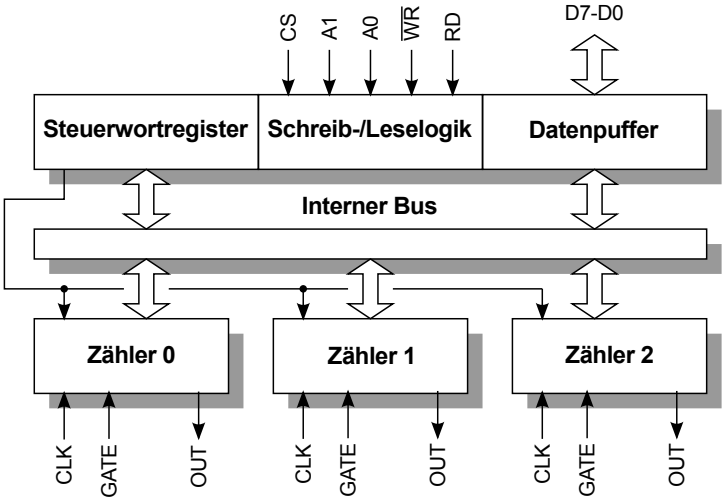


Abbildung 17.1: Hardwareschaubild des Intel 8254

Heutzutage wird nur noch der letzte Zähler für seinen ursprünglichen Zweck verwendet. Jeder dieser Zähler kann in einem der folgenden Zählermodi betrieben werden:

Modus 0 - Interrupt bei Zählende

Der OUT-Pin ist anfangs auf einem niedrigen Pegel (0 V). Der Zähler zählt vom programmierten Zählerwert abwärts. Wenn der Zähler den Stand 0 erreicht, steigt der OUT-Pegel und bleibt auf einem hohen Pegel (+5 V). Der OUT-Pegel sinkt erst wieder, wenn der Zähler neu programmiert wird. Dieser Modus läuft demnach nur einmalig ab. Gesteuert werden kann der Modus über den GATE-Pin. GATE=1 aktiviert die Zählung, GATE=0 deaktiviert sie.

Modus 1 – Programmierbares Monoflop

Nach dem Programmieren des Modi und des Anfangszählwerts liegt der Ausgang OUT zunächst auf einem hohen Pegel. Ein Trigger (Übergang niedrig-hoch) am GATE-Eingang lädt den Anfangszählwert in den Zähler. Beim nächsten Takt fällt der OUT-Pin auf einen niedrigen Pegel. Der Ausgang bleibt auf diesem Pegel, bis der Zähler den Stand 0 erreicht. Anschließend wird der Ausgang wieder auf einen hohen Pegel gesetzt. Tritt während des Zählens ein erneuter Trigger auf, wird der Anfangszählwert erneut in den Zähler geladen.

Modus 2 – Ratengenerator

Nach der Programmierung des Modi und des Anfangszählwertes beginnt der Zähler runterzuzählen. Erreicht der Zähler den Wert 1, so fällt OUT für einen Takt auf einen niedrigen Pegel (Nadelimpuls). Anschließend beginnt der Zähler von vorn. Dieser Modus ist periodisch.

Modus 3 – Rechteckgenerator

Dieser Modus ist wie Modus 2 periodisch. Allerdings wird hierbei der OUT-Pin für die erste Hälfte des Zählvorgang auf einem hohen und für die zweite Hälfte des Zählvorgangs auf einem niedrigen Pegel gehalten. GATE=0 schaltet den Zähler aus. GATE=1 startet den Zähler erneut.

Modus 4 – Software-getriggerter Impuls

Der OUT-Pin liegt zu beginn auf einem hohen Pegel. Bei Erreichen des Wertes 0 sinkt der Pegel des Ausgangs für einen Takt und steigt anschließend wieder an. Modus 4 ist nicht periodisch. Im Gegensatz zu Modus 0 kann er jedoch über das erneute Setzen des Anfangszählwertes sofort erneut gestartet werden.

Modus 5 – Hardware-getriggerte Impuls

Dieser Modus arbeitet wie Modus 4, allerdings wird in diesem Modus der

Anfangszählerwert beim Eintreten des Trigger-Signals (Übergang niedrig-hoch am GATE-Pin) neu geladen.

Der PIT ist über vier I/O-Ports programmierbar. Die Ports 0x40, 0x41 und 0x42 stehen für das Auslesen und Schreiben von Anfangszählwerten der jeweiligen Zähler zur Verfügung. Zusätzlich gibt es den Port 0x43, welcher unter anderem zum Setzen der Modi genutzt wird. Auf Port 0x43 kann ausschließlich geschrieben werden. Es wird ein speziell aufgebautes Steuerwort erwartet. Der Aufbau dieses Steuerwortes ist wie folgt:

7 6	5 4	3 2 1	0
SC	RW	Modus	BCD

```
SC:    Select Counter (Zähler auswählen)
         00=Zähler 0         01=Zähler 1         10=Zähler2
RW:    Read/Write
         00=Zähler-Latch-Befehl
         01=Schreiben nur des niederwertigen Zählerbytes
         10=Schreiben nur des höherwertigen Zählerbytes
         11=Schreiben des niederwertigen, anschließend des
                                 höherwertigen Zählerbytes
Modus: Zählmodus des Zählers
         000=Modus 0       001=Modus 1           010=Modus 2
         011=Modus 3       100=Modus 4           101=Modus 5
BCD:   Zählformat
         0=binäre Zählung (Werte 0x0000 bis 0xffff)
         1=BCD-Zählung (Werte 0 bis 9999)
```

Auf die BCD-Zählung wird in diesem Treiber nicht weiter eingegangen, da die binäre Zählung verwendet wird. Um den Zählerstand auszulesen kann man direkt die I/O-Ports auslesen. Allerdings erhält man beim Lesen entweder des niederwertige oder das höherwertige Byte des 16 Bit-Zählerwertes. Um diese zwei Bytes konsistent auszulesen wird der Zähler-Latch-Befehl verwendet. Dieser legt den aktuellen Zählerstand in separaten Flipflops ab, welche nun konsistent über die gleichen I/O-Ports ausgelesen werden können.

Da alle Zähler den gleichen I/O-Port für das Steuerwort verwenden, ist es wichtig, dass das Programmieren bzw. Auslesen der Zähler serialisiert wird.

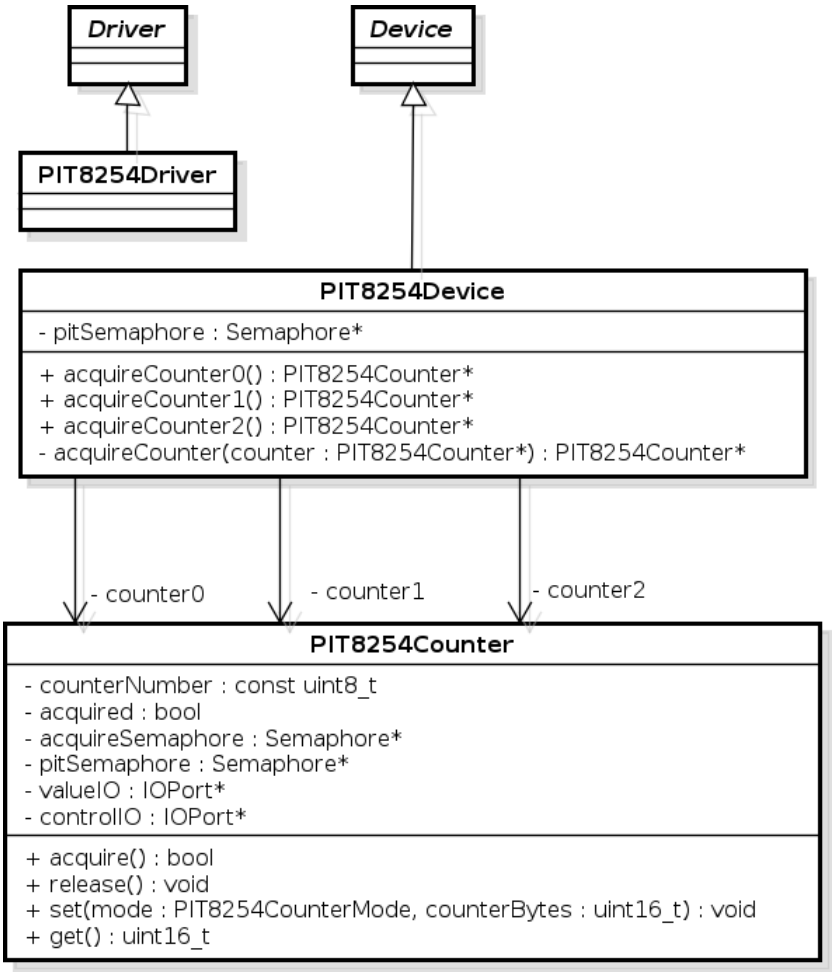


Abbildung 17.2: PIT8254 Klassenmodell

17.2 Entwurf

Der fertige Entwurf der Klassen ist in den Abbildungen 17.2 und 17.3 dargestellt. Device und Driver sind Klassen des Treiber-Frameworks. Die Klasse PIT8254Driver implementiert die geerbte abstrakte Methode checkDev() indem sie ein PIT8254Device-Objekt erzeugt. Das erzeugte Device wird über die geerbte Klasse automatisch dem DeviceManager bekannt gegeben.

Das Device hat zur Synchronisation aller Zähler eine Semaphore (pitSemaphore),

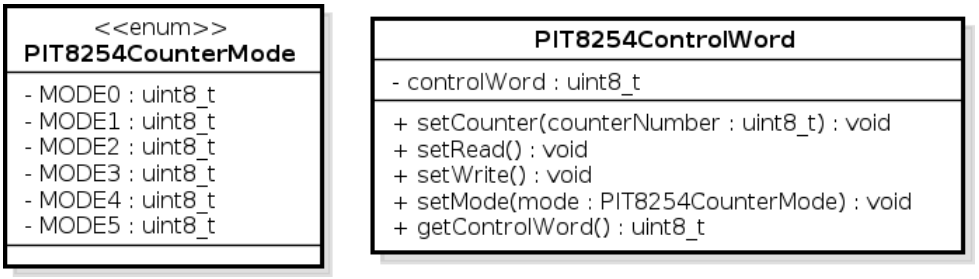


Abbildung 17.3: PIT8254 Klassenmodell Hilfsklassen

welche an alle Zähler übergeben wird. Hiermit wird sichergestellt dass jeweils nur ein Zähler gleichzeitig auf dem Controller arbeitet.

Damit jeder Zähler auch nur von einem Nutzer verwendet werden kann muss der entsprechende Zähler vom Device akquiriert werden. Der erste Aufrufer erhält den Zähler, jeder weitere Aufruf führt zu einer Fehlermeldung („fatalError“). Nach Verwendung kann der Zähler über die Methode release() wieder für andere Nutzer freigegeben werden.

Das Steuerwort des Intel 8254 wird über die Klasse PIT8254ControlWord abgebildet. Es wird zum Setzen (get()) und Auslesen (set()) der Zähler verwendet.

18 Echtzeituhr

18.1 Zeitquelle

Zur Umsetzung eines Zeitkonzepts muss zunächst eine Zeitquelle gefunden werden. Die naheliegende Lösung liegt in der Verwendung des PIT8254, für welchen bereits ein Treiber implementiert ist. Eine Alternative bietet die Echtzeituhr (Real Time Clock, RTC), die ebenfalls einen Interrupt generieren kann. Beide Ansätze werden im Folgenden analysiert und daraus ein Konzept für die Zeit entwickelt.

18.1.1 Der Zeitgeber des PIT8254

Der erste Zeitgeber (Timer0) des PIT8254 ist bereits als Treiber realisiert. Er wird vom Scheduler verwendet, um ein Zeitscheibenverfahren für Prozesswechsel zu realisieren. Somit ist allerdings die Frequenz des Zeitgebers bereits durch den Scheduler vorgegeben und vom Scheduling-Verfahren abhängig. Ein zusätzlicher Interrupt-Behandler kann auf dem IRQ des Zeitgebers registriert werden, um die aktuelle Laufzeit zu inkrementieren.

18.1.2 Echtzeituhr

Die Echtzeituhr (Real Time Clock, RTC) wird über das CMOS angesprochen. Bei dem CMOS handelt es sich hier nicht um die Technologie, sondern um ein 64 Byte großes, batteriegepuffertes SRAM, das vom BIOS genutzt wird um Informationen abzuspeichern. Somit läuft die Echtzeituhr auch wenn ein Computer heruntergefahren wurde oder die Stromversorgung verloren hat.

Die RTC bietet im Gegensatz zum PIT eine genauere Möglichkeit, um einen Interrupt auszulösen. Diese hat eine einen Bereich von 2 HZ bis 8 kHZ in 2er-Potenz-Schritten (also 2,4,8,16,...). Dadurch lässt sich die Zeit genauer errechnen als mit dem PIT.

18.2 Entwurf und Umsetzung

18.2.1 Klassenmodell

In Abbildung 18.1 sind die Klassen der Zeitmessung und des Wartens dargestellt. Für die Zeitmessung werden die Klassen `RTCHandler` (IRQ-Behandler zur Behandlung des RTC-Interrupt), `RTCDevice` (Implementierung der Echtzeituhr) und `Time` (Zeitzähler mit Operationen für externen Zugriff) verwendet. Um die aktuelle Systemzeit (in ms) zu ermitteln, genügt ein Aufruf der Operation `getTime()` im `Time`-Objekt. Die Initialisierung findet bei der Treiberinitialisierung bei Systemstart statt. Da die Frequenz des RTC-Interrupt konfigurierbar ist, aber nicht exakt eine Millisekunde beträgt, werden lediglich *Ticks* gezählt. Aus der RTC-Zeit bei Systemstart und der seither gezählten Ticks lässt sich so die aktuelle Zeit berechnen, ohne wiederholt die RTC-Zeit zu lesen. Die Länge eines Ticks ist abhängig von der eingestellten Frequenz des Interrupt und beträgt $31,25ms$ bei $32Hz$. Die Länge eines Tick bestimmt somit die Auflösung (Genauigkeit) der Zeitmessung.

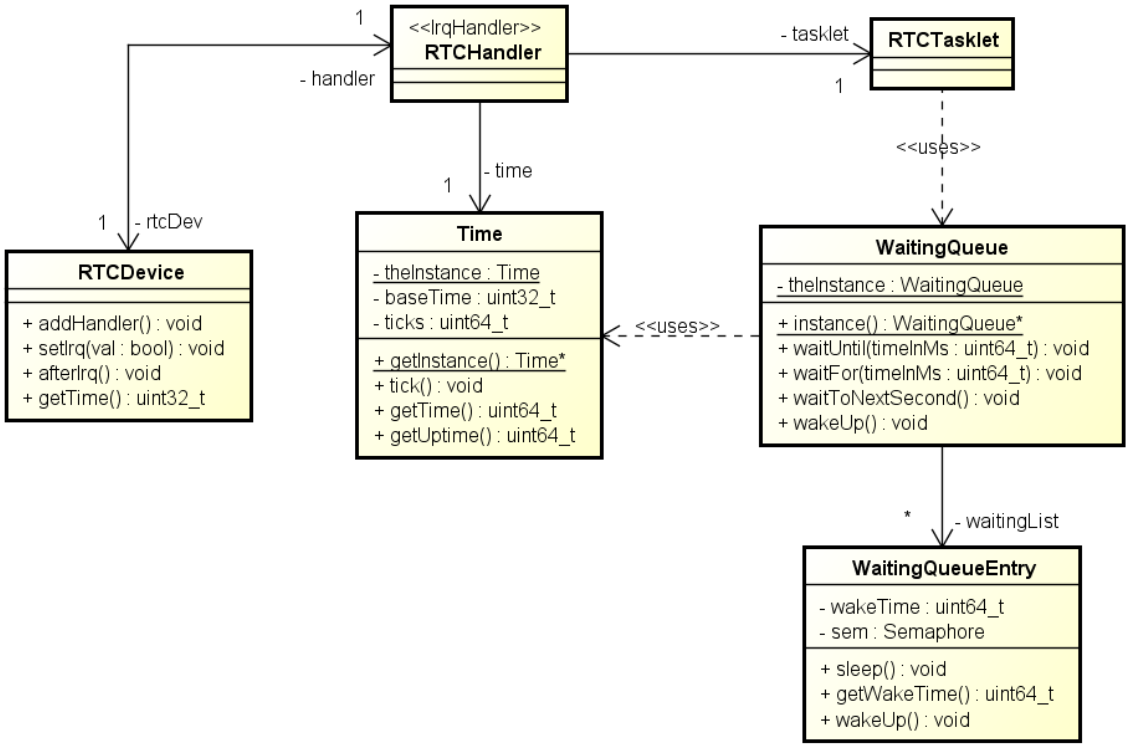


Abbildung 18.1: Klassenmodell zu Zeit und Warten

Mit der Verfügbarkeit einer Zeitmessung lässt sich auch das Warten bis zu einem Zeitpunkt realisieren. Dieser Zeitpunkt kann entweder absolut oder relativ (zur aktuellen Zeit) angegeben werden. Als Zeiteinheit dient die Millisekunde (ms). Hierzu werden zwei Klassen verwendet, eine geordnete Liste mit wartenden Threads (`WaitingQueue`) und ein Thread, der wartende Threads aufweckt. Ein

0x00	RTC seconds
0x01	RTC seconds alarm
0x02	RTC minutes
0x03	RTC minutes alarm
0x04	RTC hours
0x05	RTC hours alarm
0x06	RTC day of week
0x07	RTC day of month
0x08	RTC month
0x09	RTC year

Tabelle 18.1: RTC Bytes 0x00 – 0x09: Datum und Uhrzeit

Thread wartet, indem er z. B. die Operation `waitUntil(ms)` aufruft. Dies resultiert in einen neuen Listeneintrag, der neben der Weckzeit auch eine Semaphore enthält. Auf dieser wird anschließend `down()` ausgeführt. Der Thread wird geweckt (`semaphore.up()`), wenn nach einem Tick die Weckzeit erreicht ist, also die aktuelle Zeit kleiner oder gleich der Weckzeit ist.

Beachte: Da das Warten durch Semaphoren realisiert ist, darf ein Aufruf von `wait()` nur im PassiveLevel stattfinden.

18.2.2 Der RTC-Treiber

Wie bereits erwähnt wird die RTC über das CMOS angesprochen. Der Zugriff auf das CMOS wird über die Ports 0x70 (Control) und 0x71 (Data) realisiert. Der Control-Port kann nur beschrieben werden. Beim Lesen aus dem Control-Port wird immer 0xFF zurückgegeben. Über den Control-Port wird ausgewählt, welches Byte des CMOS am Data-Port zum Lesen oder Schreiben anliegen soll. Für die RTC sind die Bytes 0x00 – 0x0C reserviert. Bytes 0x00 – 0x09 sind für das aktuelle Datum und die aktuelle Zeit reserviert. Zudem kann hier ein Alarm festgelegt werden, auf den nicht weiter eingegangen wird. Die Belegung der Bytes ist in Tabelle 18.1 dargestellt. Die Bytes 0x0A – 0x0C sind für die Steuerung der RTC reserviert und werden im Folgenden näher beschrieben.

Byte 0x0A „RTC Register A“ (Tabelle 18.2). Dieses Byte steuert hauptsächlich die Frequenz, mit der der RTC-Interrupt (IRQ8) ausgelöst wird. Dazu gibt es eine Basisfrequenz, die sich mit 64 kHZ / base (Bits 4-6) errechnen lässt. Diese Frequenz wird nochmal durch einen Divisor geteilt. Der Divisor lässt sich aus 2^{rate} (Bits 0-3) errechnen. Standardmäßig ist base 2 und rate 6 eingestellt, wodurch der IRQ mit 1024 HZ ausgelöst wird. Das Bit 7 zeigt an, ob die RTC gerade die Uhrzeit aktualisiert.

Bit (LSB = 0)	Function	Default
0-3	rate	0110
4-6	base	010
7	update in progress	–

Tabelle 18.2: RTC Register A (Frequenzwahl)

Bit (LSB = 0)	Function	Default
0	daylight saving	0 = disable
1	24 hour mode	1 = 24h mode
2	time/date format	0 = BCD
3	square wave freq.	0 = disable
4	update endet interrupt	0 = disable
5	alarm interrupt	0 = disable
6	periodic interrupt	0 = disable
7	disable clock update	0 = update

Tabelle 18.3: RTC-Statusregister B

Byte 0x0B „RTC Register B“ (Tabelle 18.3). Dieses Byte dient als Konfiguration der RTC. Hierbei werden nur die verwendeten Bits beschrieben:

Bit 0 Zeigt an ob gerade Winter- oder Sommerzeit ist.

Bit 1 Steuert das Stundenformat (12 oder 24 Stunden).

Bit 2 Steuert die Interpretation des Datums und der Uhrzeit in den Bytes 0x00-0x09. Dabei gibt es die Wahl zwischen BCD und Binärcodierung.

Bit 4 Steuert, ob der Interrupt ausgelöst werden soll wenn die RTC das CMOS aktualisiert hat.

Bit 5 Ob der Alarm Interrupt ausgelöst werden soll.

Bit 6 Steuert den periodischen IRQ.

Bit 7 Steuert, ob die RTC das CMOS aktualisieren darf.

Byte 0x0C „RTC Register C“ (Tabelle 18.4). Beinhaltet die Information nach einem Interrupt, welche Art Interrupt ausgelöst worden ist. Dabei ist zu beachten, dass z. Z. jeder Interrupt als ein Tick interpretiert wird, da der IRQ-Handler nicht zwischen den verschiedenen Interrupt-Typen differenziert. Daher sollten die Bits 4 und 5 nicht gesetzt sein. Die folgenden Bits werden auf 1 gesetzt bei jeweiliger Art des Interrupts:

Bit 4 Wenn die RTC das CMOS aktualisiert.

Bit 5 Wenn der Alarm-Interrupt ausgelöst worden ist.

Bit 6 Wenn der periodische Interrupt ausgelöst worden ist.

Bit (LSB = 0)	Function
0-3	reserved
4	update ended interrupt
5	alarm interrupt
6	periodic interrupt
7	IRQF

Tabelle 18.4: RTC-Statusregister C (Read-Only)

Bit 7 Wenn einer der RTC-spezifischen Interrupts ausgelöst worden ist.

Wichtig: Das Statusregister C muss nach jeden RTC Interrupt (egal welche Art) ausgelesen werden, da sonst keine weiteren Interrupts von der RTC ausgelöst werden.

19 ATA Festplattentreiber

19.1 Einleitung

Für das FHDW-OS wird in dieser Phase ein Festplattentreiber entwickelt der Festplatten über den *ATA*[55] *PIO*[56] *Modus* ansteuern kann. Dieser wird von allen Festplatten bzw. Laufwerken unterstützt, die der ATA Spezifikation [9] genügen. Dabei wird die Addressierungsmethode *Logische Blockadressierung (LBA)* verwendet.

Der PIO Modus erzeugt auf der CPU eine hohe Auslastung, da er jedes Byte das zwischen dem Gerät und der CPU ausgetauscht wird über den IO Port der CPU senden muss. Bei Vollauslastung der CPU können manche CPUs eine Transfergeschwindigkeit von 16 MB pro Sekunde erreichen(, vorrausgesetzt die Geräte unterstützen PIO Mode 4, welcher ab der ATA-2 Spzezifikation unterstützt wird).

Für den Anfang unseres Betriebssystems stellt der PIO Modus jedoch eine einfache Möglichkeit da mit Festplatten Daten auszutauschen zu können, ohne sich mit der komplizierteren Speicherzugriffsart *Direct Memory Access (DMA)* auseinandersetzen zu müssen.

Die LBA bietet – im Gegensatz zu der Zylinder-Kopf-Sektor-Adressierung – die Möglichkeit die Blöcke der Festplatte unabhängig von der Geometrie der Festplatte zu adressieren, was den Zugriff und die Verwaltung vereinfacht. Ferner kann der Festplattenzugriff mithilfe von Interrupts oder Polling implementiert werden. Um die Multitaskingfähigkeit des Betriebssystems weiter zu unterstützen, werden Interrupts eingesetzt.

[55] Advanced Technology Attachment
[56] Programmed input/output

19.2 Analyse

Die ATA Festplatten Spezifikation baut auf der alten ST506-Schnittstelle[57] auf. In der ST506 Spezifikation wurde jede Festplatte zu einer Kontrollerplatine mit einem Daten- und einem Komandokabel verbunden. Diese Platine wurde dann mit dem Bus der Hauptplatine verbunden. Die Kommunikation fand daher durch die IO Ports der CPU statt.

Die ATA Spezifikation verlagerte den Kontroller zur Festplatte. Beim Zugriff auf einen Festplatten IO Port wurden die CPU IO Ports direkt mit den Festplatten IO Ports verbunden. Dieser Vorgang wird durch einen Chip gesteuert. Der Datenaustausch zwischen der CPU und dem Kontroller ist gleich geblieben und nennt sich PIO Modus.

19.2.1 Logische Blockadressierung (LBA)

Um beim Datenaustausch Blöcke eindeutig adressieren zu können, kann die LBA verwendet werden. Dabei werden die Blöcke beginnend bei Null gezählt. Jeder LBA-Block entspricht einem einzelnen Sektor der älteren Zylinder-Kopf-Sektor-Adressierung (CHS), die Zylindernummern, Leseköpfe und Sektoren unterscheidet. LBA unterscheidet zwischen Adressen mit 28 und 48 Bit. Durch eine 28 Bit lange LBA-Adresse wird die Adressierung von 268.435.456 Blöcken ermöglicht. Bei üblicher Block- und Sektorgröße von 512 Byte – die durch die Festplatte festgelegt wird–, entspricht das 128 GiB. Für unser Betriebssystem verwenden wir vorerst die 28 Bit Variante.

19.2.2 Primärer und Sekundärar Bus

Heutige Festplattenkontroller unterstützen zwei ATA Busse pro Chip, welche sich über eine festgelegte Menge an IO Ports ansteuern lassen. Der erste/primäre ATA Bus wird durch die IO Ports 0x1F0 bis 0x1F7 und dem Gerätekontroll-/Alternativestatusregister an IO Port 0x3F6 gesteuert. Er löst den IRQ 14 aus. Der zweite/sekundäre ATA Bus wird durch die IO Ports 0x170 bis 0x177 und dem Gerätekontroll-/Alternativestatusregister an IO Port 0x376 gesteuert. Er löst den IRQ 15 aus.

[57] Die ST506-Schnittstelle wurde 1982 von der Firma Seagate entworfen und war lange Zeit der Standard.

19.2.3 Master/Slave Laufwerke

Über jeden Bus kann mit maximal zwei Laufwerken – dem Master- und dem Slavelaufwerk – kommuniziert werden. Dabei verhalten sich beide Laufwerke fast identisch, wobei die Laufwerke bei der Ansteuerung durch ein spezielles IO Port Bit unterschieden werden können.

19.2.4 Register

Innerhalb der ATA Spezifikation werden viele Register beschrieben, durch die der Datenaustausch mit den Laufwerken über die Busse stattfinden kann. Ein Überblick über die Register für den primären ATA-Bus gibt Tabelle 19.1 – die Register des sekundären ATA Busses folgen dem gleichen Schema mit einer anderen Startadresse.

Tabelle 19.1: Register

Register	Adresse	Breite[Bit]	Lesen(R)/Schreiben(W)
Datenregister	1f0h	16	R/W
Fehlerregister	1f1h	8	R
Sektorenanzahlregister	1f2h	8	R/W
LBA Bits 0-7	1f3h	8	R/W
LBA Bits 8-15	1f4h	8	R/W
LBA Bits 16-23	1f5h	8	R/W
Geräteregister/LBA Bits 24-27	1f6h	8	R/W
Statusregister	1f7h	8	R
Befehlsregister	1f7h	8	W
Alternatives Statusregister	3f6h	8	R
Digitales Ausgaberegister	3f6h	8	W
Laufwerkadressregister	3f7h	8	R

Nachfolgend werden die wichtigsten aufgeführt und erklärt.

Statusregister

Das Statusregister in Tabelle 19.2 enthält Statusinformationen über das Gerät. Das Statusregister wird stets aktualisiert um den Status des Geräts preiszugeben und über den Fortschritt des aktuellen Befehls zu informieren. Wenn das BSY Bit auf 0 gesetzt ist, sind die anderen Bits im Register gültig und die anderen Register enthalten sinnvolle Daten.

nw(nicht wichtig) wird für die aktuelle Ausbaustufe des Betriebssystems nicht benötigt.

Tabelle 19.2: Statusregister

7	6	5	4	3	2	1	0
BSY	DRDY	DF	nw	DRQ	nw	nw	ERR

BSY(Busy) wird gesetzt, wenn das Gerät die Kontrolle über die Register hat. Wenn das Bit gesetzt ist, wird das Beschreiben des Register verhindert, bzw. ignoriert. Das Gerät sollte nicht den Status des DRQ Bits ändern solange das BSY nicht auf 1 gesetzt wurde. Sobald der letzte Block der PIO Daten in dem Befehl transferiert wurde, wird das DRQ Bit auf 0 gesetzt ohne das BSY Bit zusetzen.

DRDY(Device Ready) wird gesetzt, sobald das Gerät bereit ist Befehle zu empfangen.

DF(Device Fault) zeigt an, dass ein Gerätefehler detektiert wurde.

DRQ(Data Request) zeigt an, dass das Gerät bereit ist Daten auszutauschen.

ERR(Error) gibt an, dass ein Fehler beim Ausführen des letzten Befehls entstanden ist.

Alternatives Statusregister

Das Alternative Statusregister in Tabelle 19.3 enthält die gleichen Informationen wie das Statusregister. Der einzige Unterschied ist, dass das Lesen aus dem Register keinen Empfang des Interrupts oder ein Beenden eines wartenden Interrupts impliziert.

Tabelle 19.3: Alternatives Statusregister

7	6	5	4	3	2	1	0
BSY	DRDY	DF	DSC	DRQ	CORR	IDX	ERR

Befehlsregister

Das Befehlsregister enthält die Befehle, die zu den Geräten gesendet werden. Die Ausführung eines Befehls beginnt unverzüglich, nachdem in dieses Register geschrieben wurde. Die wichtigsten Befehle für den LBA PIO Mode zusammen mit ihren Befehlscodes sind in **??** zu sehen. Die dargestellten Befehle benötigen korrekt gesetzte LBA Bits und eine Sektorenanzahl bevor sie durchgeführt werden können.

Tabelle 19.4: Befehle

Befehl	Befehlscode
Sektor lesen	2h
Sektor schreiben	3h
Sektor verifizieren	4h
Laufwerk Identifizieren	ech

Datenregister

Das Datenregister ist, abhängig von dem Typ der Daten die von dem aktuellen Befehl gesendet werden sollen, 8 oder 16-Bit breit.

Gerätekontrollregister

Das Gerätekontrollregister in Tabelle 19.5 enthält reservierte Bits(r), sowie die Bits SRST(1=Reset der Geräte), welches einen Software-Reset des Busses und der an den Bus angeschlossenen Geräte durchführt. Sowie das nIEN(0= Aktiviert die Interrupts.) Bit, welches die Interrupt, die durch den Bus-Controller ausgelöst werden, aktiviert oder deaktiviert.

Tabelle 19.5: Gerätekontrollregister

7	6	5	4	3	2	1	0
r	r	r	r	r	SRST	nIEN	0

Geräte-/LBA-Register

Das Geräte/Kopfregister in Tabelle 19.6 enthält Informationen über die Addressirungsart, sowie Adressinformationen. Wenn das Bit L auf 1 steht, wird der LBA Adressierungsmodus aktivert. DEV 0/1 selektiert das Master- respektive das Slavegerät.

Tabelle 19.6: Geräte/Kopfregister

7	6	5	4	3	2	1	0
r	L	r	DEV	LBA 27	LBA 26	LBA 25	LBA 24

Fehlerregister

Das Fehlerregister in Tabelle 19.7 enthält Diagnoseinformationen des zuletzt ausgeführten Befehls.

Tabelle 19.7: Fehlerregister

7	6	5	4	3	2	1	0
r	UNC	nw	nw	nw	ABRT	nw	nw

nw(nicht wichtig) wird für die aktuelle Ausbaustufe des Betriebssystems nicht benötigt.

UNC(Uncorrectable Data Error) bezeichnet einen unkorrigierbaren Datenfehler.

ABRT(Aborted Command) gibt an, dass der Befehl abgebrochen wurde, z.B. weil der Befehlscode ungültig war.

Sektoranzahlregister

Das Sektoranzahlregister enthält die Anzahl der Sektoren die gelesen oder geschrieben werden sollen. Ein Wert von 0 bedeutet eine Anzahl von 256 Sektoren. Nachdem ein Befehl erfolgreich ausgeführt wurde, sollte der Wert 0 sein. Sollte der Befehl durch einen Fehler unterbrochen werden, enthält das Register die Anzahl der noch zu übertragenden Sektoren.

Digitales Ausgaberegister

Durch die Bits im digitalem Ausgaberegister in Tabelle 19.9 lassen sich durch IEN=0 Interrupts aktivieren und durch SRST=1 ein Softwarereset durchführen.

Tabelle 19.8: Digitales Ausgaberegister

7	6	5	4	3	2	1	0
x	x	x	x	x	SRST	IEN	X

Laufwerkadressregister

Mithilfe des Laufwerkadressregistern, kann festgestellt werden welches Laufwerk und welcher Kopf aktive und ausgewählt ist. Ist das WTGT-Bit gelöscht, werden gerade Daten auf die Platte geschrieben. Die vier Bits HS3-HS0 geben dabei den aktiven Kopf (als 1-Komplement aufgefasst) an. Analog bezeichnen DS1-DS0 das ausgewählte Laufwerk.

Tabelle 19.9: Laufwerkadressregister

7	6	5	4	3	2	1	0
x	WTGT	HS3	HS2	HS1	HS0	DS1	DS0

19.2.5 Befehle

Beim Ausführen von Befehlen mit Hilfe des Befehlsregisters müssen einige grundlegende Anforderungen berücksichtigt werden. Zum einen müssen die Register die von dem Befehl benutzt werden zuerst mit entsprechenden Werten gefüllt werden. Zum anderen müssen Wartezeiten eingehalten werden, damit die Festplatte genug Zeit hat um z.B. den Schreibkopf zu positionieren.

Nachfolgend werden die drei wichtigsten Befehle mit ihren Anforderungen vorgestellt.

Laufwerk identifizieren

Durch den Befehl *Laufwerk identifizieren (ech)* können Informationen über das adressierte Laufwerk ausgelesen werden. Der Ablauf gestaltet sich folgendermaßen:

1. Befehlsübergabe
2. Satz des BSY-Bit und Puffervorbereitung durch den Controller
3. Entfernen des BSY-Bit durch den Controller
4. Satz des DRQ-Bit durch den Controller
5. Auslösen des Festplatten IRQ
6. Auslesen der 256 Datenworte aus dem Sektorpuffer möglich

Die wichtigsten Datenworte 60 und 61 als DWORD interpretiert liefern die Anzahl an 28 bit LBA addressierbaren Sektoren.

Sektor lesen

Durch den Befehl *Sektor lesen(2xh)* können je nach dem Wert im Sektoranzahlregister 1 bis 256 Sektoren gelesen werden. Dafür muss die LBA Adresse korrekt gesetzt werden bevor der Befehl abgeschickt wird.

Sobald der Befehel gesendet wurde, wird das BSY-Bit gesetzt und die Laufwerkpositionnierung wird durchgeführt. Nach dem Einlesen eines Sektors wird das

DRQ-Bit gesetzt und ein Festplatteninterrupt ausglöst. Jetzt können die Daten gelesen werden. Soll mindestens ein weiterer Sektor gelesen werden, setzt der Controller erneut das BSY-Bit, löscht das DRQ-Bit und liest den nächsten Sektor ein.

Sektor schreiben

Durch den Befehl *Sektor schreiben(3xh)* können je nach dem Wert im Sektoranzahlregister 1 bis 256 Sektoren geschrieben werden. Dafür muss die LBA Adresse korrekt gesetzt werden bevor der Befehl abgeschickt wird.

Sobald der Befehl gesendet wurde, wird das BSY-Bit gesetzt und die Laufwerkpositionnierung wird durchgeführt. Sobald das geschehen ist, aktiviert der Controller das DRQ-Bit und löscht das BYS-Bit. Nun können die Sektordaten in das Datenregister übergeben werden. Sofern die Daten übergeben wurden, löscht der Controller das DRQ-Bit und setzt das BSY-Bit erneut. Jetzt werden die Schreibdaten auf die Platte geschrieben.

Sollte ein weiterer Sektor geschrieben werden, setzt der Controller das DRQ-Bit, löscht das BSY-Bit und löst einen Festplatteninterrupt aus. Jetzt können weitere Daten übergeben werden.

19.3 Entwurf

Um den Festplattentreiber zu realisieren, müssen einige Komponenten der Infrastruktur des Betriebssystems ausgenutzt werden. Dazu gehört

- die Schnittstelle für Treiber und Geräte, damit die Laufwerke und Busse als Geräte aufgefasst werden können,
- die Interruptbehandlung, damit auf die – durch die Befehle ausgelösten – Interrupts reagiert werden kann, um z.B. den nächsten Sektor zu übertragen, und
- die Taskletverarbeitung, damit aufwendigere Operationen nicht durch den Interrupt, sondern durch ein entsprechendes Tasklet durchgeführt werden können.

Ferner bietet sich eine Art *Active-Object*[58] Entwurfsmuster an, damit Befehle asynchron ausgeführt werden können. Dies sorgt für eine bessere Multitaskingfähigkeit und Geschwindigkeit des Systems. So wird bei einem Schreibbefehl ein

[58] Näheres zum *Active-Object* Entwurfsmuster in [**?**]

Schreibkommando erzeugt, welches erst durch den Aufruf des Tasklets ausgeführt wird. Dadurch wird das aktive Warten auf den Schreibbefehl verhindert.

19.3.1 Klassendiagramm

In Abbildung 19.1 ist ein Klassendiagramm dargestellt, dass den Aufbau des Treibers und der Kommandos visualisiert. Die Kommunikation mit dem Geräte

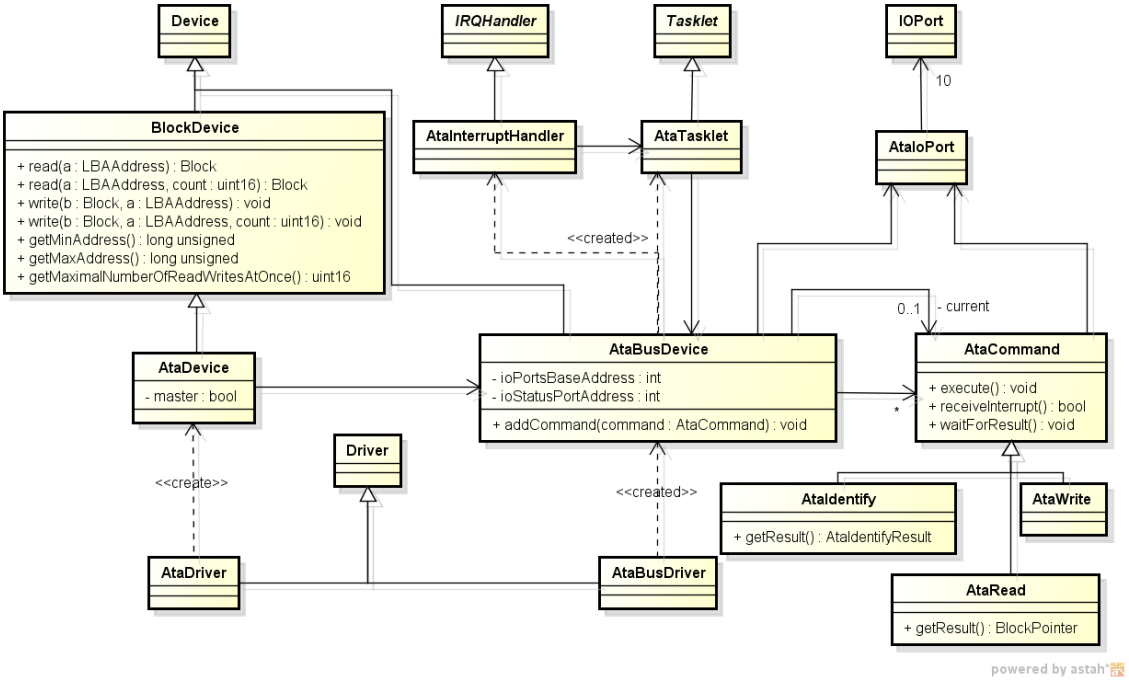


Abbildung 19.1: Klassendiagramm

welches an einen ATA-Bus angeschlossen ist findet über die Kommandos statt, die erzeugt werden. Dazu werden z.B. die Schreibkommandos mit entsprechenden Attributen (Sektoranzahl, Zeiger auf die Daten und LBA-Adresse) erzeugt, die dann durch ein Tasklet weiterverarbeitet werden. Kommandos die ein Ergebnis liefern speichern dass Ergebnis selbst und bieten Operationen an, mit denen das Ergebnis ausgelesen werden kann, sobald es zur Verfügung steht. Das Warten auf das Ergebnis kann sowohl aktiv oder passiv, mithilfe einer Semaphore, durchgeführt werden.

19.3.2 Aufbau der Treiber

Zur Ansteuerung der Festplatten wurden neue Geräte und Treiber zum Treiber-Framework des FHDW-OS hinzugefügt. Dabei wurden zwei Treiber und Geräte Paare erstellt:

- Das erste Paar repräsentiert jeweils einen ATA–Bus. Die Geräte vom Typ AtaBusDevice übernehmen die Abarbeitung der Kommandos. Dabei wird darauf geachtet, dass jeweils immer nur Kommando auf dem Bus abgearbeitet wird. Der Treiber für den ATA-Bus erstellt die einzelnen Geräte.
- Das zweite Paar repräsentiert jeweils ein Geräte welches an einen ATA–Bus angeschlossen wird. Die Geräte werden durch Objekte vom Typ AtaDevice repräsentiert. Das AtaDevice bietet sämtliche derzeit unterstützenden Operationen an. Zur Umsetzung nutzt das AtaDevice das entsprechende AtaBusDevice in dem es ein Kommando übergibt. Außerdem wartet das AtaDevice bis das Kommando abgearbeitet wurde und gibt falls vorhanden das Ergebnis zurück.

19.4 Fazit

Mithilfe des Festplattentreibers ist jetzt möglich Daten auf Festplatten zu schreiben. Insgesamt lassen sich bis zu vier Festplatten unterstützen. Dann sind der primäre und der sekundäre Bus belegt. An jedem der beiden Busse hängt dann ein Master- und ein Slavelaufwerk.

Aufgrund der Art wie der PIO Modus zwischen der CPU und den Festplattencontrollern kommuniziert, lassen sich große Datenmengen nur langsam transportieren. In einer zukünftigen Ausbaustufe des Betriebssystems könnten Festplatten schneller über den DMA-Controller angesprochen werden.

Teil VI

Allgemeines

Im Projektverlauf wurden weitere Dokumente erstellt, die nur indirekt in die Entwicklung des FHDW-OS einfließen. Dazu gehören z. B. eine Übersicht über die Unterschiede von C++ zu Java, verwendete Code-Konventionen aber auch die Erstellung des vorliegenden Handbuchs. Diese Kapitel sind im folgenden, allgemeinen Teil des Handbuchs integriert.

20 Programmieren in C++

In diesem Kapitel wird C++ vorgestellt. Dabei wird hauptsächlich auf die Unterschiede zu Java und die Eigenarten von C++ eingegangen.

20.1 Objektzugriff

Die Programmiersprache Java lässt einem keine Kontrolle darüber, ob auf ein Objekt direkt oder über eine Referenz zugegriffen wird. Der Zugriff auf Objekte geschieht in Java immer über eine Referenz und auf elementare Datentypen immer direkt. Dieses implizite Referenzen-Konzept gibt es in C++ nicht. Stattdessen hat man völlige Freiheit darüber, wie auf eine Variable zugegriffen werden soll.

20.1.1 Referenzen

Referenzen sind Verweise auf andere Entitäten. Beim Anlegen von Referenzen müssen diese sofort mit einem Objekt initialisiert werden. Diese Bindung kann nicht geändert werden.

Mit Ausnahme der Initialisierung gibt es keine Operationen die direkt auf Referenzen ausgeführt werden können. Sämtliche Operationen werden immer mit der Entität, auf welche die Referenz verweist, ausgeführt.

Die Deklaration einer Referenz besteht aus dem Typ des Zielobjekts, gefolgt vom Referenzoperator (&) und dem Namen der Referenz. Die Regeln für die Benennung von Referenzen sind die gleichen wie für Variablennamen. Siehe Listing 20.1.

Listing 20.1: Nutzung von Referenzen

```
1 int zahl = 15;
2 int& referenz = zahl;
3 referenz += 15;
4 std::cout << zahl << "\n"; // Ausgabe: 30
```

Ein mögliches Einsatzgebiet für Referenzen ist die Wertübergabe als Referenz (call by reference). Standardmäßig werden Objekte in C++ immer mit call by value übergeben. Das heißt, beim Aufruf der Funktion werden alle Parameter (auf den Stack) kopiert und die Funktion nutzt nur Kopien, weshalb Änderungen an den Parametern nach Beenden der Funktion weg sind. Listing 20.2 soll dies veranschaulichen.

Listing 20.2: Call by value/reference

```
#include <iostream>

void callByValue(int y) // lokale Kopie von x
{
  y += 10;
}

void callByReference(int& y) // Referenz auf x
{
  y += 10;
}

int main()
{
  int x = 0;
  callByValue(x);
  std::cout << x << "\n"; // Ausgabe: 0
  callByReference(x);
  std::cout << x << "\n"; // Ausgabe: 10
  return 0;
}
```

20.1.2 Zeiger

Zeiger (auch Pointer genannt) sind ebenfalls Verweise auf Daten, die an einer anderen Stelle stehen. Physikalisch gesehen sind Zeiger allerdings Objekte die als Wert die Speicheradresse einer anderen Variable enthalten. Da Zeiger reale Objekte sind, können sie im Gegensatz zu einfachen Referenzen ausgelesen und modifiziert werden. Aus diesem Grund nennt man sie auch veränderbare Referenzen. Bei der Deklaration einer Zeigervariable wird der Typ der Variable festgelegt, auf den sie verweisen soll.

Zeiger werden deklariert, indem ihrem Namen jeweils ein Stern vorangestellt wird. Dabei muss beachtet werden, dass bei der Initialisierung bereits der Typ der Variable, auf die verwiesen werden soll, angegeben werden muss. Nach der Deklaration kann der Zeiger nun auf einen bestimmten Speicherbereich verweisen. Dazu muss die Speicheradresse einer Variable mithilfe des Adressoperators (&) ermittelt werden.

Für den Zugriff auf ein referenziertes Objekt muss der Zeiger mit Hilfe des Operators * dereferenziert werden. Listing 20.3 zeigt die beispielhafte Verwendung von Zeigern. Arbeitet man mit Zeigern auf ein Objekt, so kann man die Member des Objektes über das Dereferenzieren aufrufen: „(∗objektZeiger).methode();“. Der Einfachheit halber gibt es in C++ jedoch auch einen Operator, der das Dereferenzieren (*-Operator) und den Aufruf eines Members (.-Operator) kombiniert: „objektZeiger−>methode();“.

Listing 20.3: Zeiger verwenden

```
1  char alphabet[27]; // ein Array vom Typ char mit 27 Elementen
2  char* position;    // ein Pointer auf einen einzelnen char
3
4  position = &alphabet[0]; // der Pointer wird auf die Adresse
5                           // des ersten Elements festgelegt
6  char buchstabe = 'A';
7  for(int i = 0; i < 26; i++)
8  {
9    // Wir weisen der Stelle, auf die der Pointer zeigt, einen
10   // Buchstaben zu. Danach wird die Adresse, auf die der
11   // Pointer zeigt, erhoeht.
12   *position = buchstabe;
13   position++;
14
15   buchstabe++; // naechster Buchstabe im Alphabet
16 }
17
18 *position = 0; // letztes Element muss 0 sein
19 std::cout << alphabet << "\n"; // Ausgabe: "ABC..XYZ";
```

20.2 Die Wahl des Speichers

In C++ hat man im Vergleich zu Java eine größere Kontrolle über die Speicherverwaltung, aber dadurch besteht auch ein höheres Risiko für Fehler. Es wird zwischen zwei verschiedenen Arten von Speicher unterschieden: dem „Stack“ und dem „Heap“.

20.2.1 Der Stapelspeicher – Stack

Die Verwaltung des Stacks ist deutlich einfacher, da sich der Compiler um die ordnungsgemäße „Zerstörung“ der Objekte kümmert, dies ist beim Heap nicht der Fall. Der Heap sollte nur dann zum Einsatz kommen, wenn die Situation es auch wirklich erfordert.

Bei einem Stack werden die Variablen hintereinander im Speicher abgelegt. Listing 20.4 zeigt das Anlegen von Variablen auf dem Stack. Die Position des Stacks im Speicher und in welcher Reihenfolge die Elemente auf dem Stack abgelegt werden differiert je nach Implementation des C++-Standards. Je nach Optimierung ist es auch möglich, dass Elemente erst gar nicht in den Stack kopiert werden, wenn sie später nicht genutzt werden.

Listing 20.4: Speichern auf dem Stack

```
1 int i;
2 double d;
3 char c;
```

Der Speicher für diese Variablen wird sofort freigegeben, sobald der Gültigkeitsbereich der Variable verlassen wird, also spätestens nach dem Verlassen der Methode. Dies ist ein gravierender Unterschied zu Java, denn dort wird der Speicher zu einem undefinierten späteren Zeitpunkt durch den „Garbage Collector“ freigegeben, sobald das Objekt nicht mehr benötigt wird. Falls ein „Destruktor“ (siehe Abschnitt 20.4.1) vorhanden ist, wird dieser automatisch vor der Freigabe des Speichers ausgeführt.

20.2.2 Der Haldenspeicher – Heap

Der Heap wird besonders dann benötigt, wenn mit Datenstrukturen gearbeitet wird, deren Größe von vornherein nicht bekannt ist oder wenn man das gleiche Objekt über mehrere Klassen hinweg referenzieren möchte (siehe hierzu

auch Zeiger, Abschnitt 20.1.2). Der Heap wird auch als „dynamischer Speicher“ bezeichnet.

Wie bereits zuvor beschrieben, ist diese Art der Speicherverwaltung etwas komplizierter und fehleranfälliger, denn vom Heap muss alles explizit angefordert und freigegeben werden. Hierfür stellt C++ die Schlüsselwörter „new“ und „delete“ bereit, mit denen es möglich ist Speicher anzufordern und freizugeben. Bei komplexen benutzerdefinierten Klassen führt der new-Operator zwei Schritte aus:

- Einen Heap-Speicherbereich angemessener Größe anfordern
- Konstruktoren aufrufen, die den Speicher ordnungsgemäß initialisieren

Der delete-Operator führt genau die umgekehrten Schritte aus:

- Destruktoren aufrufen, um den Speicher zu bereinigen
- Speicherbereich freigeben

Ein Beispiel der Speichernutzung zeigt Listing 20.5. In Java wird grundsätzlich der Operator new verwendet, da in Java alle Objekte auf dem Heap liegen. Ein delete ist nicht Notwendig, da Java das delete selbstständig ausführt, wenn das Objekt nicht mehr referenziert wird.

Listing 20.5: Stack und Heap

```
1  #include <iostream> // wird fuer die Konsolen-Ausgabe benoetigt
2
3  class Person
4  {
5  public:
6    Person() { std::cout << this << "␣wird␣erstellt.\n"; }
7    ~Person() { std::cout << this << "␣wird␣bereinigt.\n"; }
8    void tuWas() { }
9  };
10
11 void stack()
12 {
13   Person person;
14   person.tuWas();
15   // Hier muss nicht aufgeraeumt werden, nach Verlassen der
16   // Methode wird person automatisch entfernt.
17 }
18
```

```
19 void heap()
20 {
21   Person* personZeiger = new Person();
22   personZeiger->tuWas();
23   delete personZeiger;
24 }
25
26 int main()
27 {
28   stack();
29   heap();
30   return 0;
31 }
32
33 /* Ausgabe:
34 0x7fffbee61dff wird erstellt.
35 0x7fffbee61dff wird bereinigt.
36 0xe81010 wird erstellt.
37 0xe81010 wird bereinigt.
38 */
```

20.2.3 Erstellen von Objekten an feste Adressen

Hat man bereits einen Speicherbereich im Heap und möchte in diesem ein neues Objekt anlegen, nutzt man die Placement-Syntax des new-Operators. Listing 20.6 zeigt die Verwendung. Der im Beispiel verwendete void-Zeiger kann auf jedes Objekt zeigen. In Java wäre es vom Typ `Object`.

Listing 20.6: Placement new

```
1 void* speicherBereich = (void*)0x1000;
2 Person* personZeiger = new(speicherBereich) Person;
3 personZeiger->~Person(); // Aufruf des Destruktors
```

Ein „Placement-Delete'"gibt es nicht. Nach Benutzung muss der Speicher mit Hilfe eines expliziten Destruktor-Aufrufs freigegeben werden. Destruktoren werden im Abschnitt 20.4.1 behandelt.

20.3 Header und Implementierung

In C++ gibt es zwei Arten von Quelldateien. Normale C++-Dateien haben die Endung cpp, c++ oder cc.

Header-Dateien haben die Endung h oder hpp. Sie haben in C++ zwei Aufgaben. Zum einen trennen sie die Deklaration von der Implementierung. Prinzipiell ist die Header-Datei als eine Art Schnittstelle zu verstehen. Sie zeigt alle möglichen Funktionen, Klassen, Operationen und so weiter. Möchte man eine Bibliothek anbieten, reicht es somit, nur die Header-Dateien auszuliefern. Die Implementierung kann in bereits kompilierter Form verteilt werden.

Die zweite Aufgabe ist die Deklaration (die Bekanntmachung) von Klassen, ohne diese sofort zu Implementieren. Dies ist nötig, da der C++-Compiler eine Quelltextdatei beim Kompilieren grundsätzlich von oben nach unten liest. Wird im Programm eine Funktion aufgerufen, welche vorher (also physikalisch in der Datei) nicht definiert wurde, gibt der Compiler einen Fehler aus.

Listing 20.7: Fehler: Aufruf der Funktion ohne Deklaration

```
1  // void f();
2
3  int main()
4  {
5    f(); // Fehler! Die Funktion f ist dem Compiler noch nicht
6         // bekannt.
7    return 0;
8  }
9
10 void f()
11 {
12   // tu etwas
13 }
```

In C++ sollten deshalb alle Funktionen vor ihrer Implementierung deklariert werden. In Listing 20.7 müsste man die erste Zeile auskommentieren. Zu beachten ist das Semikolon anstelle der geschweiften Klammern. Diese Art von Deklaration wird „forward declaration“ genannt.

Das in Listing 20.7 gezeigte Beispiel kann man nun auch mit einer Header-Datei realisieren. Das folgende Listing zeigt dies.

Listing 20.8: Forward declaration mit Hilfe von Header-Dateien

```
1  // Datei main.cpp
2  #include "f.h"
3
4  int main()
5  {
6    f();
7    return 0;
8  }
9  //==========================
10 // Datei f.h
11 #ifndef F_H
12 #define F_H
13
14 void f();
15
16 #endif
17 //==========================
18 // Datei f.cpp
19 #include "f.h"
20 void f()
21 {
22   // tu etwas
23 }
```

In Zeile 2 wird die Datei `f.h` inkludiert. In C++ wird der Inhalt der Datei `f.h` komplett in die Datei `main.cpp` kopiert. Die Zeilen 11, 12 und 16 sorgen dafür, dass die Deklaration von **void** f() nur einmal gelesen wird, auch wenn die Datei in mehreren Quelltextdateien verwendet wird. Den Präprozessor-Befehl „#include“ kann man entweder mit Anführungszeichen (suchen im aktuellen Verzeichnis) oder mit eckigen Klammern (suchen im System-Verzeichnis für Header-Dateien) nutzen.

In Java ist diese Art der Deklaration nicht nötig, da der Java-Compiler den Quelltext nicht nur stupide von oben nach unten liest.

20.4 Klassendeklarationen

In C++ werden Klassen üblicherweise in Header und Implementierung aufgeteilt.

Der Header enthält die Deklaration der Klasse. In C++ sind grundsätzlich alle

global[59] definierten Klassen öffentlich. Im Gegensatz zu Java muss man den Sichtbarkeitsmodifizierer (`public`, `protected` und `private`) nicht an jedes Element binden. Setzt man einen Modifizierer, so gilt dieser so lang, bis er durch einen neuen Modifizierer ersetzt wird. Listing 20.9 und 20.10 zeigen eine beispielhafte Klasse. Klassendeklarationen werden in C++ grundsätzlich mit einem Semikolon beendet. Bei der Implementierung ist darauf zu achten, dass man sich nicht mehr im Kontext der Klasse befindet und deshalb vor jeden Methodennamen den Namen der Klasse, gefolgt vom Qualifizierungsoperator (::) schreibt. Innerhalb jeder Klassenmethode gibt es, genau wie in Java, immer einen Zeiger auf das aktuelle Objekt. Er heißt, genau wie in Java `this`.

Listing 20.9: Klassendeklaration - Datei `person.h`

```
#ifndef PERSON_H
#define PERSON_H
class Person
{
public: // fuer alle zugreifbar
  // Konstruktor und Destruktor
  Person();
  ~Person();

  // Methoden
  void f();
  void g();

  // Zugriffsfunktionen
  int getAlter() const;
  void setAlter(int neuesAlter);

protected: // fuer abgeleitete Klassen zugreifbar
  void h();

private: // fuer niemanden zugreifbar
  int alter;
}; // <— sehr wichtig! C++ erwartet nach der Deklaration einer
   // Klasse ein Semikolon.
#endif // PERSON_H
```

[59] „Global“ heißt, dass die Klasse nicht in einem anderen Gültigkeitsbereich liegt. Zum Beispiel könnte man eine Klasse innerhalb einer anderen Klasse, oder sogar innerhalb einer Methode, definieren. Beides wäre nicht „global“.

Listing 20.10: Implementierung - Datei `person.cpp`

```
#include "person.h"
#include <iostream>  // cout

using namespace std;
// Hierdurch wird der Namespace std als Standard-Namespace
// fuer das gesamte Dokument verwendet.
// (cout statt std::cout, usw.)

Person::Person()
{
  cout << this << " wird erstellt.\n";
}
Person::~Person()
{
  cout << this << " wird bereinigt.\n";
}
void Person::f() {}
void Person::g() {}
int Person::getAlter() const
{
  return this->alter;
}
void Person::setAlter(int neuesAlter)
{
  this->alter = neuesAlter;
}
void Person::h() {}
```

Diese Aufteilung in Header und Implementierung ist aber nicht zwingend vorgeschrieben. Man kann seine Klasse auch komplett (oder nur zum Teil) in der Header-Datei implementieren. Oft kommt es vor, dass get- und set-Operationen auf Grund ihrer Einfachheit direkt in der Header-Datei implementiert werden.

20.4.1 Konstruktor und Destruktor

Konstruktoren verhalten sich in C++ grundsätzlich wie in Java. Jedoch bietet C++ ein weiteres Feature: Initialisierungslisten. Damit ist es möglich, Attributen direkt einen Wert zu übergeben. Notwendig ist das, wenn man innerhalb der

Klasse ein konstantes Attribut hat. Weiterhin besteht die Möglichkeit, vor der Ausführung des Konstruktors einen anderen Konstruktor innerhalb der Klasse aufzurufen. Verdeutlicht wird dies in Listing 20.11 Zeile 13.

Listing 20.11: Konstruktoren

```
1  #include <iostream> // cout
2  #include <string>   // string
3
4  using namespace std;
5
6  class Person // inline Deklaration einer Klasse
7  {
8  public:
9    // Standardkonstruktor, ruft den spezielleren Konstruktor
10   // Person(string) auf, um einen Default-Namen zu vergeben.
11   Person() : Person("kein Name") {}
12   // Dieser Konstruktor arbeitet mit einer Initialisierungs-
13   // liste. Er weist dem Attribut name den Wert von neuerName
14   // zu.
15   Person(string neuerName) : name(neuerName) {}
16
17   void printName() { cout << name << endl; }
18
19 private:
20   // Dieses Attribut ist konstant, es MUSS also im Konstruktor
21   // initialisiert werden.
22   const string name;
23 };
24
25 int main()
26 {
27   Person person; // Aufruf des Standardkonstruktors
28
29   person.printName(); // Ausgabe: "kein Name"
30
31   return 0;
32 }
```

Analog zu Konstruktoren, welche bei der Erzeugung von Objekten aufgerufen werden, gibt es in C++ auch Destruktoren, welche beim Zerstören von Objekten

aufgerufen werden. Im Destruktor sollte der Speicher wieder aufgeräumt werden. Sollte unser Objekt zum Beispiel einen Zeiger auf eine große Liste im Speicher haben, sollte der Speicher für die Liste ebenfalls freigegeben werden.

20.4.2 Das Schlüsselwort `const`

`const` entspricht in etwa dem `final` in Java. Man kann damit Konstanten anlegen, aber der wichtigste Grund ist die Parameterübergabe als Referenz, deren Ziel nicht verändert werden darf. Dadurch ist es möglich, große Objekte als Zeiger zu übergeben (anstelle einer Kopie auf den Stack) und trotzdem die Gewissheit zu haben, dass das Objekt innerhalb der Funktion nicht verändert werden wird.

Damit die Methode Person::getName() in Listing 20.12 überhaupt ausgeführt werden kann, muss sie auch in der Klassendeklaration (Listing 20.13) mit dem Schlüsselwort const dekoriert werden. Innerhalb einer const-Methode kann nicht schreibend auf Attribute zugegriffen werden, und es dürfen nur const-Methoden des aktuellen Objekts (`this`) aufgerufen werden.

Listing 20.12: Übergabe eines const-Zeigers

```
void printPerson(const Person* personZeiger)
{
  std::cout << personZeiger->getName() << "\n";
}
```

Listing 20.13: Deklaration der Klasse `Person` für Listing

```
class Person
{
public:
  Person(string name);
  virtual ~Person();

  string getName() const;
private:
  string name;
};
```

20.4.3 Vererbung

Die Vererbung in C++ ist analog zu der Java Vererbung. Unterschiede gibt es unter anderem bei Zugriffsrechten von Elementen der Basisklasse. Mit den bereits genannten Sichtbarkeitsmodifizierern kann festgelegt werden, wie auf Elemente der Basisklasse zugegriffen werden darf. Tabelle 20.1 zeigt alle Möglichkeiten. Die Vererbung mit `public` entspricht der Vererbung in Java. Sollen Methoden in einer abgeleiteten Klasse überschrieben werden können, muss die Methode in der Basisklasse als `virtual` dekoriert werden. Dies ist vor allem dann wichtig, wenn man eine Referenz auf das Basisobjekt hat, aber die Methode der abgeleiteten Klasse nutzen möchte. Ist eine Methode mit `virtual` dekoriert, wird immer die Funktion der konkreten Klasse verwendet. Das Beispiel in Listing 20.14 soll die Verwendung von `virtual` deutlich machen.

	Modifizierer in Basisklasse		
Vererbung	**private**	**protected**	**public**
private	nicht zugreifbar	Member ist private	Member ist private
protected	nicht zugreifbar	Member ist protected	Member ist protected
public	nicht zugreifbar	Member ist protected	Member ist public

Tabelle 20.1: Vererbung mit Sichtbarkeitsmodifizierern

Listing 20.14: Virtuelle Methoden

```
1  #include <iostream>
2  class Basis
3  {
4  public:
5    virtual ~Basis() {}
6    virtual void virtuelleMethode() { std::cout << "Basis\n"; }
7    void nichtVirtuelleMethode() { std::cout << "Basis\n"; }
8  };
9  class Abgeleitet : public Basis
10 {
11 public:
12   virtual ~Abgeleitet() {}
13   virtual void virtuelleMethode() {std::cout << "Abgeleitet\n";}
14   void nichtVirtuelleMethode() { std::cout << "Abgeleitet\n"; }
15 };
16 int main()
17 {
18   Basis* basisZeiger = new Abgeleitet();
19   basisZeiger->virtuelleMethode();        // Ausgabe: Abgeleitet
```

```
20    basisZeiger->nichtVirtuelleMethode(); // Ausgabe: Basis
21    delete basisZeiger;
22    return 0;
23 }
```

Es gibt in C++ nicht nur virtuelle Methoden: auch der Destruktor kann (*und sollte bei Vererbung grundsätzlich!*) virtuell dekoriert werden. Dadurch wird gewährleistet, dass bei Vernichtung des abgeleiteten Objekts immer der abgeleitete Destruktor aufgerufen wird, auch wenn das Objekt als Basis-Typ verwendet wurde.

C++ lässt Mehrfachvererbung zu, wodurch es unter Umständen zu Namenskonflikten kommen kann. Sollte eine Bezeichnung (für ein Attribut beispielsweise) mehrfach geerbt werden, kann man den Namen der Basisklasse, durch zwei Doppelpunkte getrennt, voranstellen, um die Herkunft deutlich zu machen. (Qualifizierungs-Operator) Mehrfachvererbung kann aber auch in Form des Mixin-Patterns verwendet werden. Hierbei gibt es Klassen, welche bestimmte Basis-Funktionalitäten beinhalten, welche in andere Klassen übernommen werden sollen, ohne deren Vererbungshierarchie zu ändern. Ein Beispiel für Mixins ist in Abbildung 20.1 und Listing 20.15 zu sehen.

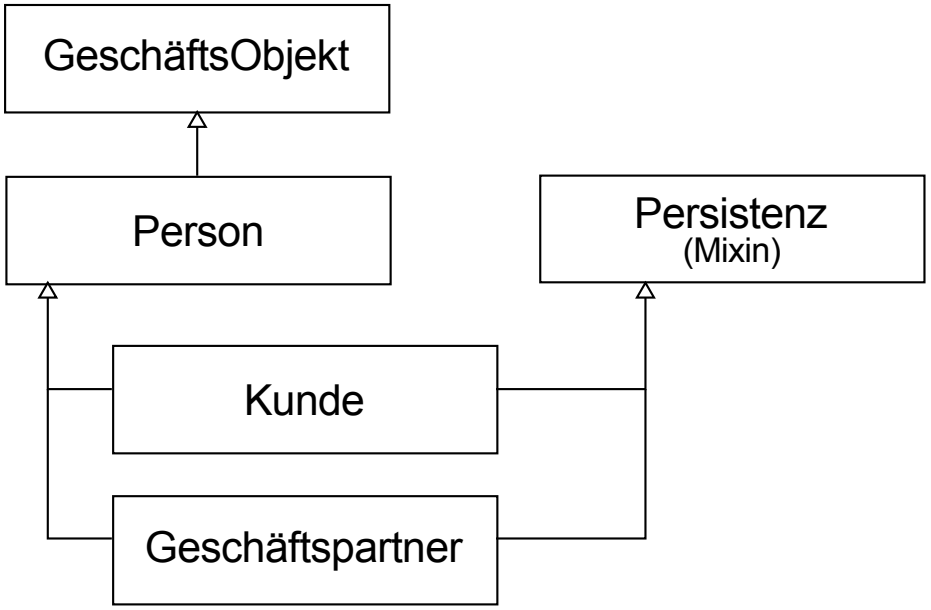


Abbildung 20.1: Mixin-Beispiel UML

Listing 20.15: Mixin-Deklaration

```
1 class Persistenz
2 {
3 public:
4   void SaveToDB() const;
```

```
5    void ReloadFromDB();
6  };
7  class GeschaeftsObjekt { /* ... */ };
8  class Person : public GeschaeftsObjekt { /* ... */ };
9  class Kunde : public Person, public Persistenz { /* ... */ };
10 class Geschaeftspartner : public Person, public Persistenz
11                                                { /* ... */ };
```

20.4.4 Abstrakte Klassen

Aufgrund von Mehrfachvererbung ist es in C++ weder notwendig noch möglich, Interfaces zu deklarieren. Es gibt nur die Möglichkeit abstrakter Klassen. Hierfür gibt es jedoch keinen Modifizierer vor der Klasse wie in Java, stattdessen muss mindestens eine Operation innerhalb der Klasse als „pure virtual“ markiert werden. Dies erreicht man, indem man einer virtuellen Operation den Wert 0 zuweist. Siehe Listing 20.16 Zeile 7.

Listing 20.16: Abstrakte Klasse

```
1  class AbstractBase // abstrakt aufgrund von f()
2  {
3  public:
4    AbstractBase() {}
5    virtual ~AbstractBase() {}
6
7    virtual void f() = 0; // pure virtual!
8  };
9
10 class Derived : public AbstractBase
11 {
12 public:
13   Derived() {}
14   virtual ~Derived() {}
15
16   virtual void f() {} // implementiert f()
17 };
```

20.5 Überladung von Operatoren

Mit Hilfe der Operatorüberladung können selbst definierte Klassen Operatoren von C++ mit einer neuen Bedeutung versehen. Die Idee ist dabei die selbe wie beim Überladen von Funktionen und Methoden. Hinter jedem Operator steckt eine entsprechende Funktionsnotation. Bei der Operatorüberladung werden diese unter der Prämisse, dass zumindest ein Argument den Typ einer (selbst definierten) Klasse oder eine Referenz darauf besitzt, überladen.

Operatorüberladung ist dann sinnvoll, wenn die Benutzung des Operators danach intuitiv ist. Die wohl bekannteste Operatorüberladung, welche in diesem Kapitel auch schon oft benutzt wurde, ist die Überladung des «-Operators. Dieser ist eigentlich für bitweises Verschieben zuständig. Im Zusammenhang mit cout schreibt er jedoch Objekte auf die Konsole. Er wird üblicherweise bei allen Stream-Klassen verwendet. Ein ausführliches Beispiel findet sich in den folgenden Listings.

Listing 20.17: Klassendeklaration für Brüche - Datei `bruch.h`

```
1  #ifndef BRUCH_H
2  #define BRUCH_H
3
4  class Bruch
5  {
6    // Damit die globale Funktion auf private Member (zaehler
7    // und nenner) zugreifen kann, wird sie hiermit zum Freund
8    // der Klasse ernannt.
9    friend Bruch operator+(const Bruch& lhs, const Bruch& rhs);
10 public:
11   Bruch(int zaehler, int nenner);
12   void ausgeben() const;
13 private:
14   int zaehler;
15   int nenner;
16 };
17
18 // Addition ueber globale Funktion
19 Bruch operator+(const Bruch& lhs, const Bruch& rhs);
20
21 #endif // BRUCH_H
```

Listing 20.18: Implementierung für Brüche - Datei `bruch.cpp`

```
#include "bruch.h"
#include <iostream>

Bruch::Bruch(int zaehler, int nenner)
  : zaehler(zaehler), nenner(nenner)
{}

void Bruch::ausgeben() const
{
  std::cout << zaehler << "/" << nenner << "\n";
}

Bruch operator+(const Bruch& lhs, const Bruch& rhs)
{
  int neuerNenner = lhs.nenner * rhs.nenner;
  int neuerZaehler = (lhs.zaehler * rhs.nenner) +
                     (rhs.zaehler * lhs.nenner);
  return Bruch(neuerZaehler, neuerNenner);
}
```

Listing 20.19: Verwendung der Klasse Bruch - Datei `main.cpp`

```
#include <bruch.h>
int main()
{
  Bruch a(1, 6);
  Bruch b(1, 3);
  Bruch c = a + b;
  c.ausgeben(); // Ausgabe: 9/18
  return 0;
}
```

20.6 Code-Konventionen

Eine Klasse pro Datei.

Klasse UpperCamelCase (`HelloWorld`)

Methoden lowerCamelCase (`helloWorld`)

Konstante Großgeschrieben, Worttrennung mit Unterstrich (`HELLO_WORLD`)

Attribute lowerCamelCase (`helloWorld`)

lokale Variable lowerCamelCase (`helloWorld`)

21 Refaktorierung und Test-Framework

21.1 Quelltexte strukturieren

Um die weitere Entwicklung des Betriebssystems zu vereinfachen soll der Quelltext strukturiert abgelegt werden. Hierzu gehört ein Überdenken der Verzeichnisstruktur und der umsichtige und konsistente Einsatz von Namensräumen. Dieser Abschnitt wird die vorgefundene Situation des Quelltextes und die notwendigen strukturellen Änderungen beschreiben.

21.1.1 Aktuelle Situation

Die vorgefundene Dateistruktur der Quelltexte des Betriebssystemkerns kann als „historisch gewachsen“ bezeichnet werden. Jedes Teilprojekt des Kerns wurde in einem eigenen Verzeichnis unterhalb des Hauptverzeichnisses implementiert. Einige Klassen wurden auch direkt im Hauptverzeichnis implementiert. In Abbildung 21.1 wird die bisherige Struktur der Verzeichnisse gezeigt. Angegeben ist jeweils die Anzahl an Quelltext- und Testfalldateien.

Wie eingangs erwähnt sind viele ähnliche oder zusammenhängende Elemente als einzelne Verzeichnisse angelegt worden. Beispielsweise gehören der Allokator („alloc“) und das Paging in die Kategorie Speicherverwaltung. Warteschlangen (Queue), Indexstrukturen (Map) und Listen haben alle die gleiche Aufgabe: mehrere Objekte strukturiert zu speichern. Zusätzlich fällt auf, dass es im Projekt derzeit sieben Verzeichnisse mit dem Namen „test(s)“ gibt. Obwohl es bereits ein Verzeichnis „test“ im Hauptverzeichnis gibt, hat jedes Teilprojekt die Testfälle im eigenen Verzeichnis angelegt. Die Testfälle sind also über die gesamte Verzeichnisstruktur verteilt.

Namensräume wurden bisher nur vereinzelt eingesetzt. Gerade bei einem Projekt mit vielen Beteiligten und mehreren parallel arbeitenden Teams ist der Einsatz

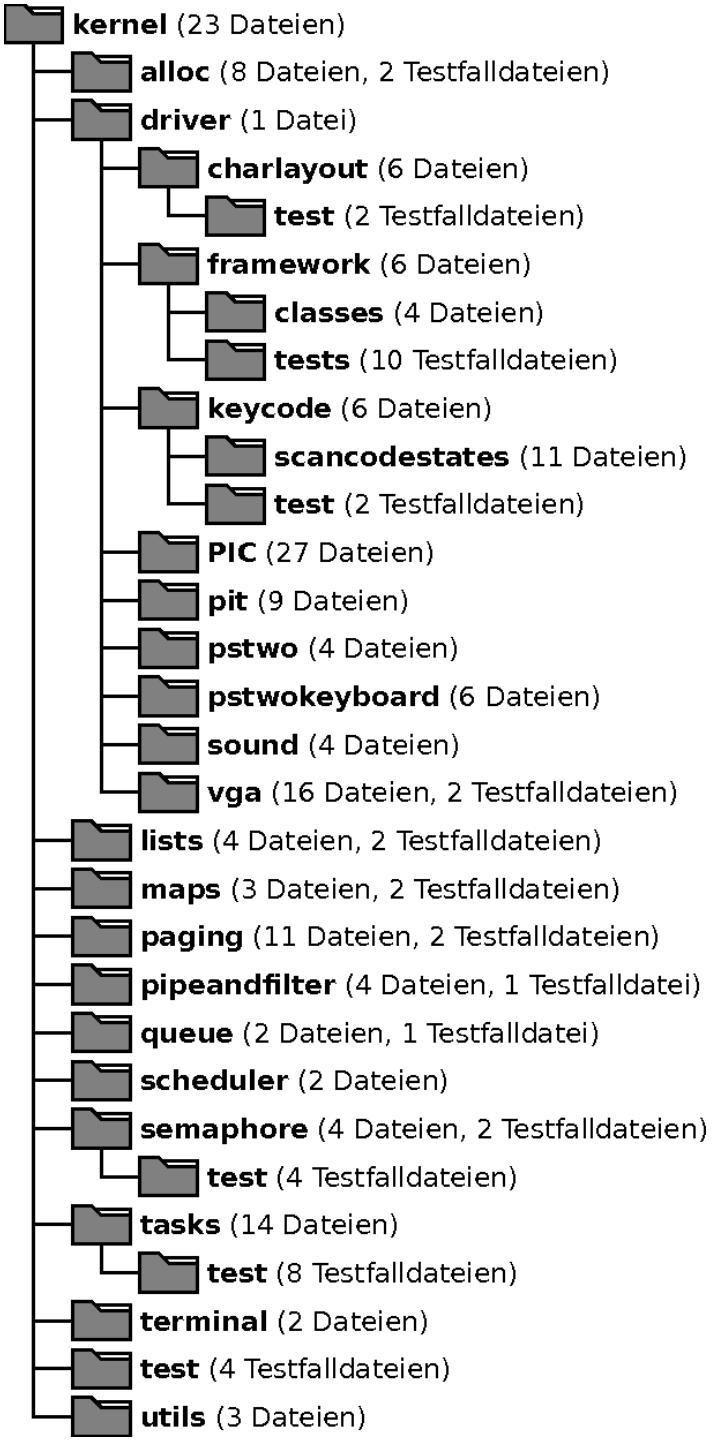


Abbildung 21.1: Verzeichnisstruktur vor dem Umbau (Revision 868)

von Namensräumen zur eindeutigen Identifikation von Klassen über das gesamte Projekt hinweg angebracht.

21.1.2 Neue Verzeichnisstruktur

Zusammen mit allen Beteiligten wurde sich für folgende grobe Gruppierungen entschieden:

cpu Dieses Verzeichnis enthält prozessorspezifische Klassen.

io In diesem Verzeichnis befindet sich alles, was mit Ein- und Ausgabe zu tun hat. Hierzu gehören auch alle Treiber/Geräte.

memory Verzeichnis für Speicherverwaltung und Datenstrukturen.

task Enthält notwendige Klassen für das Multitasking, inklusive Synchronisation.

test Hier soll ein Testframework entstehen. Die Testfälle werden in Unterverzeichnissen strukturiert abgelegt.

tool Hilfsklassen und Werkzeuge.

Diese Gruppierung wurde umgesetzt. Alle Klassen wurden in die entsprechenden Verzeichnisse kopiert. Die Verweise („#include“-Zeilen) mussten in allen Dateien angepasst werden. In diesem Zusammenhang wurden, um die Übersichtlichkeit zu erhöhen, Verweise über viele Oberverzeichnisse (z. B. „../../../tool/utils.h“) zu Verweisen, welche direkt vom Wurzelverzeichnis aus aufgerufen werden (z. B. „tool/utils.h“). Hierfür war die Anpassung des Makefiles notwendig. Der Compiler sucht bei Verweisen nun grundsätzlich vom Wurzelverzeichnis („kernel“) aus.

In Abbildung 21.2 sind die neuen Verzeichnisse dokumentiert. Alle Testfälle wurden unterhalb des Verzeichnisses „test“ abgelegt. Da sich die Struktur der Testfälle und die Verzeichnisstruktur, in der diese liegen, durch das neu entstehende Testframework noch ändern wird, wird an dieser Stelle nicht weiter darauf eingegangen.

Zusätzlich zu den neu entstanden Verzeichnissen wurden Namensräume projektübergreifend eingeführt. Jede Klasse erhält den Namensraum des Verzeichnisses, in dem sie sich befindet. Ein Beispiel: Die Klasse „SpeakerDriver“ der Datei „io/driver/sound/SpeakerDriver.h“ ist nun im Namensraum „io::driver::sound“. Für das weitere Entwickeln ist wichtig, dass innerhalb dieser Klasse jetzt ohne weiteres auf alle Klassen des gleichen, und aller übergeordneten Namensräume

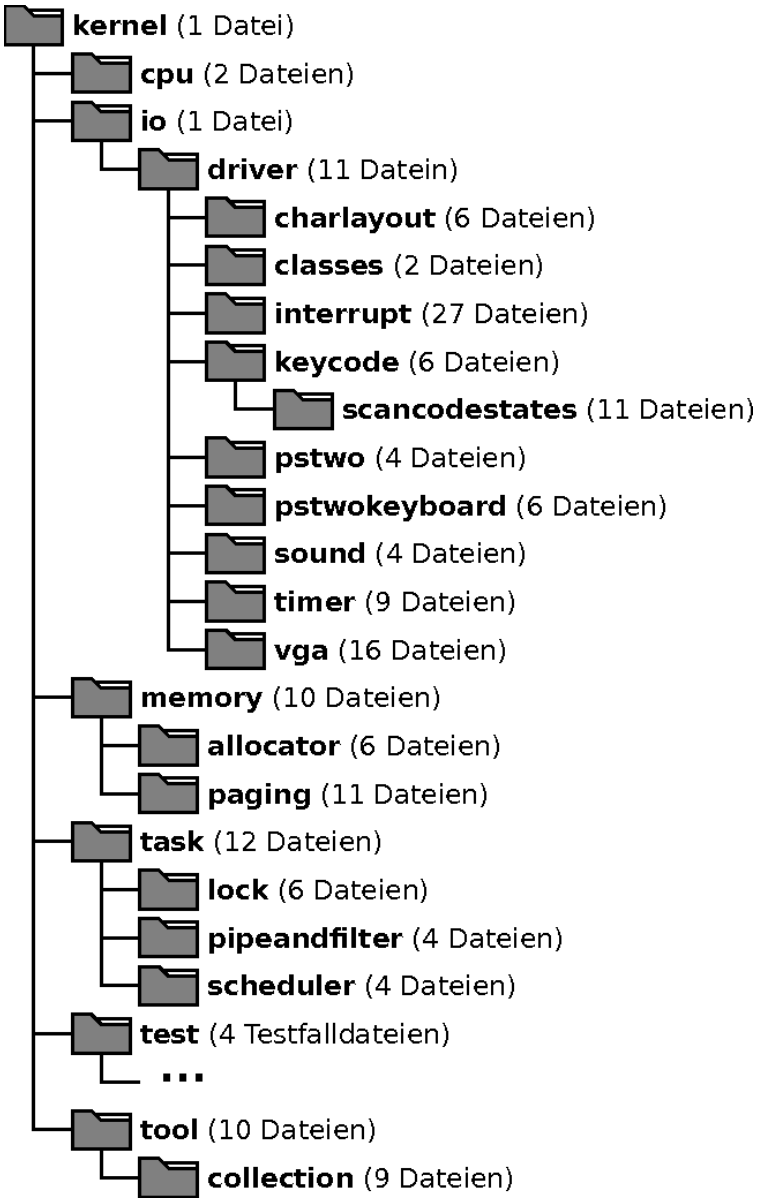


Abbildung 21.2: Verzeichnisstruktur nach der Refaktorierung (Revision 873)

(„io“ und „io::driver“) zugegriffen werden kann.[60]

Möchte man mit Klassen eines anderen Namensraums (z. B. „task::lock“) arbeiten, kann dieser über das C++-Konstrukt „using namespace task::lock;“ geschehen. Dieses Konstrukt sollte möglichst nach dem Betreten des eigentlichen Namensraumes am Anfang der Datei platziert werden. Möglich ist auch die Platzierung innerhalb einer Methode. Allerdings ist strikt darauf zu achten, dass „using namespace“-Zeilen niemals in Header-Dateien vorkommen. Da diese Dateien in andere Dateien eingebunden werden und in diesem Fall die Einbindung des Namensraums auch für die ursprüngliche Datei gilt. Dies ist in der Regel so nicht gewollt.

21.2 Testframework

Das folgende Kapitel beschäftigt sich mit dem Thema „Testframework für das FHDWOS“. Dabei sollen im ersten Schritt die existierenden Testfälle des Betriebssystems, die bereits vor Projektbeginn implementiert wurden, analysiert werden. Im zweiten Schritt wird ein Entwurf für das Testframework erstellt. Dieses Framework soll den Entwicklern des FHDWOS das Erstellen von Testfällen vereinfachen und diese in eine einheitliche Form bringen. Anschließend dient der Entwurf als Basis für die Implementierung. Im letzten Schritt wird eine Anleitung erstellt, welche hilfreiche Informationen für die Verwendung des Testframework beinhaltet.

21.2.1 Analyse

In der Analysephase werden die bereits bestehenden Testfälle und die dazugehörigen Hilfsfunktionen im Projekt des FHDWOS analysiert. Hierbei hat sich herausgestellt, dass die Klasse „Testmethods“ für den Aufruf und die Reihenfolge der bereits bestehenden Testfälle verantwortlich ist. Diese Klasse bietet mit der Methode „runTests“ die Möglichkeit alle Testfälle nacheinander auszuführen, einzelne Testfälle können bisher nur eingeschränkt ausgeführt werden.

Die bereits existierenden Hilfsmethoden befinden sich in der Header-Datei „Assert.h“. Die folgenden Methoden stehen dort zur Verfügung:

- Assert (Überladen für Zahlen, Pointer und Keycodes)
- AssertTrue

[60] Vorausgesetzt die nötigen Include-Zeilen sind vorhanden.

- AssertFalse
- printFailure (Überladen für Zahlen und Wahrheitswerte)

Für die Verwendung eines gemeinsamen Testframeworks aller bereits existierenden Testfälle ist es zusätzlich notwendig, diese auf das neue Framework umzustellen. Welche Neuerungen hierfür notwendig sind, wird im weiteren Verlauf beschrieben.

Des Weiteren sind folgende Probleme bei der Analyse der Testfälle aufgefallen:

- Verschiedene Namensgebungen
- Konsolenausgaben bei Start/Ende
- Verschiedene Arten der Implementierung (Klassen/Methoden)
- Reihenfolge des Testfälle

21.2.2 Entwurf

Die Entwurfsphase stellt die Basis der Implementierung dar und definiert alle zu implementierenden Klassen inklusive ihrer Assoziationen. Das Klassendiagramm in Abbildung 21.3 stellt die Basisklassen des Testframework inklusive ihrer Methoden dar.

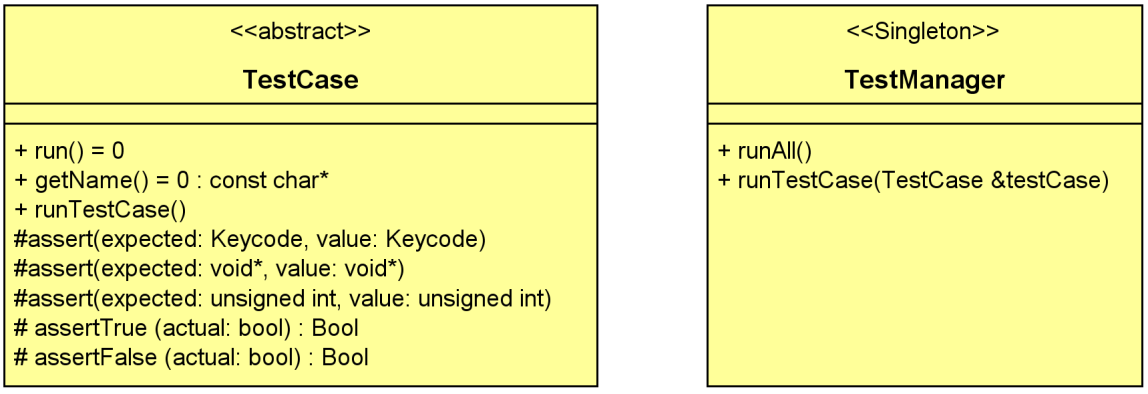


Abbildung 21.3: Basisklassen des Testframework

Die Klasse „TestManager“ ist für die Verwaltung der gesamten Testfälle zuständig. Mithilfe dieser Klasse können sowohl einzelne als auch alle Testfälle ausgeführt werden. Das Ausführen aller Testfälle wird über die Methode „runAll()“ realisiert. In dieser wird die genaue Reihenfolge der Testfälle definiert. Die Ausführung einzelner Tests ermöglicht die Funktion „runTestCase“, dieser kann ein einzelner Testfall übergeben und anschließend ausgeführt werden.

Für die Erstellung eines Testfalls steht die abstrakte Klasse „TestCase“ bereit, diese stellt für jeden Testfall die bereits in der Analysephase beschriebenen Hilfsmethoden (Assert etc.) bereit. Für die Ausführung des jeweiligen Testfalls ist „runTestCase“-Methode verantwortlich. In dieser werden die Hilfsfunktionen, die für jeden einzelnen Testfall hilfreich sind verwendet.

Für die Ausgabe von Informationen über den aktuellen Verlauf der Tests sind die Testfälle zuständig. Sie teilen sowohl den Start als auch das Ende eines Testfalls automatisch mit. Hierfür ist die Verwendung des sogenannten „Template-Pattern“ vorgesehen. Möchte ein Entwickler weitere Ausgaben während seines Testfalls ausführen, so muss er dies über die „puts“-Methode machen.

21.2.3 Verwendung des Testframework

Dieser Punkt erläutert, wie das TestFramework verwendet werden kann und erleichtert dem Entwickler die Verwendung. Es werden folgend alle wesentlichen Schritte für die Erstellung eines neuen Testfalls beschrieben.

Im ersten Schritt muss eine neue Klasse für den Testfall geschaffen werden, hierbei ist es wichtig, dass die Namenskonventionen (Beispiel: „AllocatorTestCase“) eingehalten werden und diese von der Klasse „TestCase“ erbt. Anschließend steht die „run“-Methode der Klasse für die Implementierung des Testfalls bereit. Die automatische Konsolenausgabe für den Start und das Ende des Testfalls wird durch die Methode „getName“ realisiert. Des Weiteren sollte darauf geachtet werden, dass sowohl Destruktor als auch Konstruktor der Klasse implementiert werden.

Der Aufruf des Testfalls wird über die Klasse „TestManager“ realisiert, diese ruft die „runTestCase“-Methode des zuvor erstellten Testfalls auf und führt diesen somit aus. Der Aufruf sollte passend in die Reihenfolge der bereits implementierten Aufrufe eingeordnet und anschließend getestet werden.

21.2.4 Fazit

Im Zuge dieser Teilaufgabe wurden im ersten Schritt die bereits erstellten Testfälle für einzelne Bereiche des FHDWOS analysiert, bevor im zweiten Schritt ein Entwurf für ein Testframework geschaffen wurde, welches für das Erstellen von Testfällen und für eine einheitliche Struktur dieser verantwortlich ist. Anschließend wurde dieser Entwurf in der Realisierungsphase umgesetzt.

Für die Nutzung des Testframeworks wurde eine Anleitung zur Verfügung gestellt, welche bei der Verwendung dieses eine Hilfestellung bietet.

21.3 Strukturieren des Boot-Vorgangs

Der zentrale Einstiegspunkt des Systems ist die Datei `main.cc`. Im Entwicklungsprozess werden an dieser Stelle eine zunehmende Anzahl an Methodenaufrufen eingefügt, die unterschiedlichen Zielen dienen. Dazu zählen auch der Aufruf von Tests, sekundären (nicht systemrelevanten) Prozessen und der probeweise Aufruf eigener Neuentwicklungen. Mit der fortschreitenden Entwicklung des Betriebssystems wird es notwendig, den Boot-Vorgang zu strukturieren und in einen eigenen Bereich auszulagern.

Zu diesem Zweck wird die Klasse `BootManager` bereitgestellt. Nach Erstellen einer Instanz und Aufruf der Methode `boot` wird das Betriebssystem initialisiert. Listing 21.1 zeigt den aktuellen Ablauf.

Listing 21.1: Boot-Vorgang

```
1 this->initPaging(mmEntry, kernelSize);
2 this->initConsole();
3 this->initAllocator();
4 Scheduler::getInstance().initScheduler();
5 this->initDrivers();
6 Scheduler::getInstance().initSchedulerAfterDriver();
7 this->initProcessCommunication();
8 this->finalize();
```

21.4 Initialisierung der Konsole

Bisher erfolgte das Schreiben auf die Konsole meist über statische Methoden in der Datei `console.h`. Mit der Entwicklung des VGA-Treibers für die Konsole im Textmodus ist die bisherige Implementierung veraltet. Insbesondere ist die gleichzeitige Verwendung beider Implementierungen nicht möglich. Der VGA-Treiber soll so früh wie möglich für die Ausgabe von Statusmeldungen auch während des Boot-Vorgangs verwendet werden. Dies ist jedoch erst möglich, sobald das Treiber-Framework und die Treiberimplementierungen initialisiert sind.

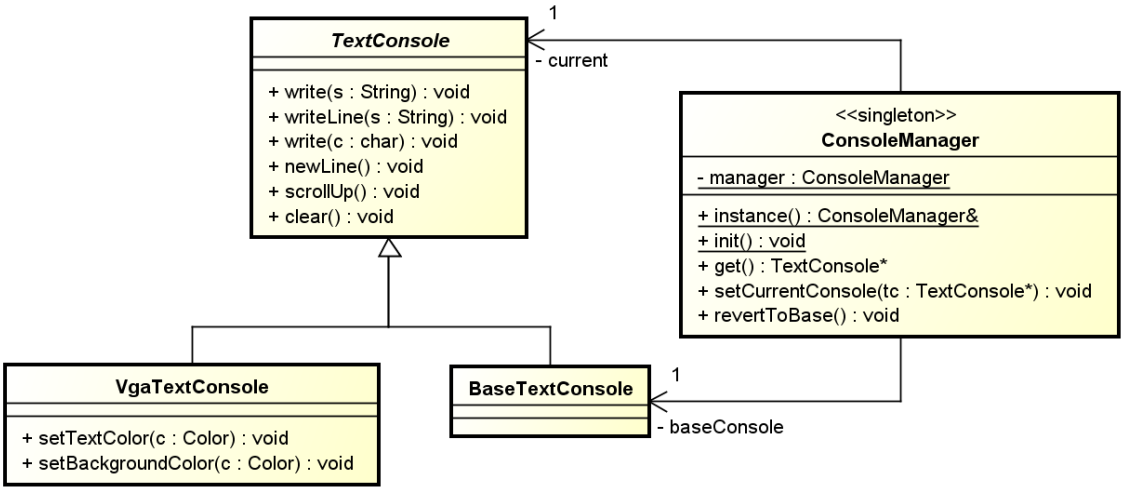


Abbildung 21.4: Klassenmodell zum ConsoleManager

Um bereits vor diesem Zeitpunkt die Konsole verwenden zu können, wird die treiberunabhängige Implementierung aus `console.h` in eine eigene Klasse, `BaseTextConsole`, ausgelagert. Um die Austauschbarkeit der zwei Konsolen zu gewährleisten, wird eine abstrakte Oberklasse `TextConsole` definiert. Ein Objekt dieser Klasse dient als austauschbares Strategieobjekt im `ConsoleManager` – dargestellt ist dies in Abbildung 21.4. Der `ConsoleManager` ist als Singleton implementiert und besitzt eine `BaseTextConsole`, die bei Systemstart und auch beim Herunterfahren des Systems verwendet werden kann.

Das Schreiben auf die Konsole erfolgt weiterhin durch die Methode `puts(char* s)`, die nun in `utils.h` deklariert ist. In einem Kontext, in dem der Treiber bereits initialisiert wurde, kann die VGA-Konsole direkt verwendet werden. Somit ist ein Zugriff auf die darin enthaltene, erweiterte Funktionalität weiterhin möglich.

22 Erstellung des Handbuchs

22.1 Prozess der Handbuch-Erstellung

Zur Erstellung des Handbuchs und seiner einzelnen Kapitel wurde das Textsatzsystem LaTeX[61] verwendet. Im Folgenden wird unterschieden zwischen dem Hauptdokument (das vorliegende Handbuch) und den Teildokumenten (die einzelnen Kapitel des Handbuchs).

22.1.1 Struktur der Teildokumente

Die Verwendung von LaTeX ermöglicht eine stark modulare Struktur der Quelldateien dieses Dokuments. Konkret bedeutet dies, dass die einzelnen Kapitel jeweils eigenständige Dokumente darstellen, die schließlich zu dem vorliegenden Handbuch zusammengefügt werden. Um einen reibungslosen Ablauf des Zusammenfügens zu erlauben, wurde bereits im Vorfeld der ersten Dokumentationsarbeiten eine einheitliche Infrastruktur der Teildokumente vereinbart. Diese Struktur sieht die Aufteilung in jeweils drei Quelldateien vor (vgl. linker Teil in der Abbildung 22.1):

document.tex ist die Hauptdatei des Dokuments. Hier werden Einstellungen vorgenommen, die ausschließlich für den Druck des Teildokuments verwendet werden. Hier werden die Vorlage und der Textteil eingebunden.

fhdw-os.tex enthält alle Paketdefinitionen für LaTeX sowie globale Einstellungen. Dieselbe Vorlage bildet die Grundlage für das Handbuch.

inhalt.tex enthält den Textteil des Dokuments.

Auf dieser Grundlage sind zum Erstellen des Handbuchs folgende Schritte notwendig, die im Weiteren näher erläutert werden:

1. Anlegen des Hauptdokuments
2. Ablage der Teildokumente in Unterordnern zum Hauptdokument

[61] siehe [5] und [6]

3. Inkludieren der `inhalt.tex` der Teildokumente
4. Lösen von Konflikten (gleiche `label`)

22.1.2 Aufbau des Hauptdokuments

Die Struktur des Hauptdokuments ist an die Struktur der Teildokumente angelehnt und in Abbildung 22.1 dargestellt. Als Vorlage für das Hauptdokument dient ebenfalls die Datei `fhdw-os.tex`, jedoch mit geringfügigen Anpassungen. Neben der Änderung der Dokumentenklasse auf *Report* sind das noch folgende:

Listing 22.1: Ergänzung der LaTeX-Vorlage

```
% Fallunterscheidungen
\usepackage{etoolbox}
% Package, um beim include von Teilen relative Pfade beizubehalten
\usepackage{import}
%Fußnoten durchgehend nummerieren
\usepackage{chngcntr}\counterwithout{footnote}{chapter}
% KOMA-package für Kopf-und Fußzeilen
\usepackage{scrpage2}
\pagestyle{headings}
\setheadsepline{.4pt}
```

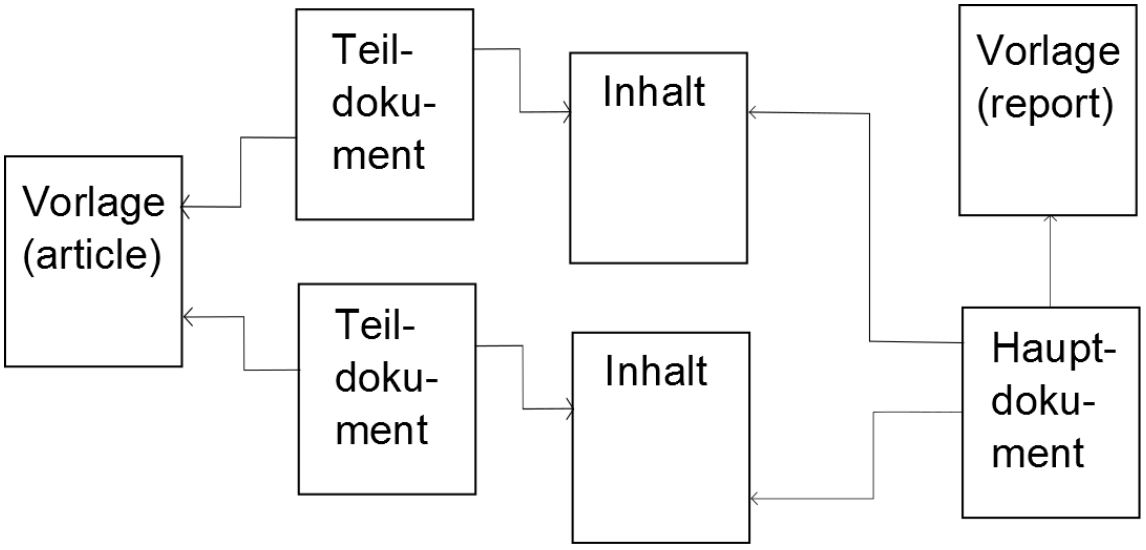


Abbildung 22.1: Interne Struktur des Handbuchs

Nach dem Anlegen der Teildokumente in den entsprechenden Unterordnern werden sie mithilfe von `subimport` in das Dokument inkludiert. So werden die relativen Pfadangaben, z. B. zu Grafiken, weiterhin korrekt aufgelöst. Jedes Teildokument erhält schließlich ein eigenes Kapitel (`chapter`). Nun wird das Dokument

kompiliert um Konflikte zwischen den Teildokumenten zu erkennen und anschließend zu beseitigen. Neben individuellen Definitionen eigener Befehle betrifft dies insbesondere global eindeutige Bezeichner (`label`), die für Referenzen verwendet werden. Eine solche mehrfache Definition ist in der Log-Datei zu finden:

```
LaTeX Warning: Label ‘lst:referenz’ multiply defined.
```

Zum Lösen des Konflikts lässt sich der Bezeichner suchen und entsprechend ersetzen. Dabei wird ein Präfix verwendet, das für jedes Kapitel eindeutig ist. Dieses Vorgehen erfordert jedoch mehrere manuelle Arbeitsschritte und ist daher folgendermaßen automatisiert:

Als Präfix (`chapterPrefix`) dient der Dateipfad vom Hauptdokument zum Teildokument. Da dieser wiederholt gesetzt werden muss, wird anstatt `subimport` ein weiterer Befehl `importChapter` verwendet. Dieser kombiniert das Setzen des Präfixes mit dem Importieren des Teildokuments (vgl. Listing 22.2). Um in den Teildokumenten wie gewohnt die Befehle `label`, `ref` und `pageref` zu verwenden, werden diese ebenfalls neu definiert (siehe Listing 22.3).

Listing 22.2: Automatisches Präfix für Kapitel

```
% Präfix für jedes Kapitel
\newcommand{\chapterPrefix}{title}
% Präfix = Pfad zum Teildokument
\newcommand{\importChapter}[1]{%
\renewcommand{\chapterPrefix}{#1}%
\subimport{#1}{inhalt}%
}
```

Listing 22.3: Verwendung des Präfixes an Bezeichnern

```
% Präfix vor label-Definitionen
\newcommand*{\unprefixedLabel}[1]{\label{#1}}
\let\unprefixedLabel\label
\renewcommand{\label}[1]
  {\unprefixedLabel{\chapterPrefix#1}}
% Präfix vor ref-Definitionen
\newcommand*{\unprefixedRef}[1]{\ref{#1}}
\let\unprefixedRef\ref
\renewcommand{\ref}[1]
  {\unprefixedRef{\chapterPrefix#1}}
% Präfix vor pageref-Definitionen
\newcommand*{\unprefixedPageref}[1]{\pageref{#1}}
```

```
\let\unprefixedPageref\pageref
\renewcommand{\pageref}[1]
  {\unprefixedPageref{\chapterPrefix#1}}
```

22.1.3 Anlegen des Index

Der Index am Ende des Handbuchs wurde mit dem LaTeX-Werkzeug **makeindex** und dem Paket `makeind` (für mehrere Index-Verzeichnisse) generiert. Die Auswahl und Anlage der Indices erfolgte beim Lesen des Handbuchs durch Hinzufügen von **\index{}-Einträgen**. Die verwendete Style-Datei für Makeindex ist in Listing 22.4 aufgeführt[62].

Listing 22.4: Makeindex Style

```
quote '+'
headings_flag 1
heading_prefix "{\\bf "
heading_suffix "}\\nopagebreak%\n \\indexspace\\nopagebreak%"
delim_0 "\\dotfill "
delim_1 "\\dotfill "
delim_2 "\\dotfill "
delim_r "~---~"
suffix_2p "\\,f."
suffix_3p "\\,ff."
```

22.2 Doxygen

Alle Klassen des Betriebssystems sind mithilfe von Kommentaren im Quellcode dokumentiert. Um aus diesen ein lesbares Dokument zu erzeugen, wird Doxygen (siehe [7]) verwendet. Damit ist es auch möglich, LaTeX-Code zu erzeugen. Die dazu notwendigen Einstellungen aus der Datei `Doxyfile` sind in Listing 22.5 aufgeführt. Zum Erstellen der Quellcode-Dokumentation wird im Ordner der Doxyfile das Programm `doxygen` ausgeführt. Die Integration in das Handbuch erfolgt durch den Import der generierten Datei `refman.tex` im Hauptdokument. Da Doxygen für die Darstellung eigene Varianten von `list`, `enumerate` und `itemize`

[62] Entnommen aus [8], dort sind auch weitere Informationen zur Anwendung von Makeindex zu finden.

verwendet, wird außerdem eine gestutzte Version der `doxygen.sty` in das Dokument eingebunden. Abschließend sind in der Datei `refman.tex` die zwei letzten Zeilen (`\printindex` sowie `\end{document}`) zu löschen.

Listing 22.5: Auszug aus der Doxyfile

```
GENERATE_LATEX          = YES
LATEX_OUTPUT            = latex
LATEX_CMD_NAME          = latex
MAKEINDEX_CMD_NAME      = makeindex
COMPACT_LATEX           = YES
PAPER_TYPE              = a4
EXTRA_PACKAGES          =
LATEX_HEADER            = doxygen_latex-header.tex
LATEX_FOOTER            = doxygen_latex-header.tex
PDF_HYPERLINKS          = YES
USE_PDFLATEX            = YES
LATEX_BATCHMODE         = NO
LATEX_HIDE_INDICES      = NO
LATEX_SOURCE_CODE       = NO
LATEX_BIB_STYLE         = plain
```

22.3 Überarbeitung des Handbuchs

Die Aufgabe war es, die Dokumentation ein weiteres Mal auf den aktuellen Stand (Phase 7) zu bringen. Dabei wurden die Ergebnisse aus Phase 4-7 korrekturgelesen und der Index wurde erweitert. Ferner wurden einige Konfigurationsdateien angepasst um das Programm Doxygen mit der aktuellsten Version (1.8.5) verwenden zu können. Infolgedessen wurde ein Teil der Code-Dokumentation – um z.B. fehlende Parameter – ergänzt. Des Weiteren mussten aufgrund der Größe der vorhandenen Dokumentation die Abstandsangaben der Nummerierungen zu den Benennungen des Inhalts- und Quelltextverzeichnisses angepasset werden.

Literaturverzeichnis

[1] Intel® 64 and IA-32 Architectures, Software Developer's Manual. Volume 3A: System Programming Guide, Part 1

[2] E.W. Dijkstra, Cooperating Sequential Processes, manuscript, 1965. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD123.html

[3] G. L. Peterson: "Myths About the Mutual Exclusion Problem", Information Processing Letters 12(3) 1981, 115–116

[4] Andrew S. Tanenbaum: Modern Operating Systems

[5] LaTeX-Projektseite http://www.latex-project.org/

[6] Homepage der „Deutschsprachige Anwendervereinigung TeX e.V.“ http://www.dante.de/

[7] Homepage von Doxygen http://www.doxygen.org/

[8] Peter Mösgen (1998): Makeindex – Sachregister erstellen mit LaTeX

[9] Information Technology - AT Attachment with Packet Interface Extension (ATA/ATAPI-4), 19 August 1998

Anhang

A Doxygen

A.1 Hierarchical Index

A.1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

io::vfs::test::AbstractDummyNode 263
  io::vfs::test::DummyLeaf 474
  io::vfs::test::DummyNode 475
task::pipeandfilter::AbstractFilter< T, V > 263
task::pipeandfilter::AbstractFilter< keycode::Keycode, char > 263
  io::driver::charlayout::CharLayoutFilter 380
task::pipeandfilter::AbstractFilter< pstwokeyboard::Scancode, Keycode >263
  io::driver::keycode::KeycodeFilter 588
io::driver::keycode::scancodestates::AbstractScanCodeConverterState . 265
  io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverter-
    State 483
  io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverter-
    State 485
  io::driver::keycode::scancodestates::EZeroScanCodeConverterState . 489
  io::driver::keycode::scancodestates::InitialScanCodeConverterState . 563
memory::lmm::AddressSpaceRange 269
memory::allocator::Allocator 286
ipc::Array< T > 292
ipc::Array< T const > 292
io::driver::block::ata::commands::AtaCommand 312
  io::driver::block::ata::commands::AtaIdentifyCommand 325
  io::driver::block::ata::commands::AtaReadCommand 330
  io::driver::block::ata::commands::AtaWriteCommand 337
io::driver::block::ata::AtaCommandRegister 317
io::driver::block::ata::AtaDigitalOutputRegister 323
io::driver::block::ata::AtaDriveLbaHighest 323

io::driver::block::ata::AtaErrorRegister 325
io::driver::block::ata::AtaIdentifyResult 327
io::driver::block::ata::AtaIoPorts 329
io::driver::block::ata::AtaStatusRegister 334
ipc::Attribute< T > 339
ipc::Attribute< Array< T > > 340
ipc::Attribute< Array< T const > > 342
ipc::Attribute< ipc::Array > 339
ipc::Attribute< P1Type_add > 339
ipc::Attribute< P1Type_increment > 339
ipc::Attribute< P1Type_incrementInPlace > 339
ipc::Attribute< P2Type_add > 339
ipc::Attribute< RetType_add > 339
ipc::Attribute< RetType_getAnswer > 339
ipc::Attribute< RetType_increment > 339
ipc::Attribute< T & > 343
ipc::Attribute< T const > 344
ipc::Attribute< T const & > 346
tool::BitField< Base, Index, BitPosition, Length > 348
io::driver::block::Block 350
boot::BootManager 364
io::vfs::fat::Bootsector 367
BPB 375
RootDirectory::Entry::Callback 377
api::io::console::ClearScreenCommand 385
fosCli::parser::CliCommand 387
fosCli::commands::ClearCliCommand 384
fosCli::commands::EchoCliCommand 478
fosCli::commands::HelpCliCommand 560
fosCli::commands::KernelVersionCliCommand 584
fosCli::commands::TimeCliCommand 884
fosCli::parser::CliCommandCreator 388
fosCli::commands::ClearCliCommandCreator 384
fosCli::commands::EchoCommandCreator 478
fosCli::commands::HelpCliCommandCreator 561
fosCli::commands::KernelVersionCliCommandCreator 584
fosCli::commands::TimeCliCommandCreator 884
ipc::test::FantasticObjectProxy::Command< Index > 493
ipc::test::FantasticObjectProxy::CommandBase 402

ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-add > 397
ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-getAnswer > 398
ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-increment > 399
ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-incrementElements > 399
ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-incrementInPlace > 400
ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_-print > 401
fosCli::parser::CommandDirectory 403
fosCli::parser::CommandDirectoryMaster 404
fosCli::parser::CommandDirectorySlave 405
ipc::CommandRelay 405
tool::collection::Comparator< T > 407
tool::collection::Comparator< char const * > 408
lib::Console 409
io::console::ConsoleManager 410
tool::collection::ConstIterator< T > 415
tool::collection::ConstArrayListIterator< T > 412
tool::collection::ConstLinkedListIterator< T > 416
task::spinlock::test::Counter 419
memory::allocator::Allocator::CriticalSection 420
io::driver::graphics::Cursor 423
io::driver::graphics::vga::VgaCursor 903
io::driver::graphics::Delta 429
io::driver::graphics::Position 727
Descriptor 940
memory::Descriptor 432
memory::CodeOrDataSegmentDescriptor 389
memory::SystemSegmentDescriptor 840
memory::GateDescriptor 552
memory::TSSDescriptor 888
io::driver::Device 436
io::driver::block::ata::AtaBusDevice 305
io::driver::block::BlockDevice 351
io::driver::block::ata::AtaDevice 318
io::driver::charlayout::CharLayoutDevice 378

io::driver::classes::AudioClass 347
io::driver::speaker::SpeakerDevice 823
io::driver::graphics::TextDevice 862
io::driver::graphics::vga::VgaTextDevice 906
io::driver::interrupt::PICDevice 708
io::driver::interrupt::PICController 704
io::driver::interrupt::MasterPICController 651
io::driver::interrupt::SlavePICController 821
io::driver::keycode::KeycodeDevice 586
io::driver::pstwo::PS2Device 749
io::driver::pstwokeyboard::PS2KeyboardDevice 757
io::driver::rtc::RTCDevice 793
io::driver::test::TestDevice 851
io::driver::test::TestDevice2 852
io::driver::timer::PIT8254Device 722
io::driver::DeviceClass 437
io::driver::DeviceManager 437
io::vfs::fat::DirectoryEntry 447
Disk 448
DiskAddress 451
io::driver::Driver 455
io::driver::block::ata::AtaBusDriver 308
io::driver::block::ata::AtaDriver 324
io::driver::charlayout::CharLayoutDriver 380
io::driver::graphics::vga::VgaDriver 905
io::driver::interrupt::PICDriver 711
io::driver::keycode::KeycodeDriver 587
io::driver::pstwo::PS2Driver 757
io::driver::pstwokeyboard::PS2KeyboardDriver 760
io::driver::rtc::RTCDriver 797
io::driver::speaker::SpeakerDriver 823
io::driver::test::TestDriver 852
io::driver::test::TestDriver2 853
io::driver::timer::PIT8254Driver 726
io::driver::DriverManager::DriverDescriptor 457
io::driver::DriverManager 459
E820_entry 954
memory::E820_entry 476
RootDirectory::Entry 479
memory::allocator::Environment 481

lib::runtime::UserEnvironment 893
memory::KernelEnvironment 582
exception
lib::System::SystemCallFailed 839
ipc::test::FantasticObject 492
ipc::test::FantasticObjectImpl 493
ipc::test::FantasticObjectProxy 493
memory::FarPointer 496
FAT 497
io::vfs::fat::FatAccess 506
FatObject 519
io::vfs::fat::FatDirectory 509
io::vfs::fat::FatRootDirectory 519
FAT::FATType 524
FAT::FAT12Type 502
FAT::FAT16Type 504
io::vfs::FileStream 528
io::vfs::fat::FatFileStream 515
io::vfs::test::DummyFileStream 469
io::vfs::FileSystem 532
io::vfs::fat::FatFileSystem 517
io::vfs::test::DummyFileSystem 472
io::vfs::FileSystemNodeVisitor 538
fosCli::FosCli 540
fosCli::scanner::elements::FosCliElement 541
fosCli::scanner::elements::FosCliStringElement 548
fosCli::scanner::elements::FosCliElementVisitor 542
fosCli::parser::Parser 696
fosCli::scanner::FosCliScanner 544
fosCli::scanner::FosCliScannerState 545
fosCli::scanner::states::FosCliExecuteScannerState 542
fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState 543
fosCli::scanner::FosCliScannerStateResult 548
cpu::GDTManager 555
cpu::interrupt::InterruptHandler 567
cpu::interrupt::DefaultExceptionHandler 427
cpu::interrupt::DefaultHandler 428
io::driver::interrupt::PICInterruptHandler 712
memory::paging::PageFaultExceptionHandler 684
cpu::interrupt::InterruptHandlerContainer 568

cpu::interrupt::InterruptManager 569
io::IOPort 573
io::driver::interrupt::IRQHandler 578
io::driver::block::ata::AtaBusIrqHandler 309
io::driver::pstwokeyboard::PS2KeyboardHandler 761
io::driver::rtc::RTCHandler 797
task::scheduler::TimerInterruptHandler 884
tool::collection::Iterator< T > 579
tool::collection::ArrayListIterator< T > 302
tool::collection::LinkedListIterator< T > 624
io::driver::keycode::Keycode 585
io::driver::charlayout::keymap_entry 589
tool::collection::KeyValuePair< Key, Value, KeyComp, ValueComp > . . . 590
io::driver::block::LBAddress 591
cpu::level::LevelManager 592
memory::lmm::LinearAddressSpaceManager 597
tool::collection::LinkedListEntry< T > 620
tool::collection::LinkedListEntry< cpu::level::TransitionHandler * > . . . 620
tool::collection::LinkedListEntry< fosCli::parser::CliCommandCreator * > 620
tool::collection::LinkedListEntry< InterruptHandler * > 620
tool::collection::LinkedListEntry< io::time::WaitingQueueEntry * > 620
tool::collection::LinkedListEntry< IRQHandler * > 620
tool::collection::LinkedListEntry< lock::Semaphore * > 620
tool::collection::LinkedListEntry< Resource > 620
tool::collection::LinkedListEntry< task::Thread * > 620
tool::collection::LinkedListEntry< Tasklet * > 620
tool::collection::LinkedListEntry< Thread * > 620
tool::collection::List< T > 628
tool::collection::ArrayList< DriverDescriptor > 294
tool::collection::ArrayList< int > 294
tool::collection::ArrayList< KeyValuePair< int, Device *, Comparator< int >, Comparator< Device * > > > 294
tool::collection::LinearMap< int, Device * > 606
tool::collection::ArrayList< KeyValuePair< Key, Value, KeyComp, ValueComp > > 294
tool::collection::LinearMap< Key, Value, KeyComp, ValueComp > . 606
tool::collection::ArrayList< KeyValuePair< uint32_t, task::Process *, Comparator< uint32_t >, Comparator< task::Process * > > > 294
tool::collection::LinearMap< uint32_t, task::Process * > 606

tool::collection::ArrayList< KeyValuePair< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > > 294
tool::collection::LinearMap< uint32_t, void * > 606
tool::collection::ArrayList< memory::paging::PageTable * > 294
tool::collection::LinkedList< cpu::level::TransitionHandler * > 613
tool::collection::LinkedList< fosCli::parser::CliCommandCreator * > . 613
tool::collection::LinkedList< InterruptHandler * > 613
tool::collection::LinkedList< io::time::WaitingQueueEntry * > 613
tool::collection::LinkedList< IRQHandler * > 613
tool::collection::LinkedList< lock::Semaphore * > 613
tool::collection::LinkedList< Resource > 613
tool::collection::LinkedList< task::Thread * > 613
tool::collection::LinkedList< Tasklet * > 613
tool::collection::LinkedList< Thread * > 613
tool::collection::ArrayList< T > 294
tool::collection::LinkedList< T > 613
tool::collection::List< elements::FosCliElement * > 628
tool::collection::List< io::vfs::fat::DirectoryEntry > 628
tool::collection::List< io::vfs::FileSystemNode * > 628
tool::collection::List< io::vfs::test::AbstractDummyNode * > 628
tool::collection::List< scanner::elements::FosCliElement * > 628
task::lock::Semaphore::Lock 636
task::lock::SpinLock::Lock 637
Logger 640
tool::collection::Map< Key, Value, KeyComp, ValueComp > 642
tool::collection::LinearMap< Key, Value, KeyComp, ValueComp > . . 606
tool::collection::Map< int, Device *, Comparator< int >, Comparator< Device * > > 642
tool::collection::LinearMap< int, Device * > 606
tool::collection::Map< uint32_t, task::Process *, Comparator< uint32_t >, Comparator< task::Process * > > 642
tool::collection::LinearMap< uint32_t, task::Process * > 606
tool::collection::Map< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > 642
tool::collection::LinearMap< uint32_t, void * > 606
ipc::MapRequest 647
tool::MD5 654
tool::collection::Queue< T >::Node 659
task::pipeandfilter::Pipe< T >::Node 660
memory::allocator::Page 663

memory::paging::PageContext 668
memory::paging::PageDirectory 672
memory::paging::PageEntry 682
memory::paging::PageFlags 685
memory::allocator::PageHeader 688
memory::paging::PageTable 689
memory::allocator::PageTable 690
memory::allocator::Page::PartDescriptor 697
ipc::Participant 698
memory::pmm::PhysicalMemoryManager 699
task::pipeandfilter::Pipe< T > 715
task::pipeandfilter::test::TesterPipe< T > 854
task::pipeandfilter::Pipe< char > 715
task::pipeandfilter::Pipe< io::driver::keycode::io::driver::keycode::Keycode > 715
task::pipeandfilter::Pipe< io::driver::keycode::Keycode > 715
task::pipeandfilter::Pipe< V > 715
io::driver::timer::PIT8254ControlWord 717
io::driver::timer::PIT8254Counter 719
memory::pmm::PmmTableEntry 727
task::Process 737
task::ProcessManager 744
task::ProgramLoader 747
tool::collection::Queue< T > 762
tool::collection::Queue< commands::AtaCommand * > 762
tool::collection::Queue< io::driver::pstwokeyboard::Scancode > 762
tool::collection::Queue< Scancode > 762
tool::collection::Queue< task::Process::Request > 762
object::Ref< T > 765
object::Ref< io::vfs::File > 765
object::Ref< io::vfs::Mount > 765
object::RefCountedObject 769
io::vfs::AbstractMount 263
io::vfs::Mount 656
io::vfs::fat::FatMount 518
io::vfs::test::DummyMount 474
io::vfs::FileSystemNode 534
io::vfs::Directory 440
io::vfs::fat::FatDirectory 509
io::vfs::fat::FatRootDirectory 519

io::vfs::test::DummyDirectory 462
io::vfs::File 525
io::vfs::fat::FatFile 513
io::vfs::test::DummyFile 467
ipc::Registry 771
ipc::RemoteObject 773
ipc::Request 775
api::io::console::ClearScreenRequest 386
api::io::console::OutputStringRequest 662
api::io::console::ReadKeyRequest 764
api::io::time::CurrentTimeRequest 422
api::kernel::GetVersionRequest 559
api::memory::lmm::AllocBlockRequest 290
api::memory::lmm::FreeBlockRequest 550
api::task::LoadProgramRequest 633
api::task::lock::CreateSemaphoreRequest 419
api::task::lock::ModifySemaphoreRequest 655
api::task::ProcessIdRequest 744
api::task::scheduler::ExitThreadRequest 489
ipc::test::FantasticObjectProxy::RequestBase 784
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_add > 779
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_getAnswer > 780
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_increment > 780
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementElements > 781
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementInPlace > 782
ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_print > 783
ipc::test::FantasticObjectProxy::Request< Index > 493
task::Process::Request 777
ipc::test::FantasticObjectProxy::Request< 0 > 778
memory::lmm::LinearAddressSpaceManager::ReservationTable 785
task::Thread::Resource 786
ipc::Result< T > 788
ipc::Result< Array< T > > 789
ipc::Result< void > 790

RootDirectory 791
task::scheduler::Scheduler 801
memory::Selector 811
task::lock::Semaphore 816
lib::Semaphore 819
task::lock::SpinLock 824
boot::SplashScreen 828
ipc::SysCallHandler 830
lib::System 835
task::tasklet::Tasklet 842
boot::CleanupTasklet 383
io::driver::block::ata::AtaTasklet 334
io::driver::keycode::KeycodeTasklet 588
io::driver::rtc::RTCTasklet 799
task::tasklet::test::TestTasklet 856
task::tasklet::TaskletManager 844
task::TaskStateSegment 846
runtime::TCB 847
test::TestCase 850
io::driver::block::ata::test::AtaReadWriteTest 332
io::driver::block::ata::test::AtaReadWriteTest2 333
io::driver::block::ata::test::AtaReadWriteTest3 333
io::driver::charlayout::test::CharLayoutTestCase 382
io::driver::graphics::vga::test::VgaCharTestCase 903
io::driver::keycode::test::KeyboardTestCase 585
io::driver::test::DriverFrameworkTestCase 459
io::vfs::test::VFSTestCase 901
ipc::test::IPCTestCase 578
ipc::test::SysCallTestCase 834
memory::allocator::test::AllocatorOutOfMemoryTestCase 288
memory::allocator::test::AllocatorTestCase 289
memory::lmm::test::LASMTest 591
memory::paging::test::PagingTestCase 695
object::test::RefTestCase 770
runtime::test::ExceptionTestCase 487
runtime::test::RTTITestCase 800
task::priorityinheritance::test::PriorityInheritanceTestCase 737
task::semaphore::test::SemaphoreTestCase 820
task::semaphore::test::ThreadASemaphoreTestCase 875
task::spinlock::test::SpinLockTestCase 827

test::memory::allocator::AllocatorTestCase 289
tool::collection::test::ArrayListTestCase 304
tool::collection::test::LinearMapTestCase 612
tool::collection::test::LinkedListTestCase 627
tool::test::StringTestCase 829
test::TestManager 855
io::driver::graphics::TextChar 857
io::console::TextConsole 860
boot::BootConsole 359
io::console::DefaultConsole 423
task::Thread 865
boot::BootThread 373
boot::UserSpaceInitThread 896
ClockThread 388
io::driver::block::ata::AtaBusWorkerThread 311
ipc::test::IPCReceiverThread 575
ipc::test::IPCSenderThread 576
LockTesterThreadDown 638
LockTesterThreadUp 639
task::pipeandfilter::FilterThread< T, V > 539
task::priorityinheritance::test::HighPrioThread 561
task::priorityinheritance::test::LowPrioThread 640
task::priorityinheritance::test::NormalThread 660
task::scheduler::IdleThread 562
task::semaphore::test::ThreadASemaphoreTestCase 875
task::spinlock::test::SpinLockTestThread 827
task::UserModeThread 895
tool::Terminal 849
task::scheduler::ThreadQueue 876
task::ThreadStack 879
io::time::Time 881
cpu::level::TransitionHandler 886
io::driver::interrupt::MaskingHandler 648
task::scheduler::SchedulerHandler 809
TypeInfo< T > 968
TypeInfo< int16_t > 891
TypeInfo< int32_t > 891
TypeInfo< int64_t > 891
TypeInfo< int8_t > 891
TypeInfo< uint16_t > 892

TypeInfo< uint32_t > 892
TypeInfo< uint64_t > 892
TypeInfo< uint8_t > 893
uint128_t 968
io::vfs::VFSManager 897
io::driver::graphics::vga::VgaChar 901
io::vfs::Volume 909
io::vfs::BlockDeviceVolume 355
io::time::WaitingQueue 914
io::time::WaitingQueueEntry 915

A.2 Data Structure Index

A.2.1 Data Structures

Here are the data structures with brief descriptions:

io::vfs::test::AbstractDummyNode 263
task::pipeandfilter::AbstractFilter< T, V > 263
io::vfs::AbstractMount
Interface for all methods, that should be supported by a mount . 263
io::driver::keycode::scancodestates::AbstractScanCodeConverterState
The ScanCodeConverterState is an abstract class for a State pattern to manage the special cases E0 and E1 which are used in the hardware keyboard driver for compatibility reasons 265
memory::lmm::AddressSpaceRange
Represents a range of linear address space managed by a **LinearAddressSpaceManager** (p. 597) 269
memory::allocator::Allocator
An **Allocator** (p. 286) object manages a heap 286
memory::allocator::test::AllocatorOutOfMemoryTestCase
Tests whether operator new throws an exception in out-of-memory situations 288
test::memory::allocator::AllocatorTestCase 289
memory::allocator::test::AllocatorTestCase 289
api::memory::lmm::AllocBlockRequest
Request for achieving an amount of space from the LinearAddressSpaceManager 290

ipc::Array< T >
Encapsulates an array for IPC 292
tool::collection::ArrayList< T >
Implements a list with an automatic growing and shrinking array 294
tool::collection::ArrayListIterator< T >
Iterator (p. 579) for ArrayLists 302
tool::collection::test::ArrayListTestCase 304
io::driver::block::ata::AtaBusDevice
Class which represent a ata bus 305
io::driver::block::ata::AtaBusDriver
Driver (p. 455) which creates the AtaBus 308
io::driver::block::ata::AtaBusIrqHandler
Interrupt handler for Interrupt which are thrown from an ATA-Bus 309
io::driver::block::ata::AtaBusWorkerThread
ATA bus worker thread which continuously requests the ATA bus device to initiate the processing of pending ATA commands . . . 311
io::driver::block::ata::commands::AtaCommand
Represents a command which can be sent to a hard disk over the ATA bus 312
io::driver::block::ata::AtaCommandRegister 317
io::driver::block::ata::AtaDevice
A storage block device which is connected to an ATA-Bus 318
io::driver::block::ata::AtaDigitalOutputRegister 323
io::driver::block::ata::AtaDriveLbaHighest 323
io::driver::block::ata::AtaDriver
Driver (p. 455) which creates the AtaDevices 324
io::driver::block::ata::AtaErrorRegister 325
io::driver::block::ata::commands::AtaIdentifyCommand
Command for executing the identify command 325
io::driver::block::ata::AtaIdentifyResult
Result of an AtaIdentifyCommand 327
io::driver::block::ata::AtaIoPorts
The io ports of an **AtaBusDevice** (p. 305) 329
io::driver::block::ata::commands::AtaReadCommand
Command which execute a read sectors on the ata bus 330
io::driver::block::ata::test::AtaReadWriteTest 332
io::driver::block::ata::test::AtaReadWriteTest2 333
io::driver::block::ata::test::AtaReadWriteTest3 333
io::driver::block::ata::AtaStatusRegister 334

io::driver::block::ata::AtaTasklet
Tasklet for the ATA Bus **Device** (p. 436) 334
io::driver::block::ata::commands::AtaWriteCommand
Command which execute a write sectors command on the ata bus 337
ipc::Attribute< **T** >
Encapsulates a normal attribute in a **Request** (p. 775) 339
ipc::Attribute< **Array**< **T** > >
Encapsulates an array attribute in a **Request** (p. 775) 340
ipc::Attribute< **Array**< **T const** > >
Encapsulates an array-of-const attribute in a **Request** (p. 775) . 342
ipc::Attribute< **T &** >
Encapsulates a reference attribute in a **Request** (p. 775) 343
ipc::Attribute< **T const** >
Encapsulates a const attribute in a **Request** (p. 775) 344
ipc::Attribute< **T const &** >
Encapsulates a reference-to-const attribute in a **Request** (p. 775) 346
io::driver::classes::AudioClass
Top Class for all Audio **Device** (p. 436) like PC Speaker 347
tool::BitField< **Base, Index, BitPosition, Length** >
Template for simple reading/writing bit fields from arrays 348
io::driver::block::Block
A struct which represents a 512 byte large block of a block-oriented device 350
io::driver::block::BlockDevice
Class which represent a block-oriented storage device 351
io::vfs::BlockDeviceVolume
Implementation of kernel/io/vfs/Volume for FAT16 355
boot::BootConsole
Class for the output during the boot 359
boot::BootManager
Controls the boot process 364
io::vfs::fat::Bootsector 367
boot::BootThread
Performs the second part of the boot process in a separate boot thread 373
BPB
Describes a **BPB** (p. 375) (BIOS parameter block) 375
RootDirectory::Entry::Callback
Describes a callback. This is an interface 377

io::driver::charlayout::CharLayoutDevice
Device (p. 436) for a Char Layout 378
io::driver::charlayout::CharLayoutDriver
Driver (p. 455) for a Char Layout 380
io::driver::charlayout::CharLayoutFilter
Filter to translate Keycodes to chars 380
io::driver::charlayout::test::CharLayoutTestCase 382
boot::CleanupTasklet
Cleans up some resources which are only used while booting . . 383
fosCli::commands::ClearCliCommand 384
fosCli::commands::ClearCliCommandCreator 384
api::io::console::ClearScreenCommand
Handles **ClearScreenRequest** (p. 386) 385
api::io::console::ClearScreenRequest
Remove all content from the screen 386
fosCli::parser::CliCommand
Abstract class for commands called through the CLI 387
fosCli::parser::CliCommandCreator 388
ClockThread 388
memory::CodeOrDataSegmentDescriptor
Describes a code or data segment descriptor 389
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_add** > 397
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_getAnswer** > 398
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_increment** > 399
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_incrementElements** > 399
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_incrementInPlace** > 400
ipc::test::FantasticObjectProxy::Command< **FantasticObjectProxy-::Id_print** > 401
ipc::test::FantasticObjectProxy::CommandBase 402
fosCli::parser::CommandDirectory 403
fosCli::parser::CommandDirectoryMaster 404
fosCli::parser::CommandDirectorySlave 405
ipc::CommandRelay
A **CommandRelay** (p. 405) is responsible for sending requests from one process to another or to the kernel 405

tool::collection::Comparator< T >
Abstracts from operator==() 407
tool::collection::Comparator< char const * >
Specializes **Comparator** (p. 407) for char const *, using strcmp() for the comparison 408
lib::Console 409
io::console::ConsoleManager
<<singleton>> Class for switching consoles during boot 410
tool::collection::ConstArrayListIterator< T >
Iterator (p. 579) for ArrayLists 412
tool::collection::ConstIterator< T >
Basic definition of an iterator over an unmodifiable collection . . 415
tool::collection::ConstLinkedListIterator< T >
Implementation of **tool::collection::Iterator** (p. 579) 416
task::spinlock::test::Counter 419
api::task::lock::CreateSemaphoreRequest
Creates a Semaphore 419
memory::allocator::Allocator::CriticalSection
Helper class for entering/leaving the **Allocator** (p. 286)'s critical section 420
api::io::time::CurrentTimeRequest
Returns the current time values which are provided by the time service 422
io::driver::graphics::Cursor
Class encapsulating a cursor 423
io::console::DefaultConsole
The Console before the VGA Device initialized 423
cpu::interrupt::DefaultExceptionHandler
Exception Handler Class Currently only one default now which prints out "Unhadled Exception!" and halts the system 427
cpu::interrupt::DefaultHandler
Default Handler Class 428
io::driver::graphics::Delta
Represents a difference between two positions 429
memory::Descriptor
Describes a generic descriptor 432
io::driver::Device
Abstract class which represents all devices in the system 436
io::driver::DeviceClass
Abstract device class 437

io::driver::DeviceManager
As a Singleton the **DeviceManager** (p. 437) is responsible for handling the addition and removing of devices 437
io::vfs::Directory 440
io::vfs::fat::DirectoryEntry 447
Disk
Represents a disk 448
DiskAddress
Represents a disk address 451
io::driver::Driver
Abstract class **Driver** (p. 455) represents all drivers in a system 455
io::driver::DriverManager::DriverDescriptor
Describes a **Driver** (p. 455) within the **DriverManager** (p. 459) 457
io::driver::test::DriverFrameworkTestCase
Test of test 459
io::driver::DriverManager
As a Singleton the **DriverManager** (p. 459) is responsible for handling the addition and removing of drivers 459
io::vfs::test::DummyDirectory 462
io::vfs::test::DummyFile 467
io::vfs::test::DummyFileStream 469
io::vfs::test::DummyFileSystem 472
io::vfs::test::DummyLeaf 474
io::vfs::test::DummyMount 474
io::vfs::test::DummyNode 475
memory::E820_entry
Describes a memory block 476
fosCli::commands::EchoCliCommand
Writes the given parameter separated with a space to the console 478
fosCli::commands::EchoCommandCreator 478
RootDirectory::Entry
Describes a directory entry 479
memory::allocator::Environment
Allows the **Allocator** (p. 286) to communicate with its environment 481
io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverter-State
Handles the extended scancodes which are introduced through e1 483

io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverter-State
Handles the extended scancodes which are introduced through e1 485
runtime::test::ExceptionTestCase
Tests exception handling in kernel mode 487
api::task::scheduler::ExitThreadRequest
Exits the current thread 489
io::driver::keycode::scancodestates::EZeroScanCodeConverterState
Handles the extended scancodes(Scancode Set 1) which are introduced through e0 489
ipc::test::FantasticObject
Test interface for IPC 492
ipc::test::FantasticObjectImpl
Implementation of the IPC interface 493
ipc::test::FantasticObjectProxy 493
memory::FarPointer
Struct to access far jumps 496
FAT
Represents a **FAT** (p. 497) (File Allocation Table) 497
FAT::FAT12Type
Handles FAT12 502
FAT::FAT16Type
Handles FAT16 504
io::vfs::fat::FatAccess
Access to Hard **Disk** (p. 448) for FAT16 506
io::vfs::fat::FatDirectory 509
io::vfs::fat::FatFile 513
io::vfs::fat::FatFileStream 515
io::vfs::fat::FatFileSystem 517
io::vfs::fat::FatMount 518
FatObject
Abstraction for FATFile and FATDirectory 519
io::vfs::fat::FatRootDirectory
Class for the root directory of a FAT16 519
FAT::FATType
FAT (p. 497) type abstraction 524
io::vfs::File
Abstract file 525

io::vfs::FileStream
Read and write files 528
io::vfs::FileSystem
The abstract representation of a file system 532
io::vfs::FileSystemNode
The abstract representation of a file system node 534
io::vfs::FileSystemNodeVisitor
Visitor for file system nodes 538
task::pipeandfilter::FilterThread< **T, V** > 539
fosCli::FosCli
FHDW-OS Super Command Line Interpreter 540
fosCli::scanner::elements::FosCliElement 541
fosCli::scanner::elements::FosCliElementVisitor
Visitor interface for FosCliElements 542
fosCli::scanner::states::FosCliExecuteScannerState 542
fosCli::scanner::states::FosCliReadSpaceDemelitedSignScanner-State 543
fosCli::scanner::FosCliScanner
Scanning the input of the keyboard 544
fosCli::scanner::FosCliScannerState
A state of the scanner 545
fosCli::scanner::FosCliScannerStateResult 548
fosCli::scanner::elements::FosCliStringElement 548
api::memory::lmm::FreeBlockRequest
A **FreeBlockRequest** (p. 550) represents the desire of freeing an amount of memory 550
memory::GateDescriptor
Describes a gate descriptor 552
cpu::GDTManager
<<singleton>> Class for managing the Global **Descriptor** (p. **??**) Table (GDT) 555
api::kernel::GetVersionRequest
Determines the kernel version 559
fosCli::commands::HelpCliCommand 560
fosCli::commands::HelpCliCommandCreator 561
task::priorityinheritance::test::HighPrioThread
A high prioritized thread that wants to access a resource that a lower prioritized thread has locked exclusively 561
task::scheduler::IdleThread
This **Thread** (p. 865) runs if no other **Thread** (p. 865) is runnable 562

io::driver::keycode::scancodestates::InitialScanCodeConverterState
Standard state handles the most keys like letters and digits and the most other characters (scancode set 1) 563
cpu::interrupt::InterruptHandler 567
cpu::interrupt::InterruptHandlerContainer 568
cpu::interrupt::InterruptManager
Manages the handler for the hardware interrupts 569
io::IOPort
Represents an I/O port 573
ipc::test::IPCReceiverThread
IPC receiver 575
ipc::test::IPCSenderThread
IPC sender 576
ipc::test::IPCTestCase
Tests interprocess communication (IPC) 578
io::driver::interrupt::IRQHandler
Encapsulates an IRQ handler 578
tool::collection::Iterator< T >
Basic definition of an iterator over a modifiable collection 579
memory::KernelEnvironment
Implements an Allocator environment for the kernel mode 582
fosCli::commands::KernelVersionCliCommand 584
fosCli::commands::KernelVersionCliCommandCreator 584
io::driver::keycode::test::KeyboardTestCase 585
io::driver::keycode::Keycode
Definition of a **Keycode** (p. 585) 585
io::driver::keycode::KeycodeDevice
A **Driver** (p. 455) for the translation of scancodes in keycodes . 586
io::driver::keycode::KeycodeDriver
Driver (p. 455) for the **KeycodeDevice** (p. 586) 587
io::driver::keycode::KeycodeFilter
Filter between the Scancode and **Keycode** (p. 585) buffer 588
io::driver::keycode::KeycodeTasklet 588
io::driver::charlayout::keymap_entry 589
tool::collection::KeyValuePair< Key, Value, KeyComp, ValueComp > 590
memory::lmm::test::LASMTest
Tests the **LinearAddressSpaceManager** (p. 597) 591
io::driver::block::LBAddress
Structure which represents a single Logic **Block** (p. 350) Address 591

cpu::level::LevelManager
Manages the levels and transitions between the levels <<singleton>> 592
memory::lmm::LinearAddressSpaceManager
The **LinearAddressSpaceManager** (p. 597) (LASM) manages the linear address space of one process 597
tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >
LinearMap (p. 606) is an implementation of the interface **Map** (p. 642) 606
tool::collection::test::LinearMapTestCase 612
tool::collection::LinkedList< T >
Implementation of **List** (p. 628) 613
tool::collection::LinkedListEntry< T >
A **LinkedListEntry** (p. 620) is the elements which encapsulates the original value stored in a link list 620
tool::collection::LinkedListIterator< T >
Implementation of **tool::collection::Iterator** (p. 579) 624
tool::collection::test::LinkedListTestCase 627
tool::collection::List< T >
Basic definition of a list 628
api::task::LoadProgramRequest
A **LoadProgramRequest** (p. 633) represents the desire to load an application 633
task::lock::Semaphore::Lock
Locker class which acquires a semaphore in the constructor and releases it in the destructor 636
task::lock::SpinLock::Lock
Locker class which acquires a spin lock in the constructor and releases it in the destructor 637
LockTesterThreadDown 638
LockTesterThreadUp 639
Logger
Logger (p. 640) to log messages of different priorities on the console 640
task::priorityinheritance::test::LowPrioThread
A low prioritized thread that has access to a resource that a higher prioritized thread wants to have access to 640
tool::collection::Map< Key, Value, KeyComp, ValueComp >
Provides a mechanism to map a key to a value 642

ipc::MapRequest
Encapsulates a request to map some memory region into some target process . . . 647
io::driver::interrupt::MaskingHandler . . . 648
io::driver::interrupt::MasterPICController . . . 651
tool::MD5 . . . 654
api::task::lock::ModifySemaphoreRequest
Modifies a Semaphore . . . 655
io::vfs::Mount
Represents an actual mount . . . 656
tool::collection::Queue< T >::Node . . . 659
task::pipeandfilter::Pipe< T >::Node . . . 660
task::priorityinheritance::test::NormalThread
A normal thread doing some computations, running at PRIO_NORMAL . . . 660
api::io::console::OutputStringRequest
Prints a string on the console . . . 662
memory::allocator::Page
A **Page** (p. 663) applies sub-allocation techniques in order to divide a page into smaller blocks . . . 663
memory::paging::PageContext . . . 668
memory::paging::PageDirectory
Encapsulates a page directory . . . 672
memory::paging::PageEntry . . . 682
memory::paging::PageFaultExceptionHandler
Handles page fault exceptions . . . 684
memory::paging::PageFlags
Encapsulates flags for a page . . . 685
memory::allocator::PageHeader
Put at the beginning of a managed page . . . 688
memory::paging::PageTable . . . 689
memory::allocator::PageTable
PageTables constitute the first-order organization structure of an **Allocator** (p. 286) . . . 690
memory::paging::test::PagingTestCase . . . 695
fosCli::parser::Parser . . . 696
memory::allocator::Page::PartDescriptor
If valid, a **PartDescriptor** (p. 697) describes a contiguous range of parts that are either allocated or free . . . 697

ipc::Participant
Represents an IPC participant 698
memory::pmm::PhysicalMemoryManager
Manages the physical memory 699
io::driver::interrupt::PICController 704
io::driver::interrupt::PICDevice
Top Class for both PIC Devices (Primary and Slave) 708
io::driver::interrupt::PICDriver
Driver (p. 455) for both PIC Devices 711
io::driver::interrupt::PICInterruptHandler
Interrupt handler installed by the PIC devices that automatically send EOI commands to the relevant PICs 712
task::pipeandfilter::Pipe$<$ **T** $>$
Pipe (p. 715) for pipe and filter pattern 715
io::driver::timer::PIT8254ControlWord
This class produces the control word for the Programmable Interval Timer Intel 82c54 717
io::driver::timer::PIT8254Counter
Represents one counter of the Intel 82c54 Programmable Interval Timer 719
io::driver::timer::PIT8254Device
Implementation of the Intel 82c54 Programmable Interval Timer 722
io::driver::timer::PIT8254Driver
Driver (p. 455) for the Programmable Interval Timer Intel 82c54 726
memory::pmm::PmmTableEntry 727
io::driver::graphics::Position
Represents a position 727
task::priorityinheritance::test::PriorityInheritanceTestCase . . . 737
task::Process
A **Process** (p. 737) represents a collection of Threads 737
api::task::ProcessIdRequest 744
task::ProcessManager 744
task::ProgramLoader
The singleton class **ProcessManager** (p. 744) is responsible for loading an application 747
io::driver::pstwo::PS2Device
Device (p. 436) for the PS2 Controller 749
io::driver::pstwo::PS2Driver
Driver (p. 455) for the PS/2 **Device** (p. 436) 757

io::driver::pstwokeyboard::PS2KeyboardDevice
Device (p. 436) for the PS/2 Keyboard 757
io::driver::pstwokeyboard::PS2KeyboardDriver
Driver (p. 455) for the PS/2 Keyboard **Device** (p. 436) 760
io::driver::pstwokeyboard::PS2KeyboardHandler
The **PS2KeyboardHandler** (p. 761) handles the interrupts which are fired when a key on the PS/2 keyboard is pressed or released 761
tool::collection::Queue< **T** >
Queue (p. 762) class which does not use any locking 762
api::io::console::ReadKeyRequest
Request for reading one character from the console 764
object::Ref< **T** >
Represents a reference to a reference-counted object 765
object::RefCountedObject
Represents a reference-counted object 769
object::test::RefTestCase
Tests reference-counted objects 770
ipc::Registry
Manages remotely accessible objects 771
ipc::RemoteObject 773
ipc::Request
Represents a request 775
task::Process::Request
Encapsulates an IPC request and a Semaphore 777
ipc::test::FantasticObjectProxy::Request< **0** > 778
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_add** > 779
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_getAnswer** > 780
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_increment** > 780
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_incrementElements** > 781
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_incrementInPlace** > 782
ipc::test::FantasticObjectProxy::Request< **FantasticObjectProxy-::Id_print** > 783
ipc::test::FantasticObjectProxy::RequestBase 784
memory::lmm::LinearAddressSpaceManager::ReservationTable
Holds entries for address space management 785

task::Thread::Resource
Associates acquired Semaphores with Threads waiting for them 786
ipc::Result< T >
Encapsulates the result of an IPC request 788
ipc::Result< Array< T > >
Encapsulates an **Array** (p. 292) result of an IPC request 789
ipc::Result< void >
Encapsulates a void result of an IPC request 790
RootDirectory
Represents the root directory 791
io::driver::rtc::RTCDevice
Device (p. 436) for the Real Time Clock (RTC) The RTC is a part of the CMOS RAM 793
io::driver::rtc::RTCDriver
Driver (p. 455) for the Real Time Clock (RTC) 797
io::driver::rtc::RTCHandler
The RTC Interrupt handler 797
io::driver::rtc::RTCTasklet
The Tasklet for the RTC IRQ handler 799
runtime::test::RTTITestCase
Tests RTTI (run-time type identification) in kernel mode 800
task::scheduler::Scheduler
The **Scheduler** (p. 801) keeps all Processes of a computer system 801
task::scheduler::SchedulerHandler 809
memory::Selector
Encapsulates a **Selector** (p. 811) 811
task::lock::Semaphore
Class allowing passive waiting for a resource 816
lib::Semaphore
Represents a **Semaphore** (p. 819) in user space 819
task::semaphore::test::SemaphoreTestCase 820
io::driver::interrupt::SlavePICController 821
io::driver::speaker::SpeakerDevice
PC Speaker device 823
io::driver::speaker::SpeakerDriver
Driver (p. 455) for the Speaker 823
task::lock::SpinLock
Mutex for a critical area using busy waiting 824
task::spinlock::test::SpinLockTestCase 827
task::spinlock::test::SpinLockTestThread 827

boot::SplashScreen
Class for writing a cool logo of the FHDW OS 828
tool::test::StringTestCase
Test case for string functions 829
ipc::SysCallHandler 830
ipc::test::SysCallTestCase
Tests the communication with the kernel via system calls 834
lib::System
The class **System** (p. 835) encapsulates many functions necessary for initializing a user application, managing the communication with the os kernel and managing memory 835
lib::System::SystemCallFailed
Thrown if a system call fails 839
memory::SystemSegmentDescriptor
Describes a system segment descriptor 840
task::tasklet::Tasklet
A **Tasklet** (p. 842) is a piece of code that is run at IRQL_DISPATCH before falling down to IRQL_PASSIVE 842
task::tasklet::TaskletManager
The **TaskletManager** (p. 844) manages all Tasklets in the system 844
task::TaskStateSegment
TSS structure described by the intel documentation 846
runtime::TCB
The **TCB** (p. 847) (Thread Control Block) 847
tool::Terminal 849
test::TestCase
Abstract class for the individual test cases 850
io::driver::test::TestDevice
Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason 851
io::driver::test::TestDevice2
Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason 852
io::driver::test::TestDriver
Class testDriver is an implementation of abstract class **Driver** (p. 455) for testing reason 852
io::driver::test::TestDriver2
Class testDriver2 is an implementation of abstract class **Driver** (p. 455) for testing reason 853

task::pipeandfilter::test::TesterPipe< T >
Pipe (p. 715) implementation for testing propose, compares the incoming values with the expected ones 854
test::TestManager 855
task::tasklet::test::TestTasklet 856
io::driver::graphics::TextChar
Class for the characters to print at the console 857
io::console::TextConsole
The top class for all text output consoles 860
io::driver::graphics::TextDevice
Driver (p. 455) for write signs in a text device and to move the cursor of the text device 862
task::Thread
Thread (p. 865) is an abstract class 865
task::semaphore::test::ThreadASemaphoreTestCase 875
task::scheduler::ThreadQueue
Represents a queue of threads 876
task::ThreadStack 879
io::time::Time
The <<singleton>> Class for all the time operations Saves the ticks that are collected from RTC 881
fosCli::commands::TimeCliCommand 884
fosCli::commands::TimeCliCommandCreator 884
task::scheduler::TimerInterruptHandler 884
cpu::level::TransitionHandler
Handles the the handler for the different levels 886
memory::TSSDescriptor
Describes a TSS descriptor 888
TypeInfo< int16_t > 891
TypeInfo< int32_t > 891
TypeInfo< int64_t > 891
TypeInfo< int8_t > 891
TypeInfo< uint16_t > 892
TypeInfo< uint32_t > 892
TypeInfo< uint64_t > 892
TypeInfo< uint8_t > 893
lib::runtime::UserEnvironment
Class **UserEnvironment** (p. 893) is the implementation of **memory::allocator::Environment** (p. 481) for the user space 893
task::UserModeThread 895

boot::UserSpaceInitThread
This thread is started at the end of the boot process 896
io::vfs::VFSManager
Singleton manager for the virtual file system 897
io::vfs::test::VFSTestCase 901
io::driver::graphics::vga::VgaChar
Represents a single element in the video buffer 901
io::driver::graphics::vga::test::VgaCharTestCase 903
io::driver::graphics::vga::VgaCursor
Implements the **Cursor** (p. 423) interface for the VGA device . . 903
io::driver::graphics::vga::VgaDriver
Driver (p. 455) for the VGA **Device** (p. 436) 905
io::driver::graphics::vga::VgaTextDevice
Console that using the VGA **Device** (p. 436) 906
io::vfs::Volume 909
io::time::WaitingQueue
Priority Queue for the Threads waiting to wake up 914
io::time::WaitingQueueEntry 915

A.3 File Index

A.3.1 File List

Here is a list of all documented files with brief descriptions:

a20.h
Contains functions related to the A20 line 917
AbstractFilter.h . . . . . **??**
AbstractMount.h . . . . . **??**
AbstractScanCodeConverterState.h . . . . . **??**
AddressSpaceRange.h 917
Allocator.h
Provides a memory allocator to manage a heap 932
AllocatorOutOfMemoryTestCase.h 932
docs/src/phase_5/refactoring/presentation/AllocatorTestCase.h . **??**
kernel/memory/allocator/test/AllocatorTestCase.h . . . . . **??**
AllocBlockCommand.h . . . . . **??**
AllocBlockRequest.h . . . . . **??**
ArrayList.h . . . . . **??**

ArrayListIterator.h . . . . . **??**
ArrayListTestCase.h . . . . . **??**
AtaBusDevice.h 933
AtaBusDriver.h 933
AtaBusIrqHandler.h 933
AtaBusWorkerThread.h 934
AtaCommand.h 934
AtaCommandRegister.h
Created on: 04.09.2013 Author: hfi410 934
AtaDevice.h 934
AtaDigitalOutputRegister.h 935
AtaDriveLbaHighest.h 935
AtaDriver.h 935
AtaErrorRegister.h 935
AtaIdentifyCommand.h 935
AtaIdentifyResult.h 935
AtaIoPorts.h 936
AtaReadCommand.h 936
AtaReadWriteTest.h 936
AtaReadWriteTest2.h . . . . . **??**
AtaReadWriteTest3.h 936
AtaStatusRegister.h . . . . . **??**
AtaTasklet.h 936
AtaWriteCommand.h 937
AudioClass.h . . . . . **??**
BitField.h 937
Block.h 937
BlockDevice.h 937
BlockDeviceVolume.h . . . . . **??**
BootConsole.h 937
BootManager.h 938
Bootsector.h . . . . . **??**
BootThread.h 938
bpb.h
Contains the definition of a **BPB** (p. 375) 938
CharLayoutDevice.h . . . . . **??**
CharLayoutDriver.h . . . . . **??**
CharLayoutFilter.h . . . . . **??**
CharLayoutTestCase.h . . . . . **??**
ClassConstants.h . . . . . **??**

CleanupTasklet.h 938
ClearCliCommand.h . . . . . **??**
ClearCliCommandCreator.h . . . . . **??**
ClearScreenCommand.h . . . . . **??**
ClearScreenRequest.h . . . . . **??**
CliCommand.h . . . . . **??**
CliCommandCreator.h . . . . . **??**
ClockThread.h . . . . . **??**
Color.h . . . . . **??**
Command.h 939
CommandDirectory.h . . . . . **??**
CommandDirectoryMaster.h . . . . . **??**
CommandDirectorySlave.h . . . . . **??**
CommandRelay.h 939
Comparator.h 940
apps/library/Console.h . . . . . **??**
loader/osloader/Console.h . . . . . **??**
ConsoleManager.h . . . . . **??**
kernel/io/vfs/fat/Constants.h . . . . . **??**
library/memory/Constants.h . . . . . **??**
Counter.h . . . . . **??**
cpu.h
Contains structures and functions related to the CPU 940
CreateSemaphoreCommand.h 945
CreateSemaphoreRequest.h 945
CurrentTimeCommand.h . . . . . **??**
CurrentTimeRequest.h . . . . . **??**
Cursor.h 946
DefaultConsole.h 946
DefaultExceptionHandler.h . . . . . **??**
DefaultHandler.h . . . . . **??**
defs.h
Contains general definitions 946
Delta.h 946
Descriptor.h . . . . . **??**
Device.h . . . . . **??**
DeviceClass.h . . . . . **??**
DeviceManager.h . . . . . **??**
dir.h
Contains functions for loading and traversing the root directory . 946

Directory.h . . . . . ??
DirectoryEntry.h . . . . . ??
disk.h
Contains functions for loading kernel from disk 947
Driver.h . . . . . ??
DriverFrameworkTestCase.h . . . . . ??
DriverManager.h . . . . . ??
DummyComposite.h . . . . . ??
DummyDirectory.h . . . . . ??
DummyFile.h . . . . . ??
DummyFileStream.h . . . . . ??
DummyFileSystem.h . . . . . ??
DummyMount.h . . . . . ??
E820.h . . . . . ??
EchoCliCommand.h . . . . . ??
EchoCommandCreator.h . . . . . ??
Enums.h . . . . . ??
Environment.h 947
EOnePhaseOneScanCodeConverterState.h . . . . . ??
EOnePhaseTwoScanCodeConverterState.h . . . . . ??
error.h
Contains functions for error handling 948
ExceptionTestCase.h . . . . . ??
ExitThreadCommand.h 949
ExitThreadRequest.h 949
EZeroScanCodeConverterState.h . . . . . ??
FantasticObject.h . . . . . ??
FantasticObjectImpl.h . . . . . ??
FarPointer.h . . . . . ??
fat.h
Contains functions for loading and traversing the **FAT** (p. 497) . 949
FatAccess.h . . . . . ??
FatDirectory.h . . . . . ??
FatFile.h . . . . . ??
FatFileStream.h . . . . . ??
FatFileSystem.h . . . . . ??
FatMount.h . . . . . ??
FatObject.h . . . . . ??
FatRootDirectory.h . . . . . ??
File.h . . . . . ??

FileStream.h . . . . . **??**
FileSystem.h . . . . . **??**
FileSystemNode.h . . . . . **??**
FileSystemNodeVisitor.h . . . . . **??**
FilterThread.h . . . . . **??**
FosCli.h . . . . . **??**
FosCliElement.h . . . . . **??**
FosCliElementVisitor.h . . . . . **??**
FosCliExecuteScannerState.h . . . . . **??**
FosCliReadSpaceDemelitedSignScannerState.h . . . . . **??**
FosCliScanner.h . . . . . **??**
FosCliScannerState.h . . . . . **??**
FosCliScannerStateResult.h . . . . . **??**
FosCliStringElement.h . . . . . **??**
FreeBlockCommand.h . . . . . **??**
FreeBlockRequest.h . . . . . **??**
GDTManager.h . . . . . **??**
getProcessIdCommand.h . . . . . **??**
GetVersionCommand.h 950
GetVersionRequest.h 950
kernel/memory/Heap.h . . . . . **??**
library/memory/Heap.h . . . . . **??**
HelpCliCommand.h . . . . . **??**
HelpCliCommandCreator.h . . . . . **??**
highmem.h
Contains functions for dealing with high memory (>= 1 MiB) . . 950
HighPrioThread.h 952
IdleThread.h . . . . . **??**
InitialScanCodeConverterState.h . . . . . **??**
InterruptHandler.h . . . . . **??**
InterruptHandlerContainer.h . . . . . **??**
InterruptManager.h 952
IOPort.h . . . . . **??**
IPCReceiverThread.h . . . . . **??**
IPCSenderThread.h . . . . . **??**
IPCTestCase.h . . . . . **??**
IRQHandler.h 953
IRQLevel.h . . . . . **??**
Iterator.h . . . . . **??**
KernelEnvironment.h 953

KernelVersionCliCommand.h . . . . . ??
KernelVersionCliCommandCreator.h . . . . . ??
KeyboardTestCase.h . . . . . ??
KeycodeDevice.h . . . . . ??
KeycodeDriver.h . . . . . ??
KeycodeFilter.h . . . . . ??
Keycodes.h . . . . . ??
KeycodeTasklet.h . . . . . ??
KeyValuePair.h . . . . . ??
LASMTest.h 953
LBAddress.h . . . . . ??
LevelManager.h . . . . . ??
LinearAddressSpaceManager.h . . . . . ??
LinearMap.h . . . . . ??
LinearMapTestCase.h . . . . . ??
LinkedList.h . . . . . ??
LinkedListEntry.h . . . . . ??
LinkedListIterator.h . . . . . ??
LinkedListTestCase.h . . . . . ??
List.h . . . . . ??
LoadProgramCommand.h . . . . . ??
LoadProgramRequest.h . . . . . ??
LockTesterThreadDown.h . . . . . ??
LockTesterThreadUp.h . . . . . ??
LowPrioThread.h 953
Map.h . . . . . ??
MaskingHandler.h . . . . . ??
MasterPICController.h . . . . . ??
MD5.h . . . . . ??
memmap.h
Contains functions for retrieval of the system memory map . . . 954
MemoryConstants.h 956
ModifySemaphoreCommand.h 957
ModifySemaphoreRequest.h 957
Mount.h . . . . . ??
nmi.h
Contains functions related to masking/unmasking the NMI 957
NormalThread.h 957
OutputStringCommand.h 958
OutputStringRequest.h 958

Page.h 958
PageContext.h . . . . . **??**
PageDirectory.h . . . . . **??**
PageEntry.h . . . . . **??**
PageFaultExceptionHandler.h 958
PageFlags.h . . . . . **??**
kernel/memory/paging/PageTable.h . . . . . **??**
library/memory/allocator/PageTable.h 959
paging.h
Contains structures and functions related to paging 959
PagingConstants.h . . . . . **??**
PagingTestCase.h . . . . . **??**
PagingUtils.h . . . . . **??**
Parser.h . . . . . **??**
Participant.h 961
PhysicalMemoryManager.h . . . . . **??**
PICController.h . . . . . **??**
PICDevice.h 961
PICDriver.h . . . . . **??**
PICInterruptHandler.h 961
Pipe.h . . . . . **??**
PIT8254ControlWord.h . . . . . **??**
PIT8254Counter.h . . . . . **??**
PIT8254CounterMode.h . . . . . **??**
PIT8254Device.h . . . . . **??**
PIT8254Driver.h . . . . . **??**
Position.h 962
PriorityInheritanceTestCase.h . . . . . **??**
Process.h . . . . . **??**
ProcessIdRequest.h . . . . . **??**
ProcessManager.h 962
ProgramLoader.h . . . . . **??**
Proxy.h 962
PS2Device.h . . . . . **??**
PS2Driver.h . . . . . **??**
PS2KeyboardDevice.h . . . . . **??**
PS2KeyboardDriver.h . . . . . **??**
PS2KeyboardHandler.h . . . . . **??**
Queue.h . . . . . **??**
ReadKeyCommand.h . . . . . **??**

ReadKeyRequest.h . . . . . **??**
Ref.h 965
RefCountedObject.h 965
RefTestCase.h . . . . . **??**
Registry.h 966
RemoteObject.h 966
Request.h 966
RTCDevice.h . . . . . **??**
RTCDriver.h . . . . . **??**
RTCHandler.h . . . . . **??**
RTCTasklet.h . . . . . **??**
RTTITestCase.h . . . . . **??**
kernel/runtime/runtime.h 966
library/runtime/runtime.h 966
Scheduler.h . . . . . **??**
SchedulerHandler.h . . . . . **??**
Selector.h . . . . . **??**
apps/library/Semaphore.h . . . . . **??**
kernel/task/lock/Semaphore.h . . . . . **??**
SemaphoreTestCase.h . . . . . **??**
SlavePICController.h . . . . . **??**
SpeakerDevice.h . . . . . **??**
SpeakerDriver.h . . . . . **??**
SpinLock.h . . . . . **??**
SpinLockTestCase.h . . . . . **??**
SpinLockTestThread.h 967
SplashScreen.h . . . . . **??**
StringComparator.h 967
StringTestCase.h . . . . . **??**
SysCallHandler.h 967
SysCallTestCase.h . . . . . **??**
System.h . . . . . **??**
Tasklet.h . . . . . **??**
TaskletManager.h . . . . . **??**
TaskStateSegment.h . . . . . **??**
Terminal.h . . . . . **??**
TestCase.h . . . . . **??**
TestDevice.h . . . . . **??**
TestDevice2.h . . . . . **??**
TestDriver.h . . . . . **??**

TestDriver2.h . . . . . **??**
TesterPipe.h . . . . . **??**
TestManager.h . . . . . **??**
TestTasklet.h . . . . . **??**
TextChar.h 967
TextConsole.h . . . . . **??**
TextDevice.h . . . . . **??**
Thread.h . . . . . **??**
ThreadASemaphoreTestCase.h . . . . . **??**
ThreadPriority.h 967
ThreadQueue.h 968
ThreadStack.h . . . . . **??**
ThreadState.h . . . . . **??**
Time.h . . . . . **??**
TimeCliCommand.h . . . . . **??**
TimeCliCommandCreator.h . . . . . **??**
TimerInterruptHandler.h . . . . . **??**
TransitionHandler.h . . . . . **??**
types.h 968
UserEnvironment.h . . . . . **??**
UserModeThread.h 969
UserSpaceInitThread.h 970
kernel/tool/utils.h . . . . . **??**
library/tool/utils.h
Contains utility functions 970
Version.h 974
VFSManager.h . . . . . **??**
VFSTestCase.h . . . . . **??**
VgaChar.h . . . . . **??**
VgaCharTestCase.h . . . . . **??**
VgaCursor.h 975
VgaDriver.h . . . . . **??**
VgaTextDevice.h . . . . . **??**
Volume.h . . . . . **??**
WaitingQueue.h . . . . . **??**
WaitingQueueEntry.h . . . . . **??**

A.4 Data Structure Documentation

A.4.1 io::vfs::test::AbstractDummyNode Class Reference

Inherited by **io::vfs::test::DummyLeaf**, and **io::vfs::test::DummyNode**.

Public Member Functions

- **AbstractDummyNode** (const char ∗name)
- const char ∗ **getName** () const
- virtual bool **isFile** () const =0
- void **setParent** (**DummyNode** ∗parent)
- **DummyNode** ∗ **getParent** () const

Private Attributes

- const char ∗ **name**
- **DummyNode** ∗ **parent**

The documentation for this class was generated from the following file:

- DummyComposite.h

A.4.2 task::pipeandfilter::AbstractFilter< T, V > Class Template Reference

Public Member Functions

- virtual void **filter** (T const &in, **Pipe**< V > &paramOutPipe)=0

The documentation for this class was generated from the following file:

- AbstractFilter.h

A.4.3 io::vfs::AbstractMount Class Reference

Interface for all methods, that should be supported by a mount.

Inherits **object::RefCountedObject**.

Inherited by **io::vfs::Mount**.

Public Member Functions

- **AbstractMount** (uint8_t **mountID**)
- virtual uint8_t **getMountID** () const

  Returns the ID, under which this mount is mounted in the vfs.
- virtual **object::Ref**< **Directory** > **getRoot** ()=0

  Returns the root directory of the mount.

Private Attributes

- const uint8_t **mountID**

  The unique identifier of this mount (mount point)

Additional Inherited Members

Detailed Description

Interface for all methods, that should be supported by a mount.

Member Function Documentation

virtual uint8_t io::vfs::AbstractMount::getMountID ( ) const `[inline]`, `[virtual]` Returns the ID, under which this mount is mounted in the vfs.

@

@plus@

@plus -@

@

@

@skip
Returns

The mount ID

References mountID.

virtual object::Ref<Directory> io::vfs::AbstractMount::getRoot ( ) [pure virtual] Returns the root directory of the mount.

@

@plus@

@plus -@

@

@

@skip
Returns

A reference to the root directory

Implemented in **io::vfs::Mount** (p. 657), and **io::vfs::test::DummyMount** (p. 475).

The documentation for this class was generated from the following file:

- AbstractMount.h

A.4.4 io::driver::keycode::scancodestates::AbstractScanCodeConverterState Class Reference

The ScanCodeConverterState is an abstract class for a State pattern to manage the special cases E0 and E1 which are used in the hardware keyboard driver for compatibility reasons.

Inherited by **io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverterState**, **io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverterState**, **io::driver::keycode::scancodestates::EZeroScanCodeConverterState**, and **io::driver::keycode::scancodestates::InitialScanCodeConverterState**.

Public Member Functions

- **AbstractScanCodeConverterState** * **handleScancode** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe)

  handles the

Protected Member Functions

- void **sendKeycode** (uint8_t keycode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const

  sends the scanCode to the Keyboard

Private Member Functions

- int **newScancode** (int scancode, bool **isBreakcode**) const
- virtual **AbstractScanCodeConverterState** * **handle** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const =0

  handles the scancode.
- virtual bool **testIfScancodeIsBreakCode** (int scancode) const =0

  test state specific if a scancode is a breakcode.

Private Attributes

- bool **isBreakcode**

  a flag which shows if the actual read scancode is a breakcode

Detailed Description

The ScanCodeConverterState is an abstract class for a State pattern to manage the special cases E0 and E1 which are used in the hardware keyboard driver for compatibility reasons.

Member Function Documentation

AbstractScanCodeConverterState* io::driver::keycode::scancodestates::AbstractScanCodeConverterState::handleScancode (int *scancode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) handles the

Parameters

scancode	in the context of the state and

@

@plus@

@plus -@

@

@

@skip

Returns

a new ScanCodeConverterState the result will be put in the

void io::driver::keycode::scancodestates::AbstractScanCodeConverterState::sendKeycode (uint8_t *keycode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) const [protected] sends the scanCode to the Keyboard

Parameters

keycode	

int io::driver::keycode::scancodestates::AbstractScanCodeConverterState::newScancode (int *scancode,* bool *isBreakcode*) const [private]

Parameters

scancode	
isBreakcode	

@

@plus@

@plus -@

@

@

@skip

Returns

a new scancode which is not a breakcode

virtual AbstractScanCodeConverterState∗ io::driver::keycode::scancodestates::AbstractScanCodeConverterState::handle (int *scancode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) const [private], [pure virtual] handles the scancode.

the case of a breakcode is resolved before in the abstract state.

Parameters

scancode	scancode
&outPipe	outPipe

@

@plus@

@plus -@

@

@

@skip

Returns

the next state

Implemented in **io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverterState** (p. 484), **io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverterState** (p. 487), **io::driver::keycode::scancodestates::EZeroScanCodeConverterState** (p. 491), and **io::driver::keycode::scancodestates::InitialScanCodeConverterState** (p. 564).

virtual bool io::driver::keycode::scancodestates::AbstractScanCodeConverterState::testIfScancodeIsBreakCode (int *scancode*) const [private], [pure virtual] test state specific if a scancode is a breakcode.

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the passed scancode is a breakcode.

Implemented in **io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverterState** (p. 484), **io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverterState** (p. 486), **io::driver::keycode::scancodestates::EZeroScanCodeConverterState** (p. 490), and **io::driver::keycode::scancodestates::InitialScanCodeConverterState** (p. 564).

The documentation for this class was generated from the following file:

- AbstractScanCodeConverterState.h

A.4.5 memory::lmm::AddressSpaceRange Class Reference

Represents a range of linear address space managed by a **LinearAddressSpaceManager** (p. 597).

Public Types

- enum **Type** : uint32_t { **INVALID** = 0, **FREE**, **ALLOCATED**, **RESERVED** }

  *Enumeration representing all possible types of reservations which can be acquired from the **LinearAddressSpaceManager** (p. 597).*

Public Member Functions

- bool **isValid** () const
- void **setInvalid** ()

  *Marks this **AddressSpaceRange** (p. 269) as INVALID.*
- uint32_t **getStartAddress** () const
- uint32_t **getEndAddress** () const
- uint32_t **getSize** () const
- uint32_t **getSizeInPages** () const
- **Type getType** () const
- char * **getStartPointer** () const
- char * **getEndPointer** () const

- bool **contains** (void const ∗address) const

  Determines if this range contains some address.

- bool **contains** (**AddressSpaceRange** const &range) const

  Determines if this range contains another one.

- uint32_t **getOffset** (void const ∗address) const

  Returns the offset within this range for some address.

- void ∗ **getPointer** (uint32_t offset) const

  Maps an offset relative to the start of this range to an absolute pointer.

- bool **map** (**paging::PageDirectory** &pageDir, **paging::PageFlags** const &flags=**paging::PageFlags**()) const

  Maps the range somewhere into the physical address space.

- bool **map** (**paging::PageDirectory** &pageDir, void ∗physAddr, **paging::-PageFlags** const &flags=**paging::PageFlags**()) const

  Maps the range into a certain region on physical address space.

- bool **map** (**paging::PageDirectory** &pageDir, **AddressSpaceRange** const &otherRange, **paging::PageDirectory** &otherPageDir) const

  Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

- bool **map** (**paging::PageDirectory** &pageDir, **AddressSpaceRange** const &otherRange, **paging::PageDirectory** &otherPageDir, **paging::PageFlags** const &flags) const

  Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

- void **unmap** (**paging::PageDirectory** &pageDir) const

  Unmaps the range from the physical address space.

- **AddressSpaceRange createAlias** (**LinearAddressSpaceManager** &dest, **LinearAddressSpaceManager** &src)

  Creates an alias mapping.

- **AddressSpaceRange createAlias** (**LinearAddressSpaceManager** &dest, **LinearAddressSpaceManager** &src, **paging::PageFlags** flags)

  Creates an alias mapping.

- void ∗ **transform** (void const ∗srcPointer, **AddressSpaceRange** const &targetRange) const

  *Transforms an address pointing into this **AddressSpaceRange** (p. 269) into an address pointing into an alias range.*

Static Public Member Functions

- static **AddressSpaceRange create** (uint32_t **startAddress**, uint32_t **endAddress**)

  Creates an address range.

- static **AddressSpaceRange create** (void const ∗**startAddress**, void const ∗**endAddress**)
- static **AddressSpaceRange create** (void const ∗**startAddress**, uint32_t sizeInBytes)

Data Fields

- enum **memory::lmm::AddressSpaceRange::Type __attribute__**

Private Member Functions

- void **setStartAddress** (uint32_t **startAddress**)

  Sets the start address.

- void **setEndAddress** (uint32_t **endAddress**)

  Sets the end address.

- void **setType** (**Type type**)

  Sets the range type.

- **AddressSpaceRange** ∗ **getPrevious** () const
- void **setPrevious** (**AddressSpaceRange** ∗**previous**)

  *Sets the previous **AddressSpaceRange** (p. 269).*

- **AddressSpaceRange** ∗ **getNext** () const
- void **setNext** (**AddressSpaceRange** ∗**next**)

  *Sets the next **AddressSpaceRange** (p. 269).*

- bool **doMap** (**paging::PageDirectory** &pageDir, **AddressSpaceRange** const &otherRange, **paging::PageDirectory** &otherPageDir, **paging::PageFlags** const ∗flags) const

  Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

Private Attributes

- uint32_t **startAddress**

  The start address of this range.

- uint32_t **endAddress**

  The first address behind this range.

- **Type type**

  The type of this range.

- **AddressSpaceRange** ∗ **previous**

  *Points to the **AddressSpaceRange** (p. 269) ending at our start address.*

- **AddressSpaceRange** ∗ **next**

  *Points to the **AddressSpaceRange** (p. 269) starting at our end address.*

Static Private Attributes

- static uint32_t const **PageOffsetMask** = (1 << memory::PAGE_OFFSET_LEN) - 1

  Mask for the lower PAGE_OFFSET_LEN bits.

Friends

- class **LinearAddressSpaceManager**

Detailed Description

Represents a range of linear address space managed by a **LinearAddressSpaceManager** (p. 597).

Member Enumeration Documentation

enum memory::lmm::AddressSpaceRange::Type : uint32_t Enumeration representing all possible types of reservations which can be acquired from the **LinearAddressSpaceManager** (p. 597).

Enumerator

INVALID The range is invalid.

FREE The range describes a memory range that is unused.

ALLOCATED The range describes a memory range that is in use.

RESERVED The range describes a memory range that is in use by the LinearAddressSpace Manager.

Member Function Documentation

static AddressSpaceRange memory::lmm::AddressSpaceRange::create (uint32_t *startAddress*, uint32_t *endAddress*) [inline], [static]

Creates an address range.

Parameters

start-Address	The start address. It is aligned on the nearest page boundary less than or equal to the address.
endAddress	The end address, i.e. the first address behind the range. If is aligned on the nearest page boundary greater than or equal to the address.

@

@plus@

@plus -@

@

@

@skip

Returns

The **AddressSpaceRange** (p. 269) object. Its type is set to ALLOCATED.

References PageOffsetMask.

bool memory::lmm::AddressSpaceRange::isValid ( ) const [inline]

Returns true if this range is valid.

uint32_t memory::lmm::AddressSpaceRange::getStartAddress ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip

Returns

The start address of this range.

References startAddress.

Referenced by contains().

uint32_t memory::lmm::AddressSpaceRange::getEndAddress ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip

Returns

The first address behind this range.

References endAddress.

Referenced by contains().

uint32_t memory::lmm::AddressSpaceRange::getSize ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

The size of the range in bytes.

References startAddress.

Referenced by getSizeInPages().

uint32_t memory::lmm::AddressSpaceRange::getSizeInPages ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

The size of the range in pages.

References getSize().

Type memory::lmm::AddressSpaceRange::getType ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

The type of this range.

References type.

char* memory::lmm::AddressSpaceRange::getStartPointer ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the memory region described.

References startAddress.

Referenced by transform().

char* memory::lmm::AddressSpaceRange::getEndPointer ( ) const [inline] @

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the following memory region.

References endAddress.

bool memory::lmm::AddressSpaceRange::contains (void const * *address*) const [inline] Determines if this range contains some address.

Parameters

address	The address to check.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this range contains the address passed, else false.

References endAddress.

bool memory::lmm::AddressSpaceRange::contains (AddressSpaceRange const & *range*) const [inline] Determines if this range contains another one.

Parameters

range	The other range to check.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this range contains the range passed, else false.

References endAddress, getEndAddress(), and getStartAddress().

uint32_t memory::lmm::AddressSpaceRange::getOffset (void const * *address*) const [inline] Returns the offset within this range for some address.

Parameters

address	The address.

@

@plus@

@plus -@

@

@

@skip

Returns

The number of bytes from the start of this range to the address passed. Only meaningful if this->contains(address) returns true.

void* memory::lmm::AddressSpaceRange::getPointer (uint32_t *offset*) const [inline] Maps an offset relative to the start of this range to an absolute pointer.

Parameters

offset	The offset to add to the start address of this range.

@

@plus@

@plus -@

@

@

@skip

Returns

The resulting pointer. Only meaningful if the offset is less than the size of this range.

bool memory::lmm::AddressSpaceRange::map (paging::PageDirectory & *pageDir*, paging::PageFlags const & *flags* = paging::PageFlags()) const Maps the range somewhere into the physical address space.

Parameters

pageDir	The page directory to use.
flags	The page flags to use.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

bool memory::lmm::AddressSpaceRange::map (paging::PageDirectory & *pageDir,* void * *physAddr,* paging::PageFlags const & *flags* = paging::PageFlags()) const Maps the range into a certain region on physical address space.

Parameters

pageDir	The page directory to use.
physAddr	The physical start address of the memory to be used for the mapping.
flags	The page flags to use.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

bool memory::lmm::AddressSpaceRange::map (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir*) const [inline] Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPageDir	The page directory the other address space range belongs to.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the whole address range could be mapped, else false.

References doMap().

bool memory::lmm::AddressSpaceRange::map (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir,* paging::PageFlags const & *flags*) const [inline] Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

The page flags can be overridden for the copy (such that e.g. a read-only ring 3 alias can be created for a read/write ring 0 buffer).

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPageDir	The page directory the other address space range belongs to.
flags	The page flags to use for the mapping.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the whole address range could be mapped, else false.

void memory::lmm::AddressSpaceRange::unmap (paging::PageDirectory & *pageDir*) const Unmaps the range from the physical address space.

Parameters

pageDir	The page directory to use.

AddressSpaceRange memory::lmm::AddressSpaceRange::createAlias (LinearAddressSpaceManager & *dest*, LinearAddressSpaceManager & *src*) Creates an alias mapping.

Parameters

dest	The **LinearAddressSpaceManager** (p. 597) to use in order to create the alias mapping.
src	The **LinearAddressSpaceManager** (p. 597) this **AddressSpaceRange** (p. 269) belongs to.

@

@plus@

@plus -@

@

@

@skip

Returns

The new **AddressSpaceRange** (p. 269) living in the memory space of the destination LASM, mapped to the same memory this **AddressSpaceRange** (p. 269) living in the memory space of the source LASM. If the mapping failed, the range returned is of type INVALID.

AddressSpaceRange memory::lmm::AddressSpaceRange::createAlias (LinearAddressSpaceManager & *dest*, LinearAddressSpaceManager & *src*, paging::PageFlags *flags*) Creates an alias mapping.

Parameters

dest	The **LinearAddressSpaceManager** (p. 597) to use in order to create the alias mapping.

src	The **LinearAddressSpaceManager** (p. 597) this **AddressSpaceRange** (p. 269) belongs to.
flags	The flags for the alias mapping.

@

@plus@

@plus -@

@

@

@skip
Returns

The new **AddressSpaceRange** (p. 269) living in the memory space of the destination LASM, mapped to the same memory this **AddressSpaceRange** (p. 269) living in the memory space of the source LASM. If the mapping failed, the range returned is of type INVALID.

void∗ memory::lmm::AddressSpaceRange::transform (void const ∗ *srcPointer,* AddressSpaceRange const & *targetRange*) const [inline]
Transforms an address pointing into this **AddressSpaceRange** (p. 269) into an address pointing into an alias range.

Parameters

srcPointer	The pointer within this **AddressSpaceRange** (p. 269).
target-Range	The alias range.

@

@plus@

@plus -@

@

@

@skip
Returns

The transformed pointer or NULL if srcPointer does not point into this **AddressSpaceRange** (p. 269).

References getStartPointer(), and startAddress.

void memory::lmm::AddressSpaceRange::setStartAddress (uint32_t *startAddress*) [inline], [private] Sets the start address.

Parameters

start-Address	The start address of this range.

References startAddress.

void memory::lmm::AddressSpaceRange::setEndAddress (uint32_t *end-Address*) [inline], [private] Sets the end address.

Parameters

endAddress	The first address behind this range.

void memory::lmm::AddressSpaceRange::setType (Type *type*) [inline], [private] Sets the range type.

Parameters

type	The type of this range.

Referenced by setInvalid().

AddressSpaceRange* memory::lmm::AddressSpaceRange::getPrevious ( ) const [inline], [private] @

@plus@

@plus -@

@

@

@skip

Returns

The previous **AddressSpaceRange** (p. 269).

References previous.

void memory::lmm::AddressSpaceRange::setPrevious (AddressSpaceRange ∗ ***previous*** **)** [inline], [private] Sets the previous **AddressSpaceRange** (p. 269).

Parameters

previous	The previous **AddressSpaceRange** (p. 269).

References previous.

AddressSpaceRange* memory::lmm::AddressSpaceRange::getNext ( ) const [inline], [private] @

@plus@

@plus -@

@

@

@skip

Returns

The next **AddressSpaceRange** (p. 269).

References next.

void memory::lmm::AddressSpaceRange::setNext (AddressSpaceRange * *next*) [inline], [private] Sets the next **AddressSpaceRange** (p. 269).

Parameters

next	The next **AddressSpaceRange** (p. 269).

References next.

bool memory::lmm::AddressSpaceRange::doMap (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir,* paging::PageFlags const * *flags*) const [private] Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

The page flags can be overridden for the copy (such that e.g. a read-only ring 3 alias can be created for a read/write ring 0 buffer).

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPage-Dir	The page directory the other address space range belongs to.
flags	The page flags to use for the mapping. If NULL, the page flags are taken unchanged from the original mapping.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

Referenced by map().

The documentation for this class was generated from the following file:

- **AddressSpaceRange.h**

A.4.6 memory::allocator::Allocator Class Reference

An **Allocator** (p. 286) object manages a heap.

Data Structures

- class **CriticalSection**

  *Helper class for entering/leaving the **Allocator** (p. 286)'s critical section.*

Public Member Functions

- **Allocator** (**Environment** ∗**environ**)

  Constructor.

- ∼**Allocator** ()

Destructor.

- char ∗ **allocate** (uint32_t size)

  Tries to allocate a memory block that is contiguous in the linear address space.

- bool **free** (char ∗address, uint32_t size)

  *Frees a block of memory previously allocated by **allocate()** (p. 288).*

- **Environment** ∗ **getEnvironment** () const

  *Returns the associated **Environment** (p. 481).*

Static Public Member Functions

- static uint32_t **getMaxSmallBlockSize** ()

  Returns the maximum size of a "small" block that is not aligned on a page boundary.

Private Attributes

- **Environment** ∗ **environ**

  The environment to use.

- **PageTable** ∗ **firstPageTable**

  *Points to the first **PageTable** (p. 690).*

Detailed Description

An **Allocator** (p. 286) object manages a heap.

It depends through the **Environment** (p. 481) on some component providing services for the allocation and deallocation of memory with page granularity. By sub-allocation techniques, these pages are divided into smaller parts in order to allow clients to allocate and free small memory blocks. Allocation requests which are larger than **getMaxSmallBlockSize()** (p. 287) are not sub-allocated and aligned on a page boundary.

Constructor & Destructor Documentation

memory::allocator::Allocator::Allocator (Environment ∗ ***environ*** **)** `[inline]`

Constructor.

Parameters

environ	The environment to use.

memory::allocator::Allocator::∼Allocator ( ) Destructor.

Frees all allocated memory.

Member Function Documentation

char∗ memory::allocator::Allocator::allocate (uint32_t *size*) Tries to allocate a memory block that is contiguous in the linear address space.

Parameters

size	The size of the memory block in bytes. It must be greater than zero. If size is larger than **getMaxSmallBlockSize()** (p. 287), the address returned is aligned on a page boundary.

bool memory::allocator::Allocator::free (char ∗ *address,* uint32_t *size*) Frees a block of memory previously allocated by **allocate()** (p. 288).

Parameters

address	The start of the memory block to free.
size	The size of the memory block to free. Use zero if the size is not known but less than **getMaxSmallBlockSize()** (p. 287).

The documentation for this class was generated from the following file:

- **Allocator.h**

A.4.7 memory::allocator::test::AllocatorOutOfMemoryTestCase Class Reference

Tests whether operator new throws an exception in out-of-memory situations.

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

Detailed Description

Tests whether operator new throws an exception in out-of-memory situations.

The documentation for this class was generated from the following file:

- **AllocatorOutOfMemoryTestCase.h**

A.4.8 test::memory::allocator::AllocatorTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- docs/src/phase_5/refactoring/presentation/AllocatorTestCase.h

A.4.9 memory::allocator::test::AllocatorTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Private Member Functions

- void **testBuddyAllocator** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- kernel/memory/allocator/test/AllocatorTestCase.h

A.4.10 api::memory::lmm::AllocBlockRequest Class Reference

Request for achieving an amount of space from the LinearAddressSpaceManager.

Inherits **ipc::Request**.

Public Member Functions

- **AllocBlockRequest** (uint32_t **sizeInPages**)

  Constructor.
- uint32_t **getSizeInPages** () const

  Getter for the required amount of pages.
- uint32_t **getStartAddr** () const

  Getter for startAddr.
- void **setStartAddr** (uint32_t addr)

  Setter for start address.

Private Attributes

- uint32_t const **sizeInPages**

Amount of pages that shall be requested.

- uint32_t **startAddr**

  Represents the return value.

Static Private Attributes

- static const uint32_t **Id** = 2

  Identifier for this request.

Additional Inherited Members

Detailed Description

Request for achieving an amount of space from the LinearAddressSpaceManager.

Constructor & Destructor Documentation

api::memory::lmm::AllocBlockRequest::AllocBlockRequest (uint32_t *sizeInPages*) `[inline]` Constructor.

Parameters

sizeInPages	is the amount of pages required

Member Function Documentation

uint32_t api::memory::lmm::AllocBlockRequest::getSizeInPages ( ) const `[inline]` Getter for the required amount of pages.

@

@plus@

@plus -@

@

@

@skip

Returns

the amount of pages

References sizeInPages.

uint32_t api::memory::lmm::AllocBlockRequest::getStartAddr ( ) const [inline] Getter for startAddr.

@

@plus@

@plus -@

@

@

@skip

Returns

the start address of the acquired memory space

References startAddr.

void api::memory::lmm::AllocBlockRequest::setStartAddr (uint32_t *addr*) [inline] Setter for start address.

Parameters

addr	is the start address of the acquired memory

References startAddr.

Field Documentation

uint32_t api::memory::lmm::AllocBlockRequest::startAddr [private] Represents the return value.

Is the start address of the acquired memory space.

Referenced by getStartAddr(), and setStartAddr().

The documentation for this class was generated from the following file:

- AllocBlockRequest.h

A.4.11 ipc::Array< T > Class Template Reference

Encapsulates an array for IPC.

Public Member Functions

- **Array** (T ∗**array**, uint32_t **numElements**)

  Constructor.

- T ∗& **getArray** ()

  Returns a reference to the pointer to the first element of the array.

- uint32_t **getNumberOfElements** () const

  Returns the number of elements in the array.

Private Attributes

- T ∗ **array**

  A pointer to the first element of the array.

- uint32_t **numElements**

  The number of elements in the array.

Detailed Description

template<typename T>class ipc::Array< T >

Encapsulates an array for IPC.

This is special as an array needs an additional memory mapping.

Constructor & Destructor Documentation

template<typename T> ipc::Array< T >::Array (T ∗ *array,* uint32_t *numElements*) [inline] Constructor.

Parameters

array	A pointer to the first element of the array.
size	The number of elements in the array.

Member Function Documentation

template<typename T> T∗& ipc::Array< T >::getArray ( ) [inline]
Returns a reference to the pointer to the first element of the array.

It's a reference because it has to be modified when transporting the **Array** (p. 292) to the target process.

Referenced by ipc::test::FantasticObjectImpl::incrementElements(), and ipc::test::FantasticObjectImpl::print().

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.12 tool::collection::ArrayList< T > Class Template Reference

implements a list with an automatic growing and shrinking array

Inherits **tool::collection::List**< **T** >.

Public Types

- typedef **ArrayListIterator**< T > **Iterator**

  The underlying iterator types.

- typedef **ConstArrayListIterator**< T > **ConstIterator**

Public Member Functions

- **ArrayList** (**memory::allocator::Allocator** &**allocator**=memory::getAllocator(), uint32_t initialCapacity=**ARRAYLIST_INITIAL_CAPACITY**)

  *Creates a new **ArrayList** (p. 294) with the specified initial capacity.*

- **ArrayList** (**ArrayList** const &other)

  Copy constructor.

- virtual ~**ArrayList** ()

  default destructor

- virtual **memory::allocator::Allocator** & **getAllocator** () const

  Returns the underlying allocator.

- virtual void **add** (T const &element)

  Adds an element to the end of the list.

- virtual bool **remove** (T const &element)

  Removes the first element from the list that is equal to the passed one.

- virtual bool **isEmpty** () const

  Returns true if there are no elements in this list.

- virtual uint32_t **getSize** () const

  Returns the number of elements in this list.

- virtual **ArrayListIterator**< T > ∗ **getIterator** ()

  Returns an iterator for this list.

- virtual **ConstArrayListIterator**
  < T > ∗ **getIterator** () const
- virtual T & **operator[ ]** (const uint32_t index)

  Returns the element at the specified index.

- virtual T const & **operator[ ]** (const uint32_t index) const

Private Member Functions

- **ArrayList**< T > & **operator=** (**ArrayList**< T > const &other)=delete
- void **resize** (const uint32_t newSize)

  resizes the content array to the given size - copies all currently hold elements

- void **unlink** (uint32_t position)

  Removes the first element, that points to the same address, as the parameter.

- T ∗ **getElement** (uint32_t position) const

  Returns a pointer to a (potential) element at some position.

Private Attributes

- **memory::allocator::Allocator** & **allocator**

  The allocator to use.

- uint32_t **capacity**

  the current capacity of the array

- uint32_t **currentSize**

  the count of elements in the list

- char * **content**

  the array, which holds the elements

Static Private Attributes

- static uint32_t const **AlignmentBits**

  The alignment requirements as an exponent of two.

- static uint32_t const **Alignment** = 1 << **AlignmentBits**

  The alignment requirements as a divisor.

- static uint32_t const **Remainder** = sizeof(T) & (**Alignment** - 1)

  The remainder of sizeof(T) when dividing through Alignment.

- static uint32_t const **ElemSize**

  The size to use for the elements.

- static const uint32_t **ARRAYLIST_INITIAL_CAPACITY** = 8

  default initial capacity

Friends

- class **ArrayListIterator**< **T** >

Detailed Description

template<class T>class tool::collection::ArrayList< T >

implements a list with an automatic growing and shrinking array

Constructor & Destructor Documentation

template<class T > tool::collection::ArrayList< T >::ArrayList (memory::allocator::Allocator & *allocator* = `memory::getAllocator()`**, uint32_t *initialCapacity* = ARRAYLIST_INITIAL_CAPACITY)** Creates a new **ArrayList** (p. 294) with the specified initial capacity.

The capacity will grow automatically, if the list gets bigger. If no capacity is given, the default initial capacity (10) is used. If you know, that you will store a lot of content in this list, you should begin your list with a larger initial capacity.

Parameters

allocator	The allocator to use.
initial-Capacity	

References tool::collection::ArrayList< T >::allocator, tool::collection::ArrayList< T >::capacity, tool::collection::ArrayList< T >::content, and tool::collection::-ArrayList< T >::ElemSize.

template<class T > tool::collection::ArrayList< T >::ArrayList (ArrayList< T > const & *other*) Copy constructor.

Parameters

other	The list to copy.

References tool::collection::ArrayList< T >::allocator, tool::collection::ArrayList< T >::capacity, tool::collection::ArrayList< T >::content, tool::collection::ArrayList< T >::currentSize, tool::collection::ArrayList< T >::ElemSize, and tool::collection-::ArrayList< T >::getElement().

Member Function Documentation

template<class T> void tool::collection::ArrayList< T >::add (T const & *element*) `[virtual]` Adds an element to the end of the list.

Parameters

element	the element to add

Implements **tool::collection::List< T >** (p. 629).

Referenced by io::driver::DriverManager::add().

template<class T> bool tool::collection::ArrayList< T >::remove (T const & *element*) `[virtual]` Removes the first element from the list that is equal to the passed one.

This requires operator==(T,T) to be defined.

Parameters

element	the element to remove

@

@plus@

@plus -@

@

@

@skip

Returns

True if the element has been found and removed, else false.

Implements **tool::collection::List**< **T** > (p. 629).

Referenced by io::driver::DriverManager::remove().

template<**class T** > **bool tool::collection::ArrayList**< **T** >**::isEmpty ( ) const** [inline], [virtual] Returns true if there are no elements in this list.

@

@plus@

@plus -@

@

@

@skip

Returns

true if the list is empty, otherwise false

Implements **tool::collection::List**< **T** > (p. 630).

Reimplemented in **tool::collection::LinearMap**< **Key, Value, KeyComp, ValueComp** > (p. 610), **tool::collection::LinearMap**< **int, Device** ∗ > (p. 610), **tool::collection::LinearMap**< **uint32_t, task::Process** ∗ > (p. 610), and **tool::collection::LinearMap**< **uint32_t, void** ∗ > (p. 610).

Referenced by tool::collection::LinearMap< uint32_t, void ∗ >::isEmpty().

template<class T > uint32_t tool::collection::ArrayList< T >::getSize ( ) const [inline], [virtual] Returns the number of elements in this list.

@

@plus@

@plus -@

@

@

@skip
Returns

the number of elements in this list

Implements **tool::collection::List< T >** (p. 631).

Reimplemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 610), **tool::collection::LinearMap< int, Device * >** (p. 610), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 610), and **tool::collection::LinearMap< uint32_t, void * >** (p. 610).

Referenced by tool::collection::LinearMap< uint32_t, void * >::getSize().

template<class T > ArrayList< T >::Iterator * tool::collection::ArrayList< T >::getIterator ( ) [inline], [virtual] Returns an iterator for this list.

@

@plus@

@plus -@

@

@

@skip
Returns

an iterator

Implements **tool::collection::List< T >** (p. 632).

Reimplemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 611), **tool::collection::LinearMap< int, Device * >** (p. 611),

tool::collection::LinearMap< uint32_t, task::Process * > (p. 611), and **tool::collection::LinearMap< uint32_t, void * >** (p. 611).

Referenced by tool::collection::LinearMap< uint32_t, void * >::getIterator().

template<class T > T & tool::collection::ArrayList< T >::operator[] (const uint32_t *index*) [inline], [virtual] Returns the element at the specified index.

Parameters

index	index of element to return

Implements **tool::collection::List< T >** (p. 632).

References fatalError().

template<class T > void tool::collection::ArrayList< T >::resize (const uint32_t *newSize*) [private] resizes the content array to the given size - copies all currently hold elements

Parameters

newSize	the new size of the internal array

template<class T > void tool::collection::ArrayList< T >::unlink (uint32_t *position*) [private] Removes the first element, that points to the same address, as the parameter.

Parameters

element	

References next.

template<class T> T* tool::collection::ArrayList< T >::getElement (uint32_t *position*) const [inline], [private] Returns a pointer to a (potential) element at some position.

Note that the position must be valid, i.e. it needs to be smaller than the capacity of the **ArrayList** (p. 294).

Parameters

position	The position of the element.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the position where this element resides (or will reside).

Referenced by tool::collection::ArrayList< T >::ArrayList().

Field Documentation

template<class T> uint32_t const tool::collection::ArrayList< T >::AlignmentBits [static], [private] Initial value:

```
=
                    sizeof(T) == 1 ? 0 :
                    sizeof(T) == 2 ? 1 :
                    sizeof(T) <= 4 ? 2 : 3
```

The alignment requirements as an exponent of two.

template<class T> uint32_t const tool::collection::ArrayList< T >::ElemSize [static], [private] Initial value:

```
= sizeof(T)
                                                        + (Remainder == 0 ? 0 :
      Alignment - Remainder)
```

The size to use for the elements.

Referenced by tool::collection::ArrayList< T >::ArrayList(), and tool::collection::ArrayList< KeyValuePair< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > >::getElement().

The documentation for this class was generated from the following file:

- ArrayList.h

A.4.13 tool::collection::ArrayListIterator< T > Class Template Reference

an iterator for ArrayLists

Inherits **tool::collection::Iterator**< **T** >.

Public Member Functions

- **ArrayListIterator** (**ArrayList**< T > &**list**)

  creates a new iterator with the given list

- virtual ~**ArrayListIterator** ()

  default destructor

- virtual bool **moveNext** ()

  moves the iterator to the next element

- virtual T & **current** ()

  *returns the current element **moveNext()** (p. 303) has to be called before this function is used the first time.*

- virtual T **remove** ()

  removes the element this iterator points to.

- virtual void **reset** ()

  resets the pointer, as if he was just initialized

Private Attributes

- uint32_t **position**

  the current position

- **ArrayList**< T > & **list**

  the list, we get the data from

Detailed Description

template<**class T**>**class tool::collection::ArrayListIterator**< **T** >

an iterator for ArrayLists

Constructor & Destructor Documentation

template<class T> tool::collection::ArrayListIterator< T >::ArrayListIterator (ArrayList< T > & *list*) creates a new iterator with the given list

Parameters

list	the list, which will be run through

Member Function Documentation

template<class T> virtual bool tool::collection::ArrayListIterator< T >::moveNext ( ) [virtual] moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip

Returns

true, if there is a next element, otherwise false

Implements **tool::collection::Iterator< T >** (p. 580).

Referenced by tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::getKeyValuePair().

template<class T> virtual T& tool::collection::ArrayListIterator< T >::current ( ) [virtual] returns the current element **moveNext()** (p. 303) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implements **tool::collection::Iterator**< **T** > (p. 581).

Referenced by tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::getKeyValuePair().

template<**class T**> **virtual T tool::collection::ArrayListIterator**< **T** >**::remove ( )** [virtual] removes the element this iterator points to.

@

@plus@

@plus -@

@

@

@skip
Returns

The element removed.

Implements **tool::collection::Iterator**< **T** > (p. 581).

The documentation for this class was generated from the following file:

- ArrayListIterator.h

A.4.14 tool::collection::test::ArrayListTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()
- virtual const char * **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- ArrayListTestCase.h

A.4.15 io::driver::block::ata::AtaBusDevice Class Reference

Class which represent a ata bus.

Inherits **io::driver::Device**.

Public Member Functions

- **AtaBusDevice** (uint16_t ioPortStart, uint16_t ioPortAlternateStatus, uint16_t pic_irq_pin, int ClassID)
- void **reinitializeBus** ()

  Do a Software-Reset of the Bus and enables Interrupts.
- void **addCommand** (**commands::AtaCommand** ∗command)

  Adds a command to the execution queue.
- **commands::AtaCommand** ∗ **getCurrentCommand** ()

  Returns the currently active command.
- **AtaIoPorts** & **getPorts** ()

  Returns the associated ports.

Private Member Functions

- void **waitForBsyClear** ()

  Busy waiting until the BSY bit is cleared and RDY bit is set.
- void **waitForBsySet** ()

  Busy waiting until the BSY bit is set.
- void **exitWorkerThread** ()

  Terminates the worker thread.
- bool **runNextCommand** ()

Starts the next AtaCommand and waits for it to complete.

- **commands::AtaCommand** ∗ **dequeueNextCommand** ()

  Removes the first command in the command queue and returns it.

- void **resetCurrentCommand** ()

  Sets currentCommand to NULL.

Private Attributes

- **AtaTasklet** ∗ **tasklet**

  *Tasklet notifying **runNextCommand()** (p. 307) when a command has been completed.*

- **tool::collection::Queue**
  < **commands::AtaCommand** ∗ > **commandQueue**

  The command queue.

- **task::lock::Semaphore queueSemaphore**

  Counts the number of commands in the command queue.

- **task::lock::Semaphore execSemaphore**

  *Serializes **runNextCommand()** (p. 307) calls.*

- **commands::AtaCommand** ∗ **currentCommand**

  Points to the command being currently pending. May be NULL.

- **AtaBusIrqHandler** ∗ **irqHandler**

  IRQ handler notifying the tasklet when a command has been completed.

- uint16_t **irqPicPin**

  The PIC pin associated with our ATA bus device.

- **AtaIoPorts ports**

  The ports associated with our ATA bus device.

- **task::lock::SpinLock mutex**

  Protects the command queue and currentCommand member.

- **task::lock::Semaphore workerExitSemaphore**

  The semaphore used for terminating the worker thread.

- **AtaBusWorkerThread** ∗ **worker**

  The worker thread.

- bool **exiting**

  Set to true before terminating the worker thread.

Friends

- class **AtaBusWorkerThread**

Detailed Description

Class which represent a ata bus.

Normaly only two objects exists for primary and secondary bus

Member Function Documentation

void io::driver::block::ata::AtaBusDevice::addCommand (commands::AtaCommand * *command*) Adds a command to the execution queue.

The command will be executed as soon as all commands added before have been executed. Requires IRQL <= DISPATCH.

Parameters

command	The command to add.

commands::AtaCommand* io::driver::block::ata::AtaBusDevice::getCurrentCommand ( ) Returns the currently active command.

Called by **AtaBusIrqHandler** (p. 309) at device IRQL.

bool io::driver::block::ata::AtaBusDevice::runNextCommand ( ) [private] Starts the next AtaCommand and waits for it to complete.

Called by the **AtaBusWorkerThread** (p. 311). After completion, the command's semaphore is up()ed to notify the command's client. Requires IRQL == PASSIVE.
@

@plus@

@plus -@

@

@

@skip
Returns

True if the worker thread should continue, false if it should exit.

commands::AtaCommand* io::driver::block::ata::AtaBusDevice::dequeueNextCommand ( ) [private] Removes the first command in the command queue and returns it.

Additionally, the command is stored in currentCommand. Requires that no other command is currently being executed (currentCommand needs to be NULL). Called by **runNextCommand()** (p. 307). Requires IRQL == PASSIVE. @

@plus@

@plus -@

@

@

@skip
Returns

The command dequeued.

void io::driver::block::ata::AtaBusDevice::resetCurrentCommand ( ) [private] Sets currentCommand to NULL.

Called by **runNextCommand()** (p. 307) just after a command has been completed. Requires IRQL <= device IRQL.

The documentation for this class was generated from the following file:

- **AtaBusDevice.h**

A.4.16 io::driver::block::ata::AtaBusDriver Class Reference

Driver (p. 455) which creates the AtaBus.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **AtaBusDevice** ∗ **primaryAtaBusDevice**

Additional Inherited Members

Detailed Description

Driver (p. 455) which creates the AtaBus.

The documentation for this class was generated from the following file:

- **AtaBusDriver.h**

A.4.17 io::driver::block::ata::AtaBusIrqHandler Class Reference

Interrupt handler for Interrupt which are thrown from an ATA-Bus.

Inherits **io::driver::interrupt::IRQHandler**.

Public Member Functions

- **AtaBusIrqHandler** (**AtaBusDevice** &device, **AtaTasklet** ∗tasklet)

  Constructor.

- virtual bool **handle** ()

  Handles the interrupt.

Private Attributes

- **AtaBusDevice** & **device**
- **AtaTasklet** ∗ **tasklet**

Detailed Description

Interrupt handler for Interrupt which are thrown from an ATA-Bus.

Constructor & Destructor Documentation

io::driver::block::ata::AtaBusIrqHandler::AtaBusIrqHandler (AtaBusDevice & *device*, AtaTasklet * *tasklet*) Constructor.

Parameters

device	The underlying **AtaBusDevice** (p. 305). Used to delegate the interrupt request to the currently pending command.
tasklet	The tasklet which should be notified when a command has been completed.

Member Function Documentation

virtual bool io::driver::block::ata::AtaBusIrqHandler::handle ( ) `[virtual]`
Handles the interrupt.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the IRQHandler handled the IRQ, else false. In the latter case, the PICInterruptHandler will call the next IRQHandler in the chain (if available).

Implements **io::driver::interrupt::IRQHandler** (p. 579).

The documentation for this class was generated from the following file:

- **AtaBusIrqHandler.h**

A.4.18 io::driver::block::ata::AtaBusWorkerThread Class Reference

ATA bus worker thread which continuously requests the ATA bus device to initiate the processing of pending ATA commands.

Inherits **task::Thread**.

Public Member Functions

- **AtaBusWorkerThread** (**AtaBusDevice** &**device**, **task::lock::Semaphore** &**exitSem**)

  Constructor.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **AtaBusDevice** & **device**

  *The associated **AtaBusDevice** (p. 305).*

- **task::lock::Semaphore** & **exitSem**

  The exit semaphore.

Detailed Description

ATA bus worker thread which continuously requests the ATA bus device to initiate the processing of pending ATA commands.

Constructor & Destructor Documentation

io::driver::block::ata::AtaBusWorkerThread::AtaBusWorkerThread (AtaBusDevice & *device*, task::lock::Semaphore & *exitSem*) Constructor.

Parameters

device	The associated **AtaBusDevice** (p. 305).
exitSem	The exit semaphore. It is up()ed by this thread just before exiting.

Member Function Documentation

virtual void io::driver::block::ata::AtaBusWorkerThread::run ( ) [protected], [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **AtaBusWorkerThread.h**

A.4.19 io::driver::block::ata::commands::AtaCommand Class Reference

Represents a command which can be sent to a hard disk over the ATA bus.

Inherited by **io::driver::block::ata::commands::AtaIdentifyCommand**, **io::driver::block::ata::commands::AtaReadCommand**, and **io::driver::block::ata::commands::AtaWriteCommand**.

Public Member Functions

- **AtaCommand** (bool master, **AtaIoPorts** &ports)
- void **execute** ()

  *Initiates the execution of this **AtaCommand** (p. 312).*
- bool **useMaster** () const

  Simple getter for this->master.
- bool **isCompleted** () const

  Returns true if the command is complete.
- void **wait** ()

Waits until this command has been completely executed.

Protected Member Functions

- virtual bool **receiveInterruptInternal** ()=0

  This method is called if an ata interrupt should be handled.

- virtual void **executeInternal** ()=0

  This method is called, when a command executed the first time.

- void **printStatusRegister** (char const ∗msg)
- void **printStatusRegister** (char const ∗msg, **AtaStatusRegister** reg)
- void **write28BitLbaAddress** (**LBAddress** address, bool master)

  Writes the.

- void **waitForBsyClearAndDrqSet** ()

  Busy waiting until the bsy bit is cleared and the drq bit is set.

- void **waitForReadySignal** ()

  Busy waiting until the rdy bit is set.

- void **initializeRegisterForCommand** (**LBAddress** address, bool master, uint8_t sectorCount)

  Initialise all registers with the given parameters.

Protected Attributes

- **AtaIoPorts** & **ports**

Private Member Functions

- bool **handleInterrupt** ()

  Handles an ATA bus interrupt.

- void **notifyWorker** ()

  *Calls up() on the worker Semaphore, potentially unblocking a worker thread (**AtaBusWorkerThread** (p. 311)) blocking in **waitForCompletion()** (p. 317).*

- void **waitForCompletion** ()

  Waits until this command has been completely executed.

- void **notifyClient** ()

  *Calls up() on the command's Semaphore, potentially unblocking a client thread blocking in **wait()** (p. 315).*

- void **acknowledgeIRQ** ()

  Acknowledges pending IRQ.

Private Attributes

- bool **completed**
- bool **master**
- **task::lock::Semaphore semaphore**
- **task::lock::Semaphore semaphoreWorker**

Friends

- class **ata::AtaBusIrqHandler**
- class **ata::AtaTasklet**
- class **ata::AtaBusDevice**

Detailed Description

Represents a command which can be sent to a hard disk over the ATA bus.

Member Function Documentation

void io::driver::block::ata::commands::AtaCommand::execute ( ) Initiates the execution of this **AtaCommand** (p. 312).

This operation does its work asynchronously, it does not wait for the command to be completed.

bool io::driver::block::ata::commands::AtaCommand::useMaster ( ) const Simple getter for this->master.

@

@plus@

@plus -@

@

@

@skip
Returns

true, if master should used.

void io::driver::block::ata::commands::AtaCommand::wait ( ) [inline]
Waits until this command has been completely executed.

Must be called at IRQL_PASSIVE.

References task::lock::Semaphore::down().

virtual bool io::driver::block::ata::commands::AtaCommand::receiveInterrupt-Internal ( ) [protected], [pure virtual] This method is called if an ata interrupt should be handled.

@

@plus@

@plus -@

@

@

@skip
Returns

true, if this commands reached the ata result phase(the command has the complete result)

Implemented in **io::driver::block::ata::commands::AtaReadCommand** (p. 332), **io::driver::block::ata::commands::AtaWriteCommand** (p. 338), and **io::driver-::block::ata::commands::AtaIdentifyCommand** (p. 327).

virtual void io::driver::block::ata::commands::AtaCommand::execute-Internal ( ) [protected], [pure virtual] This method is called, when a command executed the first time.

the ata command phase.

Implemented in **io::driver::block::ata::commands::AtaReadCommand** (p. 332), **io::driver::block::ata::commands::AtaWriteCommand** (p. 338), and **io::driver::block::ata::commands::AtaIdentifyCommand** (p. 327).

void io::driver::block::ata::commands::AtaCommand::write28BitLbaAddress (LBAddress *address,* bool *master*) [protected] Writes the.

Parameters

address	**LBAddress** (p. 591) to the correspond registers. Sets also the bit
master	for master or slave.
master	true for master, false for slave

void io::driver::block::ata::commands::AtaCommand::waitForBsyClearAndDrqSet ( ) [protected] Busy waiting until the bsy bit is cleared and the drq bit is set.

If error flag is set a fatalError is thrown.

void io::driver::block::ata::commands::AtaCommand::initializeRegisterForCommand (LBAddress *address,* bool *master,* uint8_t *sectorCount*) [protected] Initialise all registers with the given parameters.

Parameters

address,@param	master,
sectorCount	

bool io::driver::block::ata::commands::AtaCommand::handleInterrupt ( ) [private] Handles an ATA bus interrupt.

Called at device IRQL. @

@plus@

@plus -@

@

@

@skip

Returns

True if this interrupt is the last one this command needs to be finished.

void io::driver::block::ata::commands::AtaCommand::notifyWorker ( ) [inline], [private] Calls up() on the worker Semaphore, potentially unblocking a worker thread (**AtaBusWorkerThread** (p. 311)) blocking in **waitForCompletion()** (p. 317).

Called by **AtaTasklet** (p. 334) at IRQL_DISPATCH.

References task::lock::Semaphore::up().

void io::driver::block::ata::commands::AtaCommand::waitForCompletion ( ) [inline], [private] Waits until this command has been completely executed.

Called by **AtaBusDevice::runNextCommand()** (p. 307) in the context of **AtaBusWorkerThread** (p. 311) at IRQL_PASSIVE.

References task::lock::Semaphore::down().

The documentation for this class was generated from the following file:

- **AtaCommand.h**

A.4.20 io::driver::block::ata::AtaCommandRegister Struct Reference

Public Types

- enum **AtaCommandEnum** { **read** = 0x08 << 2, **write** = 0x0c << 2, **identify** = 0x3b << 2 }
- enum **EccGeneration** { **generatedByAtaController** = 0x0 <<1, **generatedByCpu** = 0x1 << 1 }
- enum **RetryBehavior** { **noRetry** = 0x0, **controllerDoRetry** = 0x1 }

Data Fields

- uint8_t **value**

Member Enumeration Documentation

enum io::driver::block::ata::AtaCommandRegister::AtaCommandEnum

enum io::driver::block::ata::AtaCommandRegister::EccGeneration

enum io::driver::block::ata::AtaCommandRegister::RetryBehavior

Field Documentation

uint8_t io::driver::block::ata::AtaCommandRegister::value The documentation for this struct was generated from the following file:

- **AtaCommandRegister.h**

A.4.21 io::driver::block::ata::AtaDevice Class Reference

A storage block device which is connected to an ATA-Bus.

Inherits **io::driver::block::BlockDevice**.

Public Member Functions

- **AtaDevice** (bool paramMaster, **AtaBusDevice** ∗paramAtaBusDevice, int ClassID)
- virtual bool **isReadOnly** ()

  Indicates if this device is a read only device.
- virtual **Block** ∗ **read** (**LBAddress** address)

  Reads the block at the.
- virtual void **write** (**Block** ∗block, **LBAddress** address)

  Writes the.
- virtual **LBAddress getMinAddress** ()
- virtual **LBAddress getMaxAddress** ()
- virtual **Block** ∗ **reads** (**LBAddress** address, uint16_t numberOfBlocksToRead)

Reads the blocks at the.

- virtual void **writes** (**Block** ∗blocks, **LBAddress** address, uint16_t numberOfBlocksToWrite)

  Writes the.

- virtual uint16_t **maximalNumberOfBlocksToReadOrWriteAtOnce** ()

Private Member Functions

- void **testForRange** (const **LBAddress** &address)

Private Attributes

- bool **master**
- **AtaBusDevice** ∗ **ataBusDevice**
- **LBAddress maxAddress**

Static Private Attributes

- static const int **MAXIMAL_NUMBER_OF_BLOCKS_TO_READ_OR_WRITE_AT_ONCE** = 256

Detailed Description

A storage block device which is connected to an ATA-Bus.

Member Function Documentation

virtual bool io::driver::block::ata::AtaDevice::isReadOnly ( ) [virtual]

Indicates if this device is a read only device.

@

@plus@

@plus -@

@

@

@skip
Returns

true -> read-only false -> write and readable

Implements **io::driver::block::BlockDevice** (p. 352).

virtual Block* io::driver::block::ata::AtaDevice::read (LBAddress *address*) [virtual] Reads the block at the.

Parameters

address	and returns it.

@

@plus@

@plus -@

@

@

@skip
Returns

the block which are stored at the

Parameters

address	

Implements **io::driver::block::BlockDevice** (p. 352).

virtual void io::driver::block::ata::AtaDevice::write (Block * *block,* LBAddress *address*) [virtual] Writes the.

Parameters

block	to the device at the given
address	

Implements **io::driver::block::BlockDevice** (p. 353).

virtual LBAddress io::driver::block::ata::AtaDevice::getMinAddress ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

Returns

the minimal **LBAddress** (p. 591) of the address range of this device

Implements **io::driver::block::BlockDevice** (p. 354).

virtual LBAddress io::driver::block::ata::AtaDevice::getMaxAddress ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

Returns

the maximal **LBAddress** (p. 591) of the address range of this device

Implements **io::driver::block::BlockDevice** (p. 354).

virtual Block* io::driver::block::ata::AtaDevice::reads (LBAddress *address,* uint16_t *numberOfBlocksToRead*) [virtual] Reads the blocks at the.

Parameters

address	and following and returns them.

@

@plus@

@plus -@

@

@

@skip

Returns

the block which are stored at the

Parameters

address	

Implements **io::driver::block::BlockDevice** (p. 353).

virtual void io::driver::block::ata::AtaDevice::writes (Block * *blocks,* LBAddress *address,* uint16_t *numberOfBlocksToWrite*) `[virtual]` Writes the.

Parameters

blocks	to the device at the given
address	and the following

Implements **io::driver::block::BlockDevice** (p. 353).

virtual uint16_t io::driver::block::ata::AtaDevice::maximalNumberOfBlocksToReadOrWriteAtOnce ( ) `[virtual]` @

@plus@

@plus -@

@

@

@skip

Returns

the maximal number of blocks which can be read or written by the **Device** (p. 436) at once.

Implements **io::driver::block::BlockDevice** (p. 354).

The documentation for this class was generated from the following file:

- **AtaDevice.h**

A.4.22 io::driver::block::ata::AtaDigitalOutputRegister Struct Reference

Public Types

- enum **OUTPUTREGISTERBITS** { **IrqDisabled** = 0x1 << 1, **SoftwareReset** = 0x1 << 2 }

Public Member Functions

- void **setSoftwareReset** ()
- void **clearSoftwareReset** ()
- void **setIrqEnabled** ()
- void **clearIrqEnabled** ()

Data Fields

- uint8_t **value**

The documentation for this struct was generated from the following file:

- **AtaDigitalOutputRegister.h**

A.4.23 io::driver::block::ata::AtaDriveLbaHighest Struct Reference

Public Types

- enum **Drive** { **master** = 0x0 << 4, **slave** = 0x1 << 4 }

Public Member Functions

- void **setLbaHighestAddress** (uint8_t highestAddress)
- void **setMaster** ()
- void **setSlave** ()

Data Fields

- uint8_t **value**

The documentation for this struct was generated from the following file:

- **AtaDriveLbaHighest.h**

A.4.24 io::driver::block::ata::AtaDriver Class Reference

Driver (p. 455) which creates the AtaDevices.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **AtaDevice** ∗ **masterAtaDevicePrimaryBus**
- **AtaDevice** ∗ **slaveAtaDevicePrimaryBus**

Additional Inherited Members

Detailed Description

Driver (p. 455) which creates the AtaDevices.

The documentation for this class was generated from the following file:

- **AtaDriver.h**

A.4.25 io::driver::block::ata::AtaErrorRegister Struct Reference

Public Types

- enum **ErrorBits** {
  NDM = 0x1, **NT0** = 0x1 << 1, **ABT** = 0x1 << 2, **MCR** = 0x1 << 3,
  NID = 0x1 << 4, **MC** = 0x1 << 5, **UNC** = 0x1 << 6, **BBK** = 0x1 << 7 }

Public Member Functions

- **AtaErrorRegister** (uint8_t paramValue)
- bool **hasBadSector** ()
- bool **hasDataError** ()
- bool **hasMediaChanged** ()
- bool **hasIdMarkNotFound** ()
- bool **hasMediaChangedRequested** ()
- bool **hasCommandAborted** ()
- bool **hasTrack0NotFound** ()
- bool **hasNoDataMarkFound** ()

Data Fields

- uint8_t **value**

The documentation for this struct was generated from the following file:

- **AtaErrorRegister.h**

A.4.26 io::driver::block::ata::commands::AtaIdentify-Command Class Reference

Command for executing the identify command.

Inherits **io::driver::block::ata::commands::AtaCommand**.

Public Member Functions

- **AtaIdentifyCommand** (bool master, **AtaIoPorts** &ports)
- **AtaIdentifyResult** & **getResult** ()

Protected Member Functions

- virtual void **executeInternal** ()

  This method is called, when a command executed the first time.
- virtual bool **receiveInterruptInternal** ()

  This method is called if an ata interrupt should be handled.

Private Member Functions

- void **checkIfDeviceExists** ()

Private Attributes

- **AtaIdentifyResult result**

Additional Inherited Members

Detailed Description

Command for executing the identify command.

Member Function Documentation

AtaIdentifyResult& io::driver::block::ata::commands::AtaIdentifyCommand-::getResult ( ) @

@plus@

@plus -@

@

@

@skip

Returns

Returns the result of this command.

virtual void io::driver::block::ata::commands::AtaIdentifyCommand::executeInternal ( ) [protected], [virtual] This method is called, when a command executed the first time.

the ata command phase.

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

virtual bool io::driver::block::ata::commands::AtaIdentifyCommand::receiveInterruptInternal ( ) [protected], [virtual] This method is called if an ata interrupt should be handled.

@

@plus@

@plus -@

@

@

@skip
Returns

true, if this commands reached the ata result phase(the command has the complete result)

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

The documentation for this class was generated from the following file:

- **AtaIdentifyCommand.h**

A.4.27 io::driver::block::ata::AtaIdentifyResult Class Reference

Result of an AtaIdentifyCommand.

Public Member Functions

- uint32_t **getLbaCount** ()

Maximum available lbaddresses.

- bool **supportsLba** ()

  Indicates if LBA-Mode is supported by the device.

Data Fields

- uint16_t **values** [256]

Private Types

- enum **identifyWord49** { **SupportDma** = 0x01 << 8, **SupportLba** = 0x01 << 9 }

Detailed Description

Result of an AtaIdentifyCommand.

Member Function Documentation

bool io::driver::block::ata::AtaIdentifyResult::supportsLba ( ) Indicates if LBA-Mode is supported by the device.

@

@plus@

@plus -@

@

@

@skip
Returns

true if LBA-Mode is supported

The documentation for this class was generated from the following file:

- **AtaIdentifyResult.h**

A.4.28 io::driver::block::ata::AtaIoPorts Class Reference

The io ports of an **AtaBusDevice** (p. 305).

Public Member Functions

- **AtaIoPorts** (uint16_t ioPortStart, uint16_t ioPortAlternateStatus)
- **IOPort getAlterStatusRegister** () const
- **IOPort getCommandStatusRegister** () const
- **IOPort getDataRegister** () const
- **IOPort getDriveAddressRegister** () const
- **IOPort getDriveLbaHighestRegister** () const
- **IOPort getFaultRegister** () const
- **IOPort getLbaHighRegister** () const
- **IOPort getLbaLowRegister** () const
- **IOPort getLbaMidRegister** () const
- **IOPort getSectorCountRegister** () const

Private Attributes

- **IOPort dataRegister**
- **IOPort faultRegister**
- **IOPort sectorCountRegister**
- **IOPort lbaLowRegister**
- **IOPort lbaMidRegister**
- **IOPort lbaHighRegister**
- **IOPort driveLbaHighestRegister**
- **IOPort commandStatusRegister**
- **IOPort alterStatusRegister**
- **IOPort driveAddressRegister**

Detailed Description

The io ports of an **AtaBusDevice** (p. 305).

The documentation for this class was generated from the following file:

- **AtaIoPorts.h**

A.4.29 io::driver::block::ata::commands::AtaReadCommand Class Reference

Command which execute a read sectors on the ata bus.

Inherits **io::driver::block::ata::commands::AtaCommand**.

Public Member Functions

- **AtaReadCommand** (bool master, **AtaIoPorts** &ports, **LBAddress** address, uint16_t numberOfSectorsToRead)

  Initializing the command with all important settings.
- **Block** ∗ **getResult** ()

Protected Member Functions

- virtual void **executeInternal** ()

  This method is called, when a command executed the first time.
- virtual bool **receiveInterruptInternal** ()

  This method is called if an ata interrupt should be handled.

Private Attributes

- uint16_t **numberOfSectorsToRead**
- uint16_t **readSectors**
- **Block** ∗ **result**
- **LBAddress address**

Additional Inherited Members

Detailed Description

Command which execute a read sectors on the ata bus.

Constructor & Destructor Documentation

io::driver::block::ata::commands::AtaReadCommand::AtaReadCommand (bool *master,* AtaIoPorts & *ports,* LBAddress *address,* uint16_t *numberOfSectorsToRead*) Initializing the command with all important settings.

Parameters

master	true if command should affect the master.
ports	the **AtaIoPorts** (p. 329) object of the ata bus on which this command should executed
address	the address of the start sector where the read command should start reading
numberOfSectorsToRead	the count of sectors which should be read.

Member Function Documentation

Block* io::driver::block::ata::commands::AtaReadCommand::getResult ( ) @

@plus@

@plus -@

@

@

@skip

Returns

the result of this command

virtual void io::driver::block::ata::commands::AtaReadCommand::executeInternal ( ) [protected], [virtual] This method is called, when a command executed the first time.

the ata command phase.

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

virtual bool io::driver::block::ata::commands::AtaReadCommand::receiveInterruptInternal ( ) [protected], [virtual] This method is called if an ata interrupt should be handled.

@

@plus@

@plus -@

@

@

@skip
Returns

true, if this commands reached the ata result phase(the command has the complete result)

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

The documentation for this class was generated from the following file:

- **AtaReadCommand.h**

A.4.30 io::driver::block::ata::test::AtaReadWriteTest Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char * **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- **AtaReadWriteTest.h**

A.4.31 io::driver::block::ata::test::AtaReadWriteTest2 Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- AtaReadWriteTest2.h

A.4.32 io::driver::block::ata::test::AtaReadWriteTest3 Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- **AtaReadWriteTest3.h**

A.4.33 io::driver::block::ata::AtaStatusRegister Struct Reference

Public Types

- enum **STATUSBIT** {
  BSY = 0x1 << 7, **RDY** = 0x1 << 6, **DF** = 0x1 << 5, **SRV** = 0x1 <<4,
  DRQ = 0x1 << 3, **ERR** = 0x1 <<0 }

Public Member Functions

- **AtaStatusRegister** (uint8_t paramValue)
- bool **isBsy** ()
- bool **isRdy** ()
- bool **isDrq** ()
- bool **isErr** ()

Data Fields

- uint8_t **value**

The documentation for this struct was generated from the following file:

- AtaStatusRegister.h

A.4.34 io::driver::block::ata::AtaTasklet Class Reference

Tasklet for the ATA Bus **Device** (p. 436).

Inherits **task::tasklet::Tasklet**.

Public Member Functions

- **AtaTasklet** (**AtaBusDevice** &**device**, cpu::level::IRQLevel deviceIRQL)

  Creates the tasklet.
- virtual task::tasklet::TaskletState **getState** () const

  Returns the tasklet state.
- virtual void **work** ()

The actual work function per tasklet.

- void **setTaskletHasWork** ()

  *Called by **AtaBusIrqHandler** (p. 309) to request the execution of the tasklet.*

- void **setKillTasklet** ()

  *Called by **AtaBusDevice** (p. 305) to request that the tasklet terminates.*

Private Member Functions

- void **setState** (task::tasklet::TaskletState newState)

  Changes the tasklet's state.

Private Attributes

- **AtaBusDevice** & **device**

  *The associated **AtaBusDevice** (p. 305).*

- **task::lock::SpinLock mutex**

  Protects the state of the tasklet.

- task::tasklet::TaskletState **state**

  The state of the tasklet.

Detailed Description

Tasklet for the ATA Bus **Device** (p. 436).

Tasklet continue Current Command or starts next Command

Constructor & Destructor Documentation

io::driver::block::ata::AtaTasklet::AtaTasklet (AtaBusDevice & *device,* cpu::level::IRQLevel *deviceIRQL*) Creates the tasklet.

Parameters

device	The associated **AtaBusDevice** (p. 305). Every time the tasklet runs, the worker thread is notified that the currently pending command of this device has been completed.

Member Function Documentation

virtual task::tasklet::TaskletState io::driver::block::ata::AtaTasklet::getState ( ) const [virtual] Returns the tasklet state.

Only if a Tasklet returns HASWORK it will have its **work()** (p. 334) method invoked.

Implements **task::tasklet::Tasklet** (p. 843).

void io::driver::block::ata::AtaTasklet::setTaskletHasWork ( ) [inline] Called by **AtaBusIrqHandler** (p. 309) to request the execution of the tasklet.

Can be called at device IRQL.

References setState().

void io::driver::block::ata::AtaTasklet::setKillTasklet ( ) [inline] Called by **AtaBusDevice** (p. 305) to request that the tasklet terminates.

Can be called at device IRQL (but is called at IRQL == PASSIVE).

References setState().

void io::driver::block::ata::AtaTasklet::setState (task::tasklet::TaskletState *newState*) [inline], [private] Changes the tasklet's state.

Can be called at device IRQL.

Parameters

newState	The new state of the tasklet.

References mutex, and state.

Referenced by setKillTasklet(), and setTaskletHasWork().

The documentation for this class was generated from the following file:

- **AtaTasklet.h**

A.4.35 io::driver::block::ata::commands::AtaWriteCommand Class Reference

Command which execute a write sectors command on the ata bus.

Inherits **io::driver::block::ata::commands::AtaCommand**.

Public Member Functions

- **AtaWriteCommand** (bool master, **AtaIoPorts** &ports, **LBAddress** address, uint16_t numberOfSectorsToWrite, **Block** ∗data)

  Initializing the command with all important settings.

Protected Member Functions

- virtual void **executeInternal** ()

  This method is called, when a command executed the first time.

- virtual bool **receiveInterruptInternal** ()

  This method is called if an ata interrupt should be handled.

Private Member Functions

- void **writeSector** ()

Private Attributes

- uint16_t **numberOfSectorsToWrite**
- uint16_t **writtenSectors**
- **Block** ∗ **data**
- **LBAddress address**

Additional Inherited Members

Detailed Description

Command which execute a write sectors command on the ata bus.

Constructor & Destructor Documentation

io::driver::block::ata::commands::AtaWriteCommand::AtaWriteCommand (bool *master,* AtaIoPorts & *ports,* LBAddress *address,* uint16_t *numberOfSectorsToWrite,* Block * *data*) Initializing the command with all important settings.

Parameters

master	true if command should affect the master.
ports	the **AtaIoPorts** (p. 329) object of the ata bus on which this command should executed
address	the address of the start sector where the write command should start writing
numberOfSectorsToWrite	the count of sectors which should be written.
data	the data which should be written to the hard drive. The length of this array should be same as the numberOfSectorsToWrite parameter

Member Function Documentation

virtual void io::driver::block::ata::commands::AtaWriteCommand::executeInternal ( ) [protected], [virtual] This method is called, when a command executed the first time.

the ata command phase.

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

virtual bool io::driver::block::ata::commands::AtaWriteCommand::receiveInterruptInternal ( ) [protected], [virtual] This method is called if an ata interrupt should be handled.

@

@plus@

@plus -@

@

@

@skip

Returns

true, if this commands reached the ata result phase(the command has the complete result)

Implements **io::driver::block::ata::commands::AtaCommand** (p. 315).

The documentation for this class was generated from the following file:

- **AtaWriteCommand.h**

A.4.36 ipc::Attribute< T > Class Template Reference

Encapsulates a normal attribute in a **Request** (p. 775).

Public Types

- typedef T const & **ResultType**

  *The result type of **get()** (p. 339).*

Public Member Functions

- **Attribute** ()

  Default constructor.

- **Attribute** (T const &value)

  Constructor.

- **ResultType get** ()

  Returns the underlying value.

- void **set** (T const &value)

  Sets the underlying value.

- void **map** (**Request** &)

  Unused.

Private Attributes

- T **value**

Detailed Description

template<typename T>class ipc::Attribute< T >

Encapsulates a normal attribute in a **Request** (p. 775).

Only attributes of this type can be used for return values.

Constructor & Destructor Documentation

template<typename T> ipc::Attribute< T >::Attribute ( ) [inline]
Default constructor.

Requires T to have an accessible default constructor.

template<typename T> ipc::Attribute< T >::Attribute (T const & *value*) [inline] Constructor.

Parameters

value	The underlying value of this attribute.

Member Function Documentation

template<typename T> void ipc::Attribute< T >::set (T const & *value*) [inline] Sets the underlying value.

Requires T to have an accessible copy assignment operator.

Parameters

value	The new underlying value.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.37 ipc::Attribute< Array< T > > Class Template Reference

Encapsulates an array attribute in a **Request** (p. 775).

Public Types

- typedef **Array**< T > & **ResultType**

  *The result type of **get()** (p. 341).*

Public Member Functions

- **Attribute** (**Array**< T > const &value)

  Constructor.

- **ResultType get** ()

  Returns the underlying value.

- void **map** (**Request** &request)

  Maps the underlying array into the target process's address space.

Private Attributes

- **Array**< T > **value**

Detailed Description

template<typename T>class ipc::Attribute< Array< T > >

Encapsulates an array attribute in a **Request** (p. 775).

Constructor & Destructor Documentation

template<typename T > ipc::Attribute< Array< T > >::Attribute (Array< T > const & *value*) [inline] Constructor.

Parameters

value	The underlying value of this attribute.

Member Function Documentation

template<typename T > void ipc::Attribute< Array< T > >::map (Request & *request*) [inline] Maps the underlying array into the target process's address space.

In order to prevent the target process from reading too much data from the source process, allocate your array at a page boundary. Otherwise, data may leak to the target process unintentionally.

Parameters

request	The originating IPC **Request** (p. 775).

References ipc::Request::addMapRequest().

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.38 ipc::Attribute< Array< T const > > Class Template Reference

Encapsulates an array-of-const attribute in a **Request** (p. 775).

Public Types

- typedef **Array**< T const > & **ResultType**

  *The result type of **get()** (p. 342).*

Public Member Functions

- **Attribute** (**Array**< T const > const &value)

  Constructor.

- **ResultType get** ()

  Returns the underlying value.

- void **map** (**Request** &request)

  Maps the underlying array into the target process's address space.

Private Attributes

- **Array**< T const > **value**

Detailed Description

template<typename T>class ipc::Attribute< Array< T const > >

Encapsulates an array-of-const attribute in a **Request** (p. 775).

Constructor & Destructor Documentation

template<typename T > ipc::Attribute< Array< T const > >::Attribute (Array< T const > const & *value*) [inline] Constructor.

Parameters

value	The underlying value of this attribute.

Member Function Documentation

template<typename T > void ipc::Attribute< Array< T const > >::map (Request & *request*) [inline] Maps the underlying array into the target process's address space.

In order to prevent the target process from reading too much data from the source process, allocate your array at a page boundary. Otherwise, data may leak to the target process unintentionally.

Parameters

request	The originating IPC **Request** (p. 775).

References ipc::Request::addMapRequest().

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.39 ipc::Attribute< T & > Class Template Reference

Encapsulates a reference attribute in a **Request** (p. 775).

Public Types

- typedef T & **ResultType**

  *The result type of **get()** (p. 344).*

Public Member Functions

- **Attribute** (T &value)

  Constructor.

- **ResultType get** ()

  Returns the underlying value.

- void **map** (**Request** &)

  Unused.

Private Attributes

- T & **value**

Detailed Description

template<typename T>class ipc::Attribute< T & >

Encapsulates a reference attribute in a **Request** (p. 775).

Constructor & Destructor Documentation

template<typename T > ipc::Attribute< T & >::Attribute (T & *value*) `[inline]` Constructor.

Parameters

value	The underlying value of this attribute.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.40 ipc::Attribute< T const > Class Template Reference

Encapsulates a const attribute in a **Request** (p. 775).

Public Types

- typedef T const & **ResultType**

  *The result type of **get()** (p. 345).*

Public Member Functions

- **Attribute** (T const &value)

  Constructor.

- **ResultType get** () const

  Returns the underlying value.

- void **map** (**Request** &)

  Unused.

Private Attributes

- T const **value**

Detailed Description

template<typename T>class ipc::Attribute< T const >

Encapsulates a const attribute in a **Request** (p. 775).

Constructor & Destructor Documentation

template<typename T > ipc::Attribute< T const >::Attribute (T const & *value*) `[inline]` Constructor.

Parameters

value	The underlying value of this attribute.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.41 ipc::Attribute< T const & > Class Template Reference

Encapsulates a reference-to-const attribute in a **Request** (p. 775).

Public Types

- typedef T const & **ResultType**

  *The result type of **get()** (p. 346).*

Public Member Functions

- **Attribute** (T const &value)

  Constructor.
- **ResultType get** () const

  Returns the underlying value.
- void **map** (**Request** &)

  Unused.

Private Attributes

- T const & **value**

Detailed Description

template<typename T>class ipc::Attribute< T const & >

Encapsulates a reference-to-const attribute in a **Request** (p. 775).

Constructor & Destructor Documentation

template<typename T > ipc::Attribute< T const & >::Attribute (T const & *value*) [inline] Constructor.

Parameters

value	The underlying value of this attribute.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.42 io::driver::classes::AudioClass Class Reference

Top Class for all Audio **Device** (p. 436) like PC Speaker.

Inherits **io::driver::Device**.

Inherited by **io::driver::speaker::SpeakerDevice**.

Public Member Functions

- virtual void **on** ()=0

  Turns the device on.

- virtual void **off** ()=0

  Turns the device off.

- virtual void **setFrequency** (int freq)=0

  Set the frequency of the sound.

Protected Member Functions

- **AudioClass** (int deviceClassID)

Detailed Description

Top Class for all Audio **Device** (p. 436) like PC Speaker.

The documentation for this class was generated from the following file:

- AudioClass.h

A.4.43 tool::BitField< Base, Index, BitPosition, Length > Class Template Reference

Template for simple reading/writing bit fields from arrays.

Static Public Member Functions

- static Base **get** (Base const *field)

  Returns the value of the bit field.

- static void **set** (Base *field, Base value)

  Sets the value of the bit field.

Static Private Attributes

- static Base const **Mask** = (1 << Length) - 1

  The bit mask used for accessing the bit field.

Detailed Description

template<typename Base, uint32_t Index, uint32_t BitPosition, uint32_t Length>class tool::BitField< Base, Index, BitPosition, Length >

Template for simple reading/writing bit fields from arrays.

Template Parameters

Base	The underlying object type, e.g. uint32_t.
Index	The index within the array of objects containing the bit fields. Use 0 if you want to access a bit field within a single value.
BitPosition	The position of the bit field. 0 means the bit field starts at the lowest significant bit.
Length	The length of the bit field in bits.

Member Function Documentation

template<typename Base , uint32_t Index, uint32_t BitPosition, uint32_t Length> static Base tool::BitField< Base, Index, BitPosition, Length >::get (Base const * *field*) [inline], [static] Returns the value of the bit field.

Parameters

field	A pointer to the object containing the bit field.

@

@plus@

@plus -@

@

@

@skip

Returns

The value.

References tool::BitField< Base, Index, BitPosition, Length >::Mask.

Referenced by memory::CodeOrDataSegmentDescriptor::getBase(), memory::Descriptor::getDPL(), memory::Selector::getIndex(), memory::CodeOrDataSegmentDescriptor::getLimit(), memory::GateDescriptor::getOffset(), memory::GateDescriptor::getParamCount(), memory::Selector::getRPL(), memory::GateDescriptor::getSelector(), memory::SystemSegmentDescriptor::getSystemSegmentType(), memory::Descriptor::getType(), io::driver::pstwo::PS2Device::hasDualChannel(), memory::CodeOrDataSegmentDescriptor::isAccessed(), memory::CodeOrDataSegmentDescriptor::isCodeConforming(), memory::CodeOrDataSegmentDescriptor::isCodeExecuteRead(), memory::CodeOrDataSegmentDescriptor::isDataExpandDown(), memory::CodeOrDataSegmentDescriptor::isDataReadWrite(), memory::Selector::isLDT(), memory::CodeOrDataSegmentDescriptor::isLimit4G(), memory::Descriptor::isPresent(), and memory::CodeOrDataSegmentDescriptor::isSeg32().

template<typename Base , uint32_t Index, uint32_t BitPosition, uint32_t Length> static void tool::BitField< Base, Index, BitPosition, Length >::set (Base * *field*, Base *value*) [inline], [static] Sets the value of the bit field.

Parameters

field	A pointer to the object containing the bit field.
value	The new value.

References tool::BitField< Base, Index, BitPosition, Length >::Mask.

Referenced by io::driver::pstwo::PS2Device::disableFirstPS2Interrupt(), io::driver::pstwo::PS2Device::disableSecondPS2Interrupt(), io::driver::pstwo::PS2Device::enableFirstPS2Interrupt(), io::driver::pstwo::PS2Device::enableFirstPS2PortTranslation(), io::driver::pstwo::PS2Device::enableSecondPS2Interrupt(), memory::CodeOrDataSegmentDescriptor::setAccessed(), memory::CodeOrDataSegmentDescriptor::setBase(), memory::CodeOrDataSegmentDescriptor::setCodeConforming(), memory::CodeOrDataSegmentDescriptor::setCodeExecuteRead(), memory::CodeOrDataSegmentDescriptor::setDataExpandDown(), memory::CodeOrDataSegmentDescriptor::setDataReadWrite(), memory::Descriptor::setDPL(), memory::Selector::setIndex(), memory::Selector::setLDT(), memory::CodeOrDataSegmentDescriptor::setLimit(), memory::CodeOrDataSegmentDescriptor::setLimit4G(), memory::GateDescriptor::setOffset(), memory::GateDescriptor::setParamCount(), memory::Descriptor::setPresent(), memory::Selector::setRPL(), memory::CodeOrDataSegmentDescriptor::setSeg32(), memory::GateDescriptor::setSelector(), memory::SystemSegmentDescriptor::setSystemSegmentType(), and memory::Descriptor::setType().

The documentation for this class was generated from the following file:

- **BitField.h**

A.4.44 io::driver::block::Block Struct Reference

A struct which represents a 512 byte large block of a block-oriented device.

Data Fields

- char **data** [BLOCK_SIZE]

  The data of this block.

Detailed Description

A struct which represents a 512 byte large block of a block-oriented device.

Field Documentation

char io::driver::block::Block::data[BLOCK_SIZE] The data of this block.

The documentation for this struct was generated from the following file:

- **Block.h**

A.4.45 io::driver::block::BlockDevice Class Reference

Class which represent a block-oriented storage device.

Inherits **io::driver::Device**.

Inherited by **io::driver::block::ata::AtaDevice**.

Public Member Functions

- **BlockDevice** (int ClassID)
- virtual bool **isReadOnly** ()=0

  Indicates if this device is a read only device.
- virtual **Block** ∗ **read** (**LBAddress** address)=0

  Reads the block at the.
- virtual void **write** (**Block** ∗block, **LBAddress** address)=0

  Writes the.
- virtual **Block** ∗ **reads** (**LBAddress** address, uint16_t numberOfBlocksToRead)=0

  Reads the blocks at the.
- virtual void **writes** (**Block** ∗blocks, **LBAddress** address, uint16_t numberOfBlocksToWrite)=0

  Writes the.
- virtual uint16_t **maximalNumberOfBlocksToReadOrWriteAtOnce** ()=0
- virtual **LBAddress getMinAddress** ()=0
- virtual **LBAddress getMaxAddress** ()=0

Detailed Description

Class which represent a block-oriented storage device.

Member Function Documentation

virtual bool io::driver::block::BlockDevice::isReadOnly ( ) [pure virtual]
Indicates if this device is a read only device.

@

@plus@

@plus -@

@

@

@skip
Returns

true -> read-only false -> write and readable

Implemented in **io::driver::block::ata::AtaDevice** (p. 319).

virtual Block* io::driver::block::BlockDevice::read (LBAddress *address*) [pure virtual] Reads the block at the.

Parameters

address	and returns it.

@

@plus@

@plus -@

@

@

@skip
Returns

the block which are stored at the

Parameters

address	

Implemented in **io::driver::block::ata::AtaDevice** (p. 320).

virtual void io::driver::block::BlockDevice::write (Block * *block,* LBAddress *address*) [pure virtual] Writes the.

Parameters

block	to the device at the given
address	

Implemented in **io::driver::block::ata::AtaDevice** (p. 320).

virtual Block* io::driver::block::BlockDevice::reads (LBAddress *address,* uint16_t *numberOfBlocksToRead*) [pure virtual] Reads the blocks at the.

Parameters

address	and following and returns them.

@

@plus@

@plus -@

@

@

@skip

Returns

the block which are stored at the

Parameters

address	

Implemented in **io::driver::block::ata::AtaDevice** (p. 321).

virtual void io::driver::block::BlockDevice::writes (Block * *blocks,* LBAddress *address,* uint16_t *numberOfBlocksToWrite*) [pure virtual] Writes the.

Parameters

blocks	to the device at the given
address	and the following

Implemented in **io::driver::block::ata::AtaDevice** (p. 322).

virtual uint16_t io::driver::block::BlockDevice::maximalNumberOfBlocksToReadOrWriteAtOnce ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the maximal number of blocks which can be read or written by the **Device** (p. 436) at once.

Implemented in **io::driver::block::ata::AtaDevice** (p. 322).

virtual LBAddress io::driver::block::BlockDevice::getMinAddress ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the minimal **LBAddress** (p. 591) of the address range of this device

Implemented in **io::driver::block::ata::AtaDevice** (p. 320).

virtual LBAddress io::driver::block::BlockDevice::getMaxAddress ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the maximal **LBAddress** (p. 591) of the address range of this device

Implemented in **io::driver::block::ata::AtaDevice** (p. 321).

The documentation for this class was generated from the following file:

- **BlockDevice.h**

A.4.46 io::vfs::BlockDeviceVolume Class Reference

Implementation of kernel/io/vfs/Volume for FAT16.

Inherits **io::vfs::Volume**.

Public Member Functions

- **BlockDeviceVolume** (**io::driver::block::BlockDevice** ∗bDevice)
- virtual bool **isReadOnly** ()

  Indicates if this device is a read only device.
- virtual **io::driver::block::Block** ∗ **read** (**io::driver::block::LBAddress** address)

  Reads the block at the.
- virtual void **write** (**io::driver::block::Block** ∗block, **io::driver::block::-LBAddress** address)

  Writes the.
- virtual **io::driver::block::Block** ∗ **reads** (**io::driver::block::LBAddress** address, uint16_t numberOfBlocksToRead)

  Reads the blocks at the.
- virtual void **writes** (**io::driver::block::Block** ∗blocks, **io::driver::block-::LBAddress** address, uint16_t numberOfBlocksToWrite)

  Writes the.

- virtual uint16_t **maximalNumberOfBlocksToReadOrWriteAtOnce** ()
- virtual **io::driver::block::LBAddress getMinAddress** ()
- virtual **io::driver::block::LBAddress getMaxAddress** ()

Private Attributes

- **io::driver::block::BlockDevice** ∗ **bdevice**

Detailed Description

Implementation of kernel/io/vfs/Volume for FAT16.

Member Function Documentation

virtual bool io::vfs::BlockDeviceVolume::isReadOnly ( ) [virtual] Indicates if this device is a read only device.

@

@plus@

@plus -@

@

@

@skip
Returns

true -> read-only false -> write and readable

Implements **io::vfs::Volume** (p. 910).

virtual io::driver::block::Block∗ io::vfs::BlockDeviceVolume::read (io::driver::block::LBAddress *address*) [virtual] Reads the block at the.

Parameters

address	and returns it.

@

@plus@

@plus -@

@

@

@skip

Returns

the block which are stored at the

Parameters

address	

Implements **io::vfs::Volume** (p. 910).

virtual void io::vfs::BlockDeviceVolume::write (io::driver::block::Block * *block,* io::driver::block::LBAddress *address*) [virtual] Writes the.

Parameters

block	to the device at the given
address	

Implements **io::vfs::Volume** (p. 911).

virtual io::driver::block::Block* io::vfs::BlockDeviceVolume::reads (io::driver::block::LBAddress *address,* uint16_t *numberOfBlocksToRead*) [virtual] Reads the blocks at the.

Parameters

address	and following and returns them.

@

@plus@

@plus -@

@

@

@skip
Returns

the block which are stored at the

Parameters

address	

Implements **io::vfs::Volume** (p. 912).

virtual void io::vfs::BlockDeviceVolume::writes (io::driver::block::Block * *blocks,* io::driver::block::LBAddress *address,* uint16_t *numberOfBlocksToWrite*) [virtual] Writes the.

Parameters

blocks	to the device at the given
address	and the following

Implements **io::vfs::Volume** (p. 912).

virtual uint16_t io::vfs::BlockDeviceVolume::maximalNumberOfBlocksToReadOrWriteAtOnce ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the maximal number of blocks which can be read or written by the Device at once.

Implements **io::vfs::Volume** (p. 912).

virtual io::driver::block::LBAddress io::vfs::BlockDeviceVolume::getMinAddress ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the minimal LBAddress of the address range of this device

Implements **io::vfs::Volume** (p. 913).

virtual io::driver::block::LBAddress io::vfs::BlockDeviceVolume::getMaxAddress ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the maximal LBAddress of the address range of this device

Implements **io::vfs::Volume** (p. 913).

The documentation for this class was generated from the following file:

- BlockDeviceVolume.h

A.4.47 boot::BootConsole Class Reference

Class for the output during the boot.

Inherits **io::console::TextConsole**.

Public Member Functions

- **BootConsole** (**memory::paging::PageDirectory** ∗**initDir**, uint32_t linStartOfVideoRAM)

  Constructor.

- void **setTextColor** (io::console::Color color)

  Sets the text color.

- void **setBackgroundColor** (io::console::Color color)

  Sets the text background color.

- virtual void **write** (char s)

  writes one character at the current cursor position

- virtual void **write** (char const ∗s)

  writes a string at the current cursor position

- virtual void **writeLine** (char const ∗s)

  writes a string at the current cursor position and put the cursor on an new line

- virtual void **newLine** ()

  breaks the line at the current cursor position

- virtual void **scrollUp** ()

  scrolls the screen up

- virtual void **clear** ()

  clears the screen and puts the cursor to position 0,0 (top left)

- virtual void **activate** ()

  activates the console not fully implemented yet

- virtual void **deactivate** ()

  not implemented yet

Private Member Functions

- void **readCursorPosition** ()

  Reads the current cursor position from the VGA CRT controller and adjusts videoCur accordingly.

- void **setCursorPosition** ()

  Computes the current cursor position from videoCur and programs the VGA CRT controller accordingly.

Private Attributes

- **io::driver::graphics::vga::VgaChar ∗ videoStart**

  Points to the start of the video buffer.

- **io::driver::graphics::vga::VgaChar ∗ videoEnd**

Points to the first byte behind the video buffer.

- **io::driver::graphics::vga::VgaChar** ∗ **videoCur**

  Points to the current output position within the video buffer.

- **memory::paging::PageDirectory** ∗ **initDir**

  The page directory used to map the video RAM.

- io::console::Color **fgColor**

  The foreground color.

- io::console::Color **bgColor**

  The background color.

Static Private Attributes

- static unsigned int const **PhysicalStartAddressOfVideoRAM** = 0xB8000

  Physical start address of the Video RAM.

- static uint16_t const **CRTAddressPort** = 0x3d4

  CRT address port.

- static uint16_t const **CRTDataPort** = **CRTAddressPort** + 1

  CRT data port.

- static uint16_t const **CRTCursorHigh** = 0xE

  CRT Cursor High.

- static uint16_t const **CRTCursorLow** = **CRTCursorHigh** + 1

  CRT Cursor Low.

- static io::console::Color const **DefBG**

  Default foreground colour.

- static io::console::Color const **DefFG**

  Default background colour.

- static uint32_t const **NumCols** = 80

  Number of columns.

- static uint32_t const **NumLines** = 25

  Number of lines.

Detailed Description

Class for the output during the boot.

Constructor & Destructor Documentation

boot::BootConsole::BootConsole (memory::paging::PageDirectory * *initDir*, uint32_t *linStartOfVideoRAM*) Constructor.

Parameters

initDir	The initial Page Directory
linStartOf-VideoRAM	Startaddress of the VGA RAM before the VGA driver

Member Function Documentation

void boot::BootConsole::setTextColor (io::console::Color *color*) `[inline]`, `[virtual]` Sets the text color.

Parameters

color	The color number to set

Implements **io::console::TextConsole** (p. 861).

References fgColor.

void boot::BootConsole::setBackgroundColor (io::console::Color *color*) `[inline]`, `[virtual]` Sets the text background color.

Parameters

color	The color number to set

Implements **io::console::TextConsole** (p. 861).

References bgColor.

virtual void boot::BootConsole::write (char *s*) `[virtual]` writes one character at the current cursor position

Parameters

s	character to print

Implements **io::console::TextConsole** (p. 861).

virtual void boot::BootConsole::write (char const * *s*) [virtual] writes a string at the current cursor position

Parameters

s	pointer to the first character of the string

Implements **io::console::TextConsole** (p. 861).

virtual void boot::BootConsole::writeLine (char const * *s*) [virtual] writes a string at the current cursor position and put the cursor on an new line

Parameters

s	pointer to the first character of the string

Implements **io::console::TextConsole** (p. 862).

Field Documentation

io::console::Color const boot::BootConsole::DefBG [static], [private]

Initial value:

```
=
                io::console::Color::BLACK
```

Default foreground colour.

io::console::Color const boot::BootConsole::DefFG [static], [private]

Initial value:

```
=
                io::console::Color::GREY
```

Default background colour.

The documentation for this class was generated from the following file:

- **BootConsole.h**

A.4.48 boot::BootManager Class Reference

Controls the boot process.

Public Member Functions

- **BootManager** (uint32_t **imageSize**, uint32_t **kernelSize**, uint32_t **bss-Size**)

  Constructor.

- void **boot** (**memory::E820_entry** const ∗mmEntry)

  Boots the system by initializing all kernel subsystems.

Private Member Functions

- void **boot** ()

  Boots the system by initializing all kernel subsystems.

- void **abortBoot** ()

  Called when an error occurred while booting.

- void **initConsoleManager** ()

  Initializes the console manager.

- void **initTestManager** ()

  Initializes the test manager.

- void **initGDTManager** ()

  Initializes the GDT manager.

- void **initPaging** ()

  Initializes the paging subsystem.

- void **initPhysicalMemoryManager** ()

  Initializes the physical memory manager.

- void **initLevelManager** ()

  Initializes the level manager.

- void **initKernelLASM** ()

  Initializes the kernel linear address space manager.

- void **initKernelHeaps** ()

Initializes the kernel heaps.

- void **initInterruptManager** ()

  Initializes the interrupt manager.

- void **initSysCallHandler** ()

  Initializes the system call handler.

- void **initTaskletManager** ()

  Initializes the tasklet manager.

- void **initIPCRegistry** ()

  Initializes the IPC registry.

- void **initProcessManager** ()

  Initializes the process manager and creates the initial system process.

- void **initDeviceManager** ()

  Initializes the device manager.

- void **initDriverManager** ()

  Initializes the driver manager.

- void **initBootDrivers** ()

  Initializes all boot drivers.

- void **initScheduler** ()

  Initializes the scheduler and creates the idle thread.

Static Private Member Functions

- static **BootConsole** & **getBootConsole** ()

  *Returns a reference to the **BootConsole** (p. 359).*

- static void **cleanupAfterBoot** ()

  *Called by **CleanupTasklet::work()** (p. 383) to clean up some resources only used while booting.*

Private Attributes

- uint32_t **imageSize**

  The size in bytes of the complete kernel image.

- uint32_t **kernelSize**

  The size in bytes of the kernel code and data.

- uint32_t **bssSize**

  The size in bytes of uninitialized data.

- **memory::E820_entry** ∗ **mmEntryFirst**

  Points to the first entry in the BIOS memory chain.

- **memory::E820_entry** ∗ **mmEntryLast**

  Points to the last entry in the BIOS memory chain.

- **memory::paging::PageDirectory** ∗ **initPageDir**

  Initial page directory.

- char ∗ **firstFreeLinAddr**

  The first free linear address behind the kernel code and data.

- char ∗ **firstFreePhysAddr**

  The first free physical address behind the kernel code and data.

Static Private Attributes

- static char **bootConsole** [ ]

  The boot console.

Friends

- class **CleanupTasklet**

Detailed Description

Controls the boot process.

Constructor & Destructor Documentation

boot::BootManager::BootManager (uint32_t *imageSize,* uint32_t *kernelSize,* uint32_t *bssSize*) Constructor.

Parameters

imageSize	The size in bytes of the complete kernel image, including attached applications, but excluding uninitialized data (which is discarded at link time).
kernelSize	The size in bytes of the kernel code and data.
bssSize	The size in bytes of uninitialized data.

Member Function Documentation

void boot::BootManager::boot (memory::E820_entry const ∗ *mmEntry*) Boots the system by initializing all kernel subsystems.

Parameters

mmEntry	BIOS information about the available physical memory.

The documentation for this class was generated from the following file:

- **BootManager.h**

A.4.49 io::vfs::fat::Bootsector Class Reference

Public Member Functions

- uint8_t **getBotSig** () const

  Bootsignature of the Bootsektor.

- uint16_t **getBytesPerSec** () const

  *Defines the Bytes per Sector on the **Volume** (p. 909).*

- uint8_t **getDrvNum** () const

  *The drive Number from the Hard **Disk** (p. 448).*

- uint16_t **getFatSize16** () const

  *The size of the **FAT** (p. 497) on the **Disk** (p. 448).*

- const char ∗ **getFilSysType** () const

  The file System type.

- uint32_t **getHiddSec** () const

  *The number of hidden sectors on the **Disk** (p. 448).*

- const char ∗ **getJmpBoot** () const

The jmpBoot instruction from the disk.

- uint8_t **getMediaDesc** () const

  The media deskriptor from the disk.

- uint8_t **getNumFaTs** () const

  The number of FATs on the disk.

- uint16_t **getNumOfHeads** () const

  The number of heads the disk has.

- const char ∗ **getOemName** () const

  The name of the programm which formatted the disk.

- uint8_t **getReserved1** () const

  Count of reserved bytes.

- uint16_t **getRootEntryCount** () const
- uint16_t **getRsvdSecCnt** () const
- uint8_t **getSecPerClus** () const
- uint16_t **getSecPerTrk** () const
- uint16_t **getTotSec** () const
- uint32_t **getTotSec32** () const
- uint32_t **getVolId** () const
- const char ∗ **getVolLabel** () const

Data Fields

- char **jmpBoot** [3]
- char **OEMName** [8]
- uint16_t **BytesPerSec**
- uint8_t **SecPerClus**
- uint16_t **RsvdSecCnt**
- uint8_t **NumFATs**
- uint16_t **RootEntryCount**
- uint16_t **TotSec**
- uint8_t **MediaDesc**
- uint16_t **FATSize16**

- uint16_t **SecPerTrk**
- uint16_t **NumOfHeads**
- uint32_t **HiddSec**
- uint32_t **TotSec32**
- uint8_t **DrvNum**
- uint8_t **Reserved1**
- uint8_t **BotSig**
- uint32_t **VolId**
- char **VolLabel** [11]
- char **FilSysType** [8]

Member Function Documentation

uint8_t io::vfs::fat::Bootsector::getBotSig ( ) const [inline] Bootsignature of the Bootsektor.

@

@plus@

@plus -@

@

@

@skip
Returns

uint16_t io::vfs::fat::Bootsector::getBytesPerSec ( ) const [inline] Defines the Bytes per Sector on the **Volume** (p. 909).

@

@plus@

@plus -@

@

@

@skip

Returns

uint8_t io::vfs::fat::Bootsector::getDrvNum ( ) const [inline] The drive Number from the Hard **Disk** (p. 448).

@

@plus@

@plus -@

@

@

@skip
Returns

uint16_t io::vfs::fat::Bootsector::getFatSize16 ( ) const [inline] The size of the **FAT** (p. 497) on the **Disk** (p. 448).

@

@plus@

@plus -@

@

@

@skip
Returns

const char* io::vfs::fat::Bootsector::getFilSysType ( ) const [inline] The file System type.

Not reliable!! @

@plus@

@plus -@

@

@

@skip
Returns

uint32_t io::vfs::fat::Bootsector::getHiddSec ( ) const [inline] The number of hidden sectors on the **Disk** (p. 448).

@

@plus@

@plus -@

@

@

@skip
Returns

const char* io::vfs::fat::Bootsector::getJmpBoot ( ) const [inline] The jmpBoot instruction from the disk.

@

@plus@

@plus -@

@

@

@skip
Returns

uint8_t io::vfs::fat::Bootsector::getMediaDesc ( ) const [inline] The media deskriptor from the disk.

@

@plus@

@plus -@

@

@

@skip
Returns

uint8_t io::vfs::fat::Bootsector::getNumFaTs ( ) const [inline] The number of FATs on the disk.

@

@plus@

@plus -@

@

@

@skip
Returns

uint16_t io::vfs::fat::Bootsector::getNumOfHeads ( ) const [inline] The number of heads the disk has.

@

@plus@

@plus -@

@

@

@skip
Returns

const char* io::vfs::fat::Bootsector::getOemName ( ) const [inline]
The name of the programm which formatted the disk.

@

@plus@

@plus -@

@

@

@skip
Returns

uint8_t io::vfs::fat::Bootsector::getReserved1 ( ) const [inline] Count of reserved bytes.

@

@plus@

@plus -@

@

@

@skip
Returns

The documentation for this class was generated from the following file:

- Bootsector.h

A.4.50 boot::BootThread Class Reference

Performs the second part of the boot process in a separate boot thread.

Inherits **task::Thread**.

Public Member Functions

- **BootThread** (char ∗**addrAppArea**, uint32_t **sizeAppArea**)

  Constructor.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Member Functions

- void **initOtherDrivers** ()

  Initializes all other drivers.
- void **initVGAConsole** ()

  Initializes the VGA console.

Private Attributes

- char ∗ **addrAppArea**

  The start address of the application area.
- uint32_t **sizeAppArea**

  The size of the application area in bytes.

Detailed Description

Performs the second part of the boot process in a separate boot thread.

Constructor & Destructor Documentation

boot::BootThread::BootThread (char ∗ *addrAppArea,* uint32_t *sizeAppArea*) Constructor.

Parameters

addrAppArea	The start address of the application area.
sizeAppArea	The size of the application area in bytes.

Member Function Documentation

virtual void boot::BootThread::run ( ) [protected], [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **BootThread.h**

A.4.51 BPB Struct Reference

Describes a **BPB** (p. 375) (BIOS parameter block).

Public Member Functions

- unsigned long **getNumberOfSectors** () const

  Returns the total number of sectors on this disk.

Data Fields

- unsigned short **numBytesPerSector**

  Number of bytes per sector.

- unsigned char **numSectorsPerCluster**

  Number of sectors per cluster.

- unsigned short **numReservedSectors**

  Number of reserved sectors.

- unsigned char **numFATs**

  Number of FATs.

- unsigned short **numRootDirEntries**

  Number of root directory entries.

- unsigned short **numSectors**

  Number of total sectors on disk (16-bit value).

- unsigned char **mediaDesc**

  Media descriptor.

- unsigned short **numSectorsPerFAT**

  *Number of sectors per **FAT** (p. 497).*

- unsigned short **numSectorsPerTrack**

  Number of sectors per track.

- unsigned short **numHeads**

  Number of heads.

- unsigned long **numHiddenSectors**

  Number of hidden sectors (always zero for unpartitioned disks).

- unsigned long **numSectorsLong**

  Number of total sectors on disk (32-bit value).

- unsigned char **physicalDrive**

  Number of physical drive we have booted from.

- unsigned char **reserved**

  Reserved for future use.

- unsigned char **bootSignature**

  Boot signature.

- unsigned char **volumeSignature** [4]

  Volume signature.

- unsigned char **partitionLabel** [11]

  Partition label, filled by blanks and not terminated by NUL.

- unsigned char **fsName** [8]

  Name of file system, filled by blanks and not terminated by NUL.

Detailed Description

Describes a **BPB** (p. 375) (BIOS parameter block).

Member Function Documentation

unsigned long BPB::getNumberOfSectors ( ) const [inline] Returns the total number of sectors on this disk.

@

@plus@

@plus -@

@

@

@skip
Returns

The total number of sectors.

References numSectors, and numSectorsLong.

The documentation for this struct was generated from the following file:

- **bpb.h**

A.4.52 RootDirectory::Entry::Callback Struct Reference

Describes a callback. This is an interface.

Public Member Functions

- virtual bool **run** (**Entry** const &entry, void const ∗**buffer**, unsigned size)=0

  Called when a buffer fills up while loading a file system entry or if the loading process ends.

Detailed Description

Describes a callback. This is an interface.

Member Function Documentation

virtual bool RootDirectory::Entry::Callback::run (Entry const & *entry,* void const * *buffer,* unsigned *size*) `[pure virtual]` Called when a buffer fills up while loading a file system entry or if the loading process ends.

An implementation of this operation is expected to process the buffer's contents appropriately.

Parameters

entry	The entry.
buffer	The buffer to be processed.
size	The number of bytes in the buffer to process.

@

@plus@

@plus -@

@

@

@skip

Returns

True if callback succeeded, false otherwise.

The documentation for this struct was generated from the following file:

- **dir.h**

A.4.53 io::driver::charlayout::CharLayoutDevice Class Reference

Device (p. 436) for a Char Layout.

Inherits **io::driver::Device**.

Public Member Functions

- **CharLayoutDevice** (**task::pipeandfilter::Pipe**< **keycode::Keycode** > &keycodeBuffer)

  Constructor.

- **task::pipeandfilter::Pipe**< char > & **getCharBuffer** ()

  Getter for the Output Buffer.

Private Attributes

- **task::pipeandfilter::Pipe**< char > **charBuffer**

  Output Buffer for the chars.

Detailed Description

Device (p. 436) for a Char Layout.

Constructor & Destructor Documentation

io::driver::charlayout::CharLayoutDevice::CharLayoutDevice (task::pipeandfilte ::Pipe< keycode::Keycode > & *keycodeBuffer*) Constructor.

Parameters

keycode-Buffer	The keycode buffer to read from

Member Function Documentation

task::pipeandfilter::Pipe<char>& io::driver::charlayout::CharLayoutDevice- ::getCharBuffer ( ) [inline] Getter for the Output Buffer.

@

@plus@

@plus -@

@

@

@skip

Returns

Returns the Output Buffer

References charBuffer.

The documentation for this class was generated from the following file:

- CharLayoutDevice.h

A.4.54 io::driver::charlayout::CharLayoutDriver Class Reference

Driver (p. 455) for a Char Layout.

Inherits **io::driver::Driver**.

Public Member Functions

- **CharLayoutDriver** ()

  Constructor.

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **CharLayoutDevice** ∗ **charLayoutDevice**

Additional Inherited Members

Detailed Description

Driver (p. 455) for a Char Layout.

The documentation for this class was generated from the following file:

- CharLayoutDriver.h

A.4.55 io::driver::charlayout::CharLayoutFilter Class Reference

Filter to translate Keycodes to chars.

Inherits **task::pipeandfilter::AbstractFilter**< **keycode::Keycode, char** >.

Public Member Functions

- **CharLayoutFilter** (**keymap_entry** paramKeymap[ ])

  Constructor.

- virtual void **filter** (**keycode::Keycode** const &keycode, **task::pipeandfilter::Pipe**< char > &paramOutPipe)

  The real translation work.

Private Member Functions

- **keymap_entry** ∗ **getKeyMapEntry** (uint8_t keycode) const

Private Attributes

- **keymap_entry** ∗ **keymap**

  language keymap

- bool **shift**

  special keys buffer

- bool **control**
- bool **alt**
- bool **altgr**

Detailed Description

Filter to translate Keycodes to chars.

Constructor & Destructor Documentation

io::driver::charlayout::CharLayoutFilter::CharLayoutFilter (keymap_entry *paramKeymap[ ]*) Constructor.

Parameters

paramKeymap	The translation map to use

Member Function Documentation

virtual void io::driver::charlayout::CharLayoutFilter::filter (keycode::Keycode const & *keycode*, task::pipeandfilter::Pipe< char > & *paramOutPipe*) [virtual] The real translation work.

Parameters

keycode	the input keycode stream
paramOutPipe	the output where the translated chars will land

keymap_entry* io::driver::charlayout::CharLayoutFilter::getKeyMapEntry (uint8_t *keycode*) const [private]

Parameters

keycode	

@

@plus@

@plus -@

@

@

@skip

Returns

the corresponding keymap entry

The documentation for this class was generated from the following file:

- CharLayoutFilter.h

A.4.56 io::driver::charlayout::test::CharLayoutTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- CharLayoutTestCase.h

A.4.57 boot::CleanupTasklet Class Reference

Cleans up some resources which are only used while booting.

Inherits **task::tasklet::Tasklet**.

Public Member Functions

- **CleanupTasklet** ()

  Constructor.
- virtual task::tasklet::TaskletState **getState** () const

  Used for check if cleanup ran and then destroy the tasklet.
- virtual void **work** ()

  The work of the Tasklet.

Private Attributes

- bool **cleanupRun**

  True if already cleaned up, else false.

Detailed Description

Cleans up some resources which are only used while booting.

Member Function Documentation

virtual task::tasklet::TaskletState boot::CleanupTasklet::getState ( ) const [virtual] Used for check if cleanup ran and then destroy the tasklet.

@

@plus@

@plus -@

@

@

@skip
Returns

TOKILL when finished the cleanup

Implements **task::tasklet::Tasklet** (p. 843).

The documentation for this class was generated from the following file:

- **CleanupTasklet.h**

A.4.58 fosCli::commands::ClearCliCommand Class Reference

Inherits **fosCli::parser::CliCommand**.

Public Member Functions

- virtual void **execute** ()
- virtual void **addParameter** (char const ∗parameter)

The documentation for this class was generated from the following file:

- ClearCliCommand.h

A.4.59 fosCli::commands::ClearCliCommandCreator Class Reference

Inherits **fosCli::parser::CliCommandCreator**.

Public Member Functions

- virtual **parser::CliCommand** ∗ **create** ()

The documentation for this class was generated from the following file:

- ClearCliCommandCreator.h

A.4.60 api::io::console::ClearScreenCommand Class Reference

Handles **ClearScreenRequest** (p. 386).

Public Member Functions

- bool **execute** (**ClearScreenRequest** &request, **ipc::Participant** sender) const

Static Public Member Functions

- static bool **handle** (**ipc::Request** &request, **ipc::Participant** sender)

Detailed Description

Handles **ClearScreenRequest** (p. 386).

Member Function Documentation

static bool api::io::console::ClearScreenCommand::handle (ipc::Request & *request,* ipc::Participant *sender*) `[inline]`**,** `[static]`

- Handles a Request by calling **execute()** (p. 386). ∗

  Parameters

request	∗ The Request to handle. ∗

sender	∗ The sender of the Request. ∗

@

@plus@

@plus -@

@

@

@skip
Returns

∗ True if the Request has been successfully handled, else false.

bool api::io::console::ClearScreenCommand::execute (ClearScreenRequest & *request,* ipc::Participant *sender*) const

- Executes the Command. ∗

  Parameters

request	∗ The originating Request. ∗
sender	∗ The sender of the Request. ∗

@

@plus@

@plus -@

@

@

@skip
Returns

∗ True if the Request has been successfully handled, else false.

The documentation for this class was generated from the following file:

- ClearScreenCommand.h

A.4.61 api::io::console::ClearScreenRequest Class Reference

Remove all content from the screen.

Inherits **ipc::Request**.

Static Public Attributes

- static const uint32_t **Id** = 7

  The identifier of this Request.

Additional Inherited Members

Detailed Description

Remove all content from the screen.

The documentation for this class was generated from the following file:

- ClearScreenRequest.h

A.4.62 fosCli::parser::CliCommand Class Reference

Abstract class for commands called through the CLI.

Inherited by **fosCli::commands::ClearCliCommand**, **fosCli::commands::EchoCliCommand**, **fosCli::commands::HelpCliCommand**, **fosCli::commands::KernelVersionCliCommand**, and **fosCli::commands::TimeCliCommand**.

Public Member Functions

- virtual void **execute** ()=0
- virtual void **addParameter** (char const ∗parameter)=0

Detailed Description

Abstract class for commands called through the CLI.

For proper intialization, any instances of sub classes are created by the corresponding **CliCommandCreator** (p. 388).

The documentation for this class was generated from the following file:

- CliCommand.h

A.4.63 fosCli::parser::CliCommandCreator Class Reference

Inherited by **fosCli::commands::ClearCliCommandCreator**, **fosCli::commands::EchoCommandCreator**, **fosCli::commands::HelpCliCommandCreator**, **fosCli::commands::KernelVersionCliCommandCreator**, and **fosCli::commands::TimeCliCommandCreator**.

Public Member Functions

- **CliCommandCreator** (char const ∗label)
- virtual **CliCommand** ∗ **create** ()=0
- char const ∗ **getLabel** ()

Private Attributes

- char const ∗ **cmdLabel**

The documentation for this class was generated from the following file:

- CliCommandCreator.h

A.4.64 ClockThread Class Reference

Inherits **task::Thread**.

Public Member Functions

- **ClockThread** (**task::Process** &parent)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Additional Inherited Members

Member Function Documentation

virtual void ClockThread::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- ClockThread.h

A.4.65 memory::CodeOrDataSegmentDescriptor Class Reference

Describes a code or data segment descriptor.

Inherits **memory::Descriptor**.

Public Member Functions

- uint32_t **getBase** () const

  Returns the segment's base address.

- **CodeOrDataSegmentDescriptor** & **setBase** (uint32_t base)

  Sets the segment's base address.

- uint32_t **getLimit** () const

  Returns the segment's limit.

- **CodeOrDataSegmentDescriptor** & **setLimit** (uint32_t limit)

  Sets the segment's limit.

- bool **isAccessed** () const

  Returns the Accessed flag.

- **CodeOrDataSegmentDescriptor** & **setAccessed** (bool accessed)

  Sets the Accessed flag.

- bool **isDataReadWrite** () const

  Returns the Data Read/Write flag.

- **CodeOrDataSegmentDescriptor** & **setDataReadWrite** (bool readWrite)

  Sets the Data Read/Write flag.

- bool **isDataExpandDown** () const

  Returns the Data Expand Down flag.

- **CodeOrDataSegmentDescriptor** & **setDataExpandDown** (bool readWrite)

  Sets the Data Expand Down flag.

- bool **isCodeExecuteRead** () const

  Returns the Code Execute/Read flag.

- **CodeOrDataSegmentDescriptor** & **setCodeExecuteRead** (bool executeRead)

  Sets the Code Execute/Read flag.

- bool **isCodeConforming** () const

  Returns the Code Conforming flag.

- **CodeOrDataSegmentDescriptor** & **setCodeConforming** (bool conforming)

  Sets the Code Conforming flag.

- bool **isSeg32** () const

  Returns the 32-bit segment (D/B) flag.

- **CodeOrDataSegmentDescriptor** & **setSeg32** (bool seg32)

  Sets the 32-bit segment (D/B) flag.

- bool **isLimit4G** () const

  Returns the 4GiB Limit (G) flag.

- **CodeOrDataSegmentDescriptor** & **setLimit4G** (bool limit4g)

  Sets the 4GiB Limit (G) flag.

Private Types

- typedef **tool::BitField**
  < uint32_t, 0, 0, 16 > **LimitLowField**

  Describes the bits 0..15 of the limit.

- typedef **tool::BitField**
  < uint32_t, 0, 16, 16 > **BaseLowField**

  Describes the bits 0..15 of the base address.

- typedef **tool::BitField**
  < uint32_t, 1, 0, 8 > **BaseHigh1Field**

  Describes the bits 16..23 of the base address.

- typedef **tool::BitField**
  < uint32_t, 1, 8, 1 > **AccessedField**

  Describes the Accessed flag.

- typedef **tool::BitField**
  < uint32_t, 1, 9, 1 > **DataReadWriteField**

  Describes the Data Read/Write flag.

- typedef **tool::BitField**
  < uint32_t, 1, 9, 1 > **CodeExecuteReadField**

  Describes the Code Execute/Read flag.

- typedef **tool::BitField**
  < uint32_t, 1, 10, 1 > **DataExpandDownField**

  Describes the Data Expand Down flag.

- typedef **tool::BitField**
  < uint32_t, 1, 10, 1 > **CodeConformingField**

  Describes the Code Conforming flag.

- typedef **tool::BitField**
  < uint32_t, 1, 16, 4 > **LimitHighField**

  Describes the bits 16..19 of the limit.

- typedef **tool::BitField**
  < uint32_t, 1, 22, 1 > **Seg32Field**

  Describes the 32-bit Segment (D/B) flag.

- typedef **tool::BitField**
  < uint32_t, 1, 23, 1 > **Limit4GField**

  Describes the 4GiB Limit (G) flag.

- typedef **tool::BitField**
  < uint32_t, 1, 24, 8 > **BaseHigh2Field**

  Describes the bits 24..31 of the base address.

Additional Inherited Members

Detailed Description

Describes a code or data segment descriptor.

Member Function Documentation

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setBase (uint32_t *base*) [inline] Sets the segment's base address.

Parameters

base	The new base address.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

uint32_t memory::CodeOrDataSegmentDescriptor::getLimit ( ) const [inline] Returns the segment's limit.

The value returned is always a full 32-bit value, taking the value of the 4GiB Limit flag (G) into account.

References memory::Descriptor::data, tool::BitField< Base, Index, BitPosition, Length >::get(), and isLimit4G().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setLimit (uint32_t *limit*) [inline] Sets the segment's limit.

If the limit is greater than or equal to 2^20, the 4GiB Limit flag (G) is set.

Parameters

limit	The new limit.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, tool::BitField< Base, Index, BitPosition, Length >::set(), and setLimit4G().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setAccessed (bool *accessed*) [inline] Sets the Accessed flag.

Parameters

accessed	The Accessed flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setDataReadWrite (bool *readWrite*) [inline] Sets the Data Read/-Write flag.

Parameters

readWrite	The Data Read/Write flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setDataExpandDown (bool *readWrite*) [inline] Sets the Data Expand Down flag.

Parameters

readWrite	The Data Expand Down flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setCodeExecuteRead (bool *executeRead*) [inline] Sets the Code Execute/Read flag.

Parameters

execute-Read	The Code Execute/Read flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setCodeConforming (bool *conforming*) [inline] Sets the Code Conforming flag.

Parameters

conforming	The Code Conforming flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setSeg32 (bool *seg32*) [inline] Sets the 32-bit segment (D/B) flag.

Parameters

seg32	The 32-bit segment flag. If true, a code segment contains code which natively uses 32-bit addressing, and a data segment used as a stack is addressed by the 32-bit ESP stack pointer. If false, a code segment contains code which natively uses 16-bit addressing, and a data segment used as a stack is addressed by the 16-bit SP stack pointer.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

CodeOrDataSegmentDescriptor& memory::CodeOrDataSegmentDescriptor::setLimit4G (bool *limit4g*) [inline] Sets the 4GiB Limit (G) flag.

Don't call this explicitly, let **setLimit()** (p. 392) determine the correct value for this flag.

Parameters

limit4g	True if the limit is shifted 12 bit to the left, false if no shifting should be done.

@

@plus@

@plus -@

@

@

@skip

Returns

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

Referenced by setLimit().

The documentation for this class was generated from the following file:

- Descriptor.h

A.4.66 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add >

- **Command** (p. **??**) handling a binary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. *

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.67 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_getAnswer > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_getAnswer >

- **Command** (p. **??**) handling a nullary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy-::Id_getAnswer >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.68 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_increment > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_increment >

- **Command** (p. **??**) handling a unary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy-::Id_increment >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.69 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementElements > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementElements >

- **Command** (p. **??**) handling a unary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementElements >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.70 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementInPlace > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementInPlace >

- **Command** (p. **??**) handling a unary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementInPlace >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.71 ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_print > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::CommandBase**.

Public Member Functions

- virtual void **call** (**RequestBase** &request)

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_print >

- **Command** (p. **??**) handling a unary request.

Member Function Documentation

void ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_print >::call (RequestBase & *request*) [inline], [virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implements **ipc::test::FantasticObjectProxy::CommandBase** (p. 403).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.72 ipc::test::FantasticObjectProxy::CommandBase Class Reference

Inherited by **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add >**, **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_getAnswer >**, **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_increment >**, **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementElements >**, **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementInPlace >**, and **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_print >**.

Public Member Functions

- virtual ∼**CommandBase** ()
- virtual void **call** (**RequestBase** &request)=0

Detailed Description

- Base class of all generated commands.

Constructor & Destructor Documentation

virtual ipc::test::FantasticObjectProxy::CommandBase::~CommandBase ( ) [inline], [virtual]

- Destructor.

Member Function Documentation

virtual void ipc::test::FantasticObjectProxy::CommandBase::call (RequestBase & *request*) [pure virtual]

- Dispatches the request to the correct method. ∗

  Parameters

request	∗ The originating request.

Implemented in **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_print >** (p. 402), **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_add >** (p. 397), **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementElements >** (p. 400), **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_incrementInPlace >** (p. 401), **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_increment >** (p. 399), and **ipc::test::FantasticObjectProxy::Command< FantasticObjectProxy::Id_getAnswer >** (p. 398).

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.73 fosCli::parser::CommandDirectory Class Reference

Inherited by **fosCli::parser::CommandDirectoryMaster**, and **fosCli::parser::CommandDirectorySlave**.

Public Member Functions

- **CommandDirectory** (const char ∗dirName)
- bool **addCommand** (**CliCommandCreator** ∗cmd)
- **CliCommandCreator** ∗ **find** (char const ∗label)
- void **addDirectory** (**CommandDirectory** ∗dir)
- const **CommandDirectory** ∗ **getNext** () const

- const **tool::collection::LinkedList** < **CliCommandCreator** * > & **getCommands** () const
- const char * **getName** ()

Private Attributes

- **CommandDirectory** * **next**
- **tool::collection::LinkedList** < **CliCommandCreator** * > **commands**
- const char * **name**

Member Function Documentation

bool fosCli::parser::CommandDirectory::addCommand (CliCommandCreator * ***cmd*** **)**

Parameters

cmd	the Command to add

@

@plus@

@plus -@

@

@

@skip

Returns

true if the Command was added successfully, false if a command with that name already exists.

The documentation for this class was generated from the following file:

- CommandDirectory.h

A.4.74 fosCli::parser::CommandDirectoryMaster Class Reference

Inherits **fosCli::parser::CommandDirectory**.

Static Public Member Functions

- static **CommandDirectoryMaster** ∗ **getInstance** ()

Private Member Functions

- **CommandDirectoryMaster** (const char ∗dirName)

Static Private Attributes

- static **CommandDirectoryMaster** ∗ **theInstance**

Additional Inherited Members

The documentation for this class was generated from the following file:

- CommandDirectoryMaster.h

A.4.75 fosCli::parser::CommandDirectorySlave Class Reference

Inherits **fosCli::parser::CommandDirectory**.

Public Member Functions

- **CommandDirectorySlave** (char ∗dirName)

The documentation for this class was generated from the following file:

- CommandDirectorySlave.h

A.4.76 ipc::CommandRelay Class Reference

A **CommandRelay** (p. 405) is responsible for sending requests from one process to another or to the kernel.

Public Member Functions

- **CommandRelay** (**Participant sender**, **memory::Selector callGate**)

  *Constructor of class **CommandRelay** (p. 405).*

- bool **relay** (**Request** ∗request, uint32_t size, **Participant** receiver)

  Sends passed request to its receiver.

Private Attributes

- **Participant sender**

  The sender (i.e. we).

- **memory::Selector callGate**

  The call gate to use.

Detailed Description

A **CommandRelay** (p. 405) is responsible for sending requests from one process to another or to the kernel.

Constructor & Destructor Documentation

ipc::CommandRelay::CommandRelay (Participant *sender,* memory::-Selector *callGate*) Constructor of class **CommandRelay** (p. 405).

Parameters

toHandle-Command	
toResult-Gate	

Member Function Documentation

bool ipc::CommandRelay::relay (Request ∗ *request,* uint32_t *size,* Participant *receiver*) `[inline]` Sends passed request to its receiver.

Parameters

request	The request to be sent.
size	The size of the **Request** (p. 775) in bytes.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the request has been processed, else false. If successful, the **Request** (p. 775) object may contain the result of the request (if there are any).

References ipc::Participant::getProcessId(), and memory::Selector::getValue().

Referenced by lib::System::syscall().

The documentation for this class was generated from the following file:

- **CommandRelay.h**

A.4.77 tool::collection::Comparator< T > Class Template Reference

Abstracts from operator==().

Static Public Member Functions

- static bool **equals** (T const &lhs, T const &rhs)

  Compares two elements.

Detailed Description

template<typename T>class tool::collection::Comparator< T >

Abstracts from operator==().

This template may be specialized to provide type-specific comparator algorithms, e.g. strcmp() for char const ∗.

Member Function Documentation

template<typename T > static bool tool::collection::Comparator< T >::equals (T const & *lhs*, T const & *rhs*) [inline], [static] Compares two elements.

Parameters

lhs	The first element.
rhs	The second element.

@

@plus@

@plus -@

@

@

@skip
Returns

lhs==rhs

The documentation for this class was generated from the following file:

- **Comparator.h**

A.4.78 tool::collection::Comparator< char const ∗ > Class Template Reference

Specializes **Comparator** (p. 407) for char const ∗, using strcmp() for the comparison.

Static Public Member Functions

- static bool **equals** (char const ∗const &lhs, char const ∗const &rhs)

  Compares two elements.

Detailed Description

template<>class tool::collection::Comparator< char const * >

Specializes **Comparator** (p. 407) for char const *, using strcmp() for the comparison.

Member Function Documentation

static bool tool::collection::Comparator< char const * >::equals (char const *const & *lhs*, char const *const & *rhs*) [inline], [static] Compares two elements.

Parameters

lhs	The first element.
rhs	The second element.

@

@plus@

@plus -@

@

@

@skip
Returns

strcmp(lhs, rhs)

The documentation for this class was generated from the following file:

- **StringComparator.h**

A.4.79 lib::Console Class Reference

Static Public Member Functions

- static void **write** (const char *msg)
- static char **readKey** ()
- static void **clear** ()

The documentation for this class was generated from the following file:

- apps/library/Console.h

A.4.80 io::console::ConsoleManager Class Reference

<<singleton>> Class for switching consoles during boot

Public Member Functions

- **TextConsole** & **getCurrentConsole** () const

  Returns the current active console.

- **TextConsole** ∗ **setCurrentConsole** (**TextConsole** ∗**console**)

  Changes the active console.

Static Public Member Functions

- static **ConsoleManager** & **instance** ()

  The manager instance getter.

Private Member Functions

- **ConsoleManager** ()

  Constructor.

Static Private Member Functions

- static void **init** ()

  Creates the static instance of the manager (FIXME: create a real singleton)

Private Attributes

- **TextConsole** ∗ **console**

  The current active console (VGA Device or direct)

Static Private Attributes

- static **ConsoleManager manager**

  The static attribute for the singleton pattern.

Friends

- class **boot::BootManager**

Detailed Description

<<singleton>> Class for switching consoles during boot

Member Function Documentation

static ConsoleManager& io::console::ConsoleManager::instance ( ) `[inline]`**,** `[static]` The manager instance getter.

@

@plus@

@plus -@

@

@

@skip
Returns

The static instance

References manager.

TextConsole& io::console::ConsoleManager::getCurrentConsole ( ) const `[inline]` Returns the current active console.

@

@plus@

@plus -@

@

@

@skip
Returns

Direct access console before VGA Device initialized. VGA Device afterwards.

References console.

TextConsole∗ io::console::ConsoleManager::setCurrentConsole (TextConsole ∗ *console*) [inline] Changes the active console.

Parameters

console	The console that will be active

@

@plus@

@plus -@

@

@

@skip
Returns

The active console if it could be changed

References io::console::TextConsole::activate(), console, and io::console::TextConsole::deactivate().

The documentation for this class was generated from the following file:

- ConsoleManager.h

A.4.81 tool::collection::ConstArrayListIterator< T > Class Template Reference

an iterator for ArrayLists

Inherits **tool::collection::ConstIterator**< **T** >.

Public Member Functions

- **ConstArrayListIterator** (**ArrayList**< T > const &**list**)

creates a new iterator with the given list

- virtual ~**ConstArrayListIterator** ()

  default destructor

- virtual bool **moveNext** ()

  moves the iterator to the next element

- virtual T const & **current** ()

  *returns the current element **moveNext()** (p. 414) has to be called before this function is used the first time.*

- virtual void **reset** ()

  resets the pointer, as if he was just initialized

Private Attributes

- uint32_t **position**

  the current position

- **ArrayList**< T > const & **list**

  the list, we get the data from

Detailed Description

template<class T>class tool::collection::ConstArrayListIterator< T >

an iterator for ArrayLists

Constructor & Destructor Documentation

template<class T> tool::collection::ConstArrayListIterator< T >::ConstArrayListIterator (ArrayList< T > const & *list*) creates a new iterator with the given list

Parameters

list	the list, which will be run through

Member Function Documentation

template<class T> virtual bool tool::collection::ConstArrayListIterator< T >::moveNext ( ) [virtual] moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip
Returns

true, if there is a next element, otherwise false

Implements **tool::collection::ConstIterator< T >** (p. 415).

template<class T> virtual T const& tool::collection::ConstArrayListIterator< T >::current ( ) [virtual] returns the current element **moveNext()** (p. 414) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implements **tool::collection::ConstIterator< T >** (p. 416).

The documentation for this class was generated from the following file:

- ArrayListIterator.h

A.4.82 tool::collection::ConstIterator< T > Class Template Reference

Basic definition of an iterator over an unmodifiable collection.

Inherited by **tool::collection::ConstArrayListIterator< T >**, and **tool::collection::ConstLinkedListIterator< T >**.

Public Member Functions

- virtual bool **moveNext** ()=0

  moves the iterator to the next element

- virtual T const & **current** ()=0

  *returns the current element **moveNext()*** (p. *415) has to be called before this function is used the first time.*

- virtual void **reset** ()=0

  resets the pointer, as if he was just initialized

Detailed Description

template<class T>class tool::collection::ConstIterator< T >

Basic definition of an iterator over an unmodifiable collection.

Member Function Documentation

template<class T > virtual bool tool::collection::ConstIterator< T >::moveNext ( ) `[pure virtual]` moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip

Returns

true, if there is a next element, otherwise false

Implemented in **tool::collection::ConstLinkedListIterator**< **T** > (p. 418), and **tool::collection::ConstArrayListIterator**< **T** > (p. 414).

template<**class T** > **virtual T const& tool::collection::ConstIterator**< **T** >**::current ( )** [pure virtual] returns the current element **moveNext()** (p. 415) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implemented in **tool::collection::ConstLinkedListIterator**< **T** > (p. 418), and **tool::collection::ConstArrayListIterator**< **T** > (p. 414).

The documentation for this class was generated from the following file:

- Iterator.h

A.4.83 tool::collection::ConstLinkedListIterator< T > Class Template Reference

Implementation of **tool::collection::Iterator** (p. 579).

Inherits **tool::collection::ConstIterator**< **T** >.

Public Member Functions

- **ConstLinkedListIterator** (**LinkedList**< T > const &list)

  Constructor.

- virtual ~**ConstLinkedListIterator** ()

*Destructor for class **LinkedListIterator** (p. 624).*

- virtual bool **moveNext** ()

  moves the iterator to the next element

- virtual T const & **current** ()

  *returns the current element **moveNext()** (p. 418) has to be called before this function is used the first time.*

- virtual void **reset** ()

  resets the pointer, as if he was just initialized

- bool **movePrevious** ()

Private Attributes

- **LinkedListEntry**< T > ∗ **currentPosition**

  Pointer to the current element.

- **LinkedListEntry**< T > ∗const & **firstElement**

  *The first element in the **LinkedList** (p. 613).*

- **LinkedListEntry**< T > ∗const & **lastElement**

  *The last element in the **LinkedList** (p. 613).*

Detailed Description

template<class T>class tool::collection::ConstLinkedListIterator< T >

Implementation of **tool::collection::Iterator** (p. 579).

Provides the capability to iterate over a **LinkedList** (p. 613) via the **Iterator** (p. 579) interface.

Constructor & Destructor Documentation

template<class T > tool::collection::ConstLinkedListIterator< T >::ConstLinkedListIterator (LinkedList< T > const & *list*) Constructor.

Parameters

list	The underlying list.

Member Function Documentation

template<class T > virtual bool tool::collection::ConstLinkedListIterator< T >::moveNext ( ) [virtual] moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip
Returns

true, if there is a next element, otherwise false

Implements **tool::collection::ConstIterator< T >** (p. 415).

template<class T > virtual T const& tool::collection::ConstLinkedListIterator< T >::current ( ) [virtual] returns the current element **moveNext()** (p. 418) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implements **tool::collection::ConstIterator< T >** (p. 416).

The documentation for this class was generated from the following file:

- LinkedListIterator.h

A.4.84 task::spinlock::test::Counter Class Reference

Public Member Functions

- uint32_t **getValue** ()
- void **increment** ()
- void **decrement** ()

Private Attributes

- **lock::SpinLock mutex**
- uint32_t **counterValue**

The documentation for this class was generated from the following file:

- Counter.h

A.4.85 api::task::lock::CreateSemaphoreRequest Class Reference

Creates a Semaphore.

Inherits **ipc::Request**.

Public Member Functions

- uint32_t **getSemaphore** () const

  Returns a reference to the requested semaphore.
- void **setSemaphore** (uint32_t version)

  Sets the requested reference to a semaphore.
- **CreateSemaphoreRequest** ()

  Constructor.

Static Public Attributes

- static const uint32_t **Id** = 5

  The identifier of this Request.

Private Attributes

- uint32_t **semaphore**

Additional Inherited Members

Detailed Description

Creates a Semaphore.

The documentation for this class was generated from the following file:

- **CreateSemaphoreRequest.h**

A.4.86 memory::allocator::Allocator::CriticalSection Class Reference

Helper class for entering/leaving the **Allocator** (p. 286)'s critical section.

Public Member Functions

- **CriticalSection** (**Environment** ∗**environ**)

  *Enters the **Allocator** (p. 286)'s critical section.*

- ∼**CriticalSection** ()

  Leaves the critical section.

Private Attributes

- **Environment** ∗ **environ**

  The associated environment.

Detailed Description

Helper class for entering/leaving the **Allocator** (p. 286)'s critical section.

Constructor & Destructor Documentation

memory::allocator::Allocator::CriticalSection::CriticalSection (Environment * *environ*) [inline] Enters the **Allocator** (p. 286)'s critical section.

Parameters

environ	The **Allocator** (p. 286)'s environment.

References memory::allocator::Environment::enterCriticalSection().

The documentation for this class was generated from the following file:

- **Allocator.h**

A.4.87 api::io::time::CurrentTimeRequest Class Reference

Returns the current time values which are provided by the time service.

Inherits **ipc::Request**.

Public Member Functions

- uint32_t **getTimeInSeconds** () const
- void **setTimeInSeconds** (uint32_t timeInSeconds)
- uint64_t **getUpTime** () const
- void **setUpTime** (uint64_t upTime)

Static Public Attributes

- static const uint32_t **Id** = 8

  The identifier of this Request.

Private Attributes

- uint32_t **timeInSeconds**
- uint64_t **upTime**

Additional Inherited Members

Detailed Description

Returns the current time values which are provided by the time service.

The documentation for this class was generated from the following file:

- CurrentTimeRequest.h

A.4.88 io::driver::graphics::Cursor Class Reference

Class encapsulating a cursor.

Inherited by **io::driver::graphics::vga::VgaCursor**.

Public Member Functions

- virtual **Position getPosition** () const =0

  Returns the current cursor position.

- virtual void **moveTo** (**Position** position)=0

  Changes the cursor position.

Detailed Description

Class encapsulating a cursor.

Member Function Documentation

virtual void io::driver::graphics::Cursor::moveTo (Position *position*) `[pure virtual]` Changes the cursor position.

Parameters

position	The new cursor position.

Implemented in **io::driver::graphics::vga::VgaCursor** (p. 905).

The documentation for this class was generated from the following file:

- **Cursor.h**

A.4.89 io::console::DefaultConsole Class Reference

The Console before the VGA Device initialized.

Inherits **io::console::TextConsole**.

Public Member Functions

- **DefaultConsole** (**io::driver::graphics::TextDevice** &**device**)
- virtual void **setTextColor** (Color color)

  Sets the text color.
- virtual void **setBackgroundColor** (Color color)

  Sets the text background color.
- virtual void **write** (char s)

  prints one character on the console
- virtual void **write** (char const ∗s)

  prints a primitive string on the console
- virtual void **writeLine** (char const ∗s)

  *write(s); **newLine()** (p. 424);*
- virtual void **newLine** ()

  sets a new line on the console
- virtual void **scrollUp** ()

  scrolls the screen up useful when writing at the last line of the console
- virtual void **clear** ()

  Clears the screen.
- virtual void **activate** ()

  Activates the console.
- virtual void **deactivate** ()

  Deactivates the console.

Private Member Functions

- void **doWrite** (char s)
- void **doWrite** (char const ∗s)
- void **doNewLine** ()
- void **doScrollUp** ()
- void **clearRow** (**io::driver::graphics::Position** start)
- void **cursorReturn** ()

- void **moveCursorOneColumnRight** ()
- void **moveCursorOneRowDown** ()

Private Attributes

- **task::lock::SpinLock mutex**

  Spin lock protecting concurrent access to the console.
- **io::driver::graphics::TextDevice** & **device**

  The underlying TextDevice.
- Color **textColor**

  The current foreground color.
- Color **backgroundColor**

  The current background color.

Detailed Description

The Console before the VGA Device initialized.

Writes directly to the VGA RAM. Used for early output (before VGA Device)

Member Function Documentation

virtual void io::console::DefaultConsole::setTextColor (Color *color*) `[inline]`, `[virtual]` Sets the text color.

Parameters

color	The color number to set

Implements **io::console::TextConsole** (p. 861).

References textColor.

virtual void io::console::DefaultConsole::setBackgroundColor (Color *color*) `[inline]`, `[virtual]` Sets the text background color.

Parameters

color	The color number to set

Implements **io::console::TextConsole** (p. 861).

References backgroundColor.

virtual void io::console::DefaultConsole::write (char *s*) [inline], [virtual] prints one character on the console

Parameters

s	the character to print

Implements **io::console::TextConsole** (p. 861).

References mutex.

virtual void io::console::DefaultConsole::write (char const * *s*) [inline], [virtual] prints a primitive string on the console

Parameters

s	the startpoint of the string

Implements **io::console::TextConsole** (p. 861).

References mutex.

virtual void io::console::DefaultConsole::writeLine (char const * *s*) [inline], [virtual] write(s); **newLine()** (p. 424);

Parameters

s	

Implements **io::console::TextConsole** (p. 862).

References mutex.

virtual void io::console::DefaultConsole::clear ( ) [virtual] Clears the screen.

Well only useful on screens (printer would laugh at you :))

Implements **io::console::TextConsole** (p. 862).

The documentation for this class was generated from the following file:

- **DefaultConsole.h**

A.4.90 cpu::interrupt::DefaultExceptionHandler Class Reference

Exception Handler Class Currently only one default now which prints out "-Unhadled Exception!" and halts the system.

Inherits **cpu::interrupt::InterruptHandler**.

Public Member Functions

- virtual bool **handle** (uint32_t errorCode)

  The handler.

Detailed Description

Exception Handler Class Currently only one default now which prints out "-Unhadled Exception!" and halts the system.

Member Function Documentation

virtual bool cpu::interrupt::DefaultExceptionHandler::handle (uint32_t *errorCode*) `[virtual]` The handler.

Parameters

errorCode	Error code of the error that accured

@

@plus@

@plus -@

@

@

@skip

Returns

Returns true if handled and no further handling is needed. False otherwise.

Implements **cpu::interrupt::InterruptHandler** (p. 567).

The documentation for this class was generated from the following file:

- DefaultExceptionHandler.h

A.4.91 cpu::interrupt::DefaultHandler Class Reference

Default Handler Class.

Inherits **cpu::interrupt::InterruptHandler**.

Public Member Functions

- virtual bool **handle** (uint32_t errorCode)

  The real work operation.

Detailed Description

Default Handler Class.

Used when no actual handler ist implemented. Should no happen.

Member Function Documentation

virtual bool cpu::interrupt::DefaultHandler::handle (uint32_t *errorCode*) `[virtual]` The real work operation.

Parameters

errorCode	The error code the exception rose.

@

@plus@

@plus -@

@

@

@skip

Returns

Returns allways true, since it's an default handler.

Implements **cpu::interrupt::InterruptHandler** (p. 567).

The documentation for this class was generated from the following file:

- DefaultHandler.h

A.4.92 io::driver::graphics::Delta Class Reference

Represents a difference between two positions.

Inherited by **io::driver::graphics::Position** [`private`].

Public Member Functions

- **Delta** (int32_t deltaX, int32_t deltaY, uint32_t numCols)

  *Creates a **Delta*** *(p. 429)* *object.*

- **Delta** (int32_t **offset**)

  *Creates a **Delta*** *(p. 429)* *object.*

- int32_t **getColumnDelta** (uint32_t numCols) const

  Returns the column delta (difference between X coordinates).

- int32_t **getRowDelta** (uint32_t numCols) const

  Returns the row delta (difference between Y coordinates).

- int32_t **getOffset** () const

  Returns the offset.

- void **setOffset** (int32_t **offset**)

  Sets the offset.

- bool **operator==** (**Delta** delta) const

  Returns true if this delta is equal to another one.

- bool **operator**$<$ (**Delta** delta) const

  Returns true if this delta is less than another one.

Private Attributes

- int32_t **offset**

  The offset.

Detailed Description

Represents a difference between two positions.

Constructor & Destructor Documentation

io::driver::graphics::Delta::Delta (int32_t *deltaX*, int32_t *deltaY*, uint32_t *numCols*) [inline] Creates a **Delta** (p. 429) object.

Parameters

deltaX	The column delta (difference between X coordinates).
deltaX	The row delta (difference between Y coordinates).

Referenced by io::driver::graphics::Position::operator-().

io::driver::graphics::Delta::Delta (int32_t *offset*) [inline] Creates a **Delta** (p. 429) object.

Parameters

offset	The delta's offset.

Member Function Documentation

int32_t io::driver::graphics::Delta::getColumnDelta (uint32_t *numCols*) const [inline] Returns the column delta (difference between X coordinates).

Parameters

numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

The column delta. This delta lies in the range 0 <= delta < numCols.

References offset.

Referenced by io::driver::graphics::Position::getColumn().

int32_t io::driver::graphics::Delta::getRowDelta (uint32_t *numCols*) const [inline] Returns the row delta (difference between Y coordinates).

Parameters

numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

The row delta.

References offset.

Referenced by io::driver::graphics::Position::getRow().

void io::driver::graphics::Delta::setOffset (int32_t *offset*) [inline] Sets the offset.

Parameters

offset	The offset.

References offset.

Referenced by io::driver::graphics::Position::operator+=(), io::driver::graphics::-Position::operator-=(), io::driver::graphics::Position::setColumn(), and io::driver-::graphics::Position::setRow().

bool io::driver::graphics::Delta::operator== (Delta *delta*) const [inline] Returns true if this delta is equal to another one.

Parameters

delta	The delta to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this delta has the same offset as passed one, else false.

References getOffset().

bool io::driver::graphics::Delta::operator< (Delta *delta*) const [inline]

Returns true if this delta is less than another one.

Parameters

delta	The delta to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this delta has a smaller offset as passed one, else false.

References getOffset().

The documentation for this class was generated from the following file:

- **Delta.h**

A.4.93 memory::Descriptor Class Reference

Describes a generic descriptor.

Inherited by **memory::CodeOrDataSegmentDescriptor**, and **memory::SystemSegmentDescriptor**.

Public Types

- enum **Type** { **SystemSegment16Bit** = 0, **SystemSegment32Bit** = 1, **DataSegment** = 2, **CodeSegment** = 3 }

  *Describes the general type of a **Descriptor*** (p. *432).*

Public Member Functions

- **Descriptor** ()

  Constructor. Creates an empty descriptor.

- bool **isPresent** () const

  *Returns true if the **Descriptor*** (p. *432) is present.*

- **Descriptor** & **setPresent** (bool present)

  Sets the Present flag.

- uint32_t **getDPL** () const

  *Returns the **Descriptor*** (p. *432) Privilege Level.*

- **Descriptor** & **setDPL** (uint32_t dpl)

  Sets the DPL.

- **Type getType** () const

  *Returns the **Descriptor*** (p. *432)'s type.*

- **Descriptor** & **setType** (**Type type**)

  Sets the type.

Data Fields

- enum **memory::Descriptor::Type __attribute__**

Protected Attributes

- uint32_t **data** [2]

  The descriptor's data.

Private Types

- typedef **tool::BitField**
  < uint32_t, 1, 11, 2 > **TypeField**

  Describes the type field.

- typedef **tool::BitField**
  < uint32_t, 1, 13, 2 > **DPLField**

  Describes the DPL field.

- typedef **tool::BitField**
  < uint32_t, 1, 15, 1 > **PresentField**

  Describes the Present flag.

Detailed Description

Describes a generic descriptor.

Member Enumeration Documentation

enum memory::Descriptor::Type Describes the general type of a **Descriptor** (p. 432).

Enumerator

SystemSegment16Bit system segment, 16-bit variant

SystemSegment32Bit system segment, 32-bit variant

DataSegment data segment

CodeSegment code segment

Member Function Documentation

bool memory::Descriptor::isPresent ( ) const [inline] Returns true if the **Descriptor** (p. 432) is present.

If a **Descriptor** (p. 432) is not present, referencing it through a **Selector** (p. 811) produces an exception.

References data, and tool::BitField< Base, Index, BitPosition, Length >::get().

Descriptor& memory::Descriptor::setPresent (bool *present*) [inline] Sets the Present flag.

Parameters

present	The Present flag.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References data, and tool::BitField< Base, Index, BitPosition, Length >::set().

Referenced by cpu::GDTManager::free().

Descriptor& memory::Descriptor::setDPL (uint32_t *dpl*) [inline]
Sets the DPL.

Parameters

dpl	The new DPL.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References data, and tool::BitField< Base, Index, BitPosition, Length >::set().

Descriptor& memory::Descriptor::setType (Type *type*) [inline] Sets the type.

Parameters

type	The new type.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

References data, and tool::BitField< Base, Index, BitPosition, Length >::set().

The documentation for this class was generated from the following file:

- Descriptor.h

A.4.94 io::driver::Device Class Reference

Abstract class which represents all devices in the system.

Inherited by **io::driver::block::ata::AtaBusDevice**, **io::driver::block::BlockDevice**, **io::driver::charlayout::CharLayoutDevice**, **io::driver::classes::AudioClass**, **io::driver::graphics::TextDevice**, **io::driver::interrupt::PICDevice**, **io::driver::keycode::KeycodeDevice**, **io::driver::pstwo::PS2Device**, **io::driver::pstwokeyboard::PS2KeyboardDevice**, **io::driver::rtc::RTCDevice**, **io::driver::test::TestDevice**, **io::driver::test::TestDevice2**, and **io::driver::timer::PIT8254Device**.

Public Member Functions

- **Device** (int ClassID)
- int **getDeviceClassID** () const

  Getter for the deviceClassID.

Private Attributes

- int **deviceClassID**

Detailed Description

Abstract class which represents all devices in the system.

The documentation for this class was generated from the following file:

- Device.h

A.4.95 io::driver::DeviceClass Class Reference

Abstract device class.

Detailed Description

Abstract device class.

The documentation for this class was generated from the following file:

- DeviceClass.h

A.4.96 io::driver::DeviceManager Class Reference

As a Singleton the **DeviceManager** (p. 437) is responsible for handling the addition and removing of devices.

Public Member Functions

- void **add** (**Device** ∗dev)

  *Adds the **Device** (p. 436).*

- template<class T >
  T ∗ **get** (int deviceID)

  Returns the first.

- void **remove** (**Device** ∗dev)

  *Removes the **Device** (p. 436).*

Static Public Member Functions

- static **DeviceManager** & **getInstance** ()

  Returns.

Private Types

- typedef
  tool::collection::LinearMap
  < int, **Device** ∗ > **Devices**

  the list of the devices the manager manages

Private Member Functions

- **DeviceManager** ()

  *Private constructor for class **DeviceManager*** (p. *437).*

Static Private Member Functions

- static void **init** ()

Private Attributes

- **Devices registeredDevices**

Static Private Attributes

- static **DeviceManager theInstance**

  the static attribute for the singleton pattern

Friends

- class **boot::BootManager**

Detailed Description

As a Singleton the **DeviceManager** (p. 437) is responsible for handling the addition and removing of devices.

Constructor & Destructor Documentation

io::driver::DeviceManager::DeviceManager ( ) [inline], [private] Private constructor for class **DeviceManager** (p. 437).

Use **DeviceManager::getInstance()** (p. 439) instead.

Member Function Documentation

static DeviceManager& io::driver::DeviceManager::getInstance ( ) [inline], [static] Returns.

@

@plus@

@plus -@

@

@

@skip
Returns

the instance of this singleton **DeviceManager** (p. 437).

References theInstance.

void io::driver::DeviceManager::add (Device * *dev*) [inline] Adds the **Device** (p. 436).

Parameters

dev	to the list of registered Devices. Registered Devices can be found by the **DeviceManager** (p. 437).

References tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >-::add(), and io::driver::Device::getDeviceClassID().

template<class T > T* io::driver::DeviceManager::get (int *deviceID*) [inline] Returns the first.

@

@plus@

@plus -@

@

@

@skip
Returns

Device (p. 436) pointer to the Device-Object that is typeOf the

Parameters

device-Class.	

References tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >-::get().

void io::driver::DeviceManager::remove (Device * *dev*) [inline] Removes the **Device** (p. 436).

Parameters

dev	from the list of registered Devices.

References tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >-::remove().

The documentation for this class was generated from the following file:

- DeviceManager.h

A.4.97 io::vfs::Directory Class Reference

Inherits **io::vfs::FileSystemNode**.

Inherited by **io::vfs::fat::FatDirectory**, **io::vfs::fat::FatRootDirectory**, and **io::vfs::test::DummyDirectory**.

Public Member Functions

- **Directory** (**Mount** ∗m)
- virtual void **accept** (**FileSystemNodeVisitor** ∗v)
- virtual bool **isFile** ()
- virtual const char ∗ **getName** () const =0
- virtual uint32_t **getSize** ()
- virtual RemoveError::enum_t **removeNode** (const char ∗name)=0

  Removes the given node from the file system.
- virtual **object::Ref**< **Directory** > **getParent** ()=0
- virtual **tool::collection::List**
  < **object::Ref**< **FileSystemNode** > > ∗ **getNodes** ()=0

  Returns a list of all nodes in this directory.
- virtual **object::Ref**
  < **FileSystemNode** > **getNode** (const char ∗name)=0

  Returns the node with the given name.
- virtual **object::Ref**< **Directory** > **getNodeAsDirectory** (const char ∗name)

  Returns the directory with the given name.
- virtual **object::Ref**< **File** > **getNodeAsFile** (const char ∗name)

  Returns the file with the given name.
- virtual **object::Ref**< **File** > **createFile** (const char ∗name)=0

  Creates a new file with the given name.
- virtual **object::Ref**< **Directory** > **createDirectory** (const char ∗name)=0

  Creates a new directory with the given name.

Additional Inherited Members

Constructor & Destructor Documentation

io::vfs::Directory::Directory (Mount ∗ *m*) @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::FileSystemNode() (p. 535)

Member Function Documentation

virtual void io::vfs::Directory::accept (FileSystemNodeVisitor * *v*) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::accept() (p. 535)

Implements **io::vfs::FileSystemNode** (p. 535).

virtual bool io::vfs::Directory::isFile ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::isFile() (p. 535)

Implements **io::vfs::FileSystemNode** (p. 535).

virtual const char* io::vfs::Directory::getName ( ) const [pure virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getName() (p. 536)

Implements **io::vfs::FileSystemNode** (p. 536).

Implemented in **io::vfs::fat::FatRootDirectory** (p. 520), and **io::vfs::test::-DummyDirectory** (p. 463).

virtual uint32_t io::vfs::Directory::getSize ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getSize() (p. 536)

Implements **io::vfs::FileSystemNode** (p. 536).

Reimplemented in **io::vfs::fat::FatRootDirectory** (p. 521), and **io::vfs::fat::-FatDirectory** (p. 510).

virtual RemoveError::enum_t io::vfs::Directory::removeNode (const char * *name*) [pure virtual] Removes the given node from the file system.

Only nodes within this directory can be removed. @

@plus@

@plus -@

@

@

@skip

Returns

SUCCESS if the node was removed INUSE if the node is still in use DIRETORYNOTEMPTY if the node is a directory and is not empty NOTFOUND if no node with the given name could be found

Implemented in **io::vfs::fat::FatRootDirectory** (p. 521), **io::vfs::fat::FatDirectory** (p. 510), and **io::vfs::test::DummyDirectory** (p. 463).

virtual object::Ref<Directory> io::vfs::Directory::getParent ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::FileSystemNode** (p. 537).

Implemented in **io::vfs::fat::FatRootDirectory** (p. 522), **io::vfs::fat::FatDirectory** (p. 510), and **io::vfs::test::DummyDirectory** (p. 464).

virtual tool::collection::List<object::Ref<FileSystemNode> >* io::vfs::Directory::getNodes ( ) [pure virtual] Returns a list of all nodes in this directory.

@

@plus@

@plus -@

@

@

@skip
Returns

A list of nodes.

Implemented in **io::vfs::fat::FatRootDirectory** (p. 522), **io::vfs::fat::FatDirectory** (p. 511), and **io::vfs::test::DummyDirectory** (p. 464).

virtual object::Ref<FileSystemNode> io::vfs::Directory::getNode (const char * *name*) [pure virtual] Returns the node with the given name.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip

Returns

The reference to the node. Null, if no node was found.

Implemented in **io::vfs::fat::FatRootDirectory** (p. 522), **io::vfs::fat::FatDirectory** (p. 511), and **io::vfs::test::DummyDirectory** (p. 465).

virtual object::Ref<Directory> io::vfs::Directory::getNodeAsDirectory (const char * *name*) [virtual] Returns the directory with the given name.

Returns NULL, if no directory with the given name were found.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip

Returns

The reference to the node. Null, if no node was found.

virtual object::Ref<File> io::vfs::Directory::getNodeAsFile (const char * *name*) [virtual] Returns the file with the given name.

Returns NULL, if no file with the given name were found.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip

Returns

The reference to the node. Null, if no node was found.

virtual object::Ref<File> io::vfs::Directory::createFile (const char * *name*) [pure virtual] Creates a new file with the given name.

Parameters

name	The name of the new file.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created file. Null, if a node with the given name is already present.

Implemented in **io::vfs::fat::FatRootDirectory** (p. 523), **io::vfs::fat::FatDirectory** (p. 512), and **io::vfs::test::DummyDirectory** (p. 466).

virtual object::Ref<Directory> io::vfs::Directory::createDirectory (const char * *name*) [pure virtual] Creates a new directory with the given name.

Parameters

name	The name of the new directory.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created directory. Null, if a node with the given name is already present.

Implemented in **io::vfs::fat::FatRootDirectory** (p. 523), **io::vfs::fat::FatDirectory** (p. 512), and **io::vfs::test::DummyDirectory** (p. 466).

The documentation for this class was generated from the following file:

- Directory.h

A.4.98 io::vfs::fat::DirectoryEntry Struct Reference

Public Member Functions

- bool **operator==** (**DirectoryEntry** const &other) const

Data Fields

- char **filename** [11]
- uint8_t **fileAttributes**
- uint8_t **reserved**
- uint8_t **createTimeTenth**

- uint32_t **createDateTime**
- uint16_t **lastAccess**
- uint16_t **startFirstCluster**
- uint32_t **lastWriteAccess**
- uint16_t **firstCluster**
- uint32_t **fileSize**

The documentation for this struct was generated from the following file:

- DirectoryEntry.h

A.4.99 Disk Class Reference

Represents a disk.

Public Member Functions

- **Disk** (**BPB** const ∗**bpb**)

  *Creates a **Disk** (p. 448) object.*

- **BPB** const ∗ **getBPB** () const

  *Returns the underlying **BPB** (p. 375).*

- bool **loadSectors** (void ∗buffer, const **DiskAddress** &address, unsigned num) const

  Loads a number of sectors from this disk.

Private Attributes

- **BPB** const ∗const **bpb**

  *The underlying **BPB** (p. 375).*

Detailed Description

Represents a disk.

Constructor & Destructor Documentation

Disk::Disk (BPB const * *bpb*) [inline] Creates a **Disk** (p. 448) object.

Parameters

bpb	The underlying **BPB** (p. 375) describing the disk. The lifetime of the **BPB** (p. 375) object must exceed the lifetime of this **Disk** (p. 448) object.

Member Function Documentation

BPB const* Disk::getBPB ( ) const [inline] Returns the underlying **BPB** (p. 375).

@

@plus@

@plus -@

@

@

@skip
Returns

the underlying **BPB** (p. 375)

References bpb.

Referenced by FAT::mapClusterToSector().

bool Disk::loadSectors (void * *buffer*, const DiskAddress & *address*, unsigned *num*) const Loads a number of sectors from this disk.

Parameters

buffer	The buffer where to store the sectors.
address	The disk address where the sectors start.
num	The number of sectors to load.

@

@plus@

@plus -@

@

@

@skip

Returns

true if loading succeeded, false otherwise.

The documentation for this class was generated from the following file:

- **disk.h**

A.4.100 DiskAddress Class Reference

Represents a disk address.

Public Member Functions

- **DiskAddress** (unsigned long **sector**)

  Creates a disk address.

- **DiskAddress** & **operator+=** (unsigned const delta)

  Increments the address by some value.

- **DiskAddress** & **operator-=** (unsigned const delta)

  Decrements the address by some value.

- **DiskAddress** & **operator++** ()

  Increments the address by one.

- **DiskAddress** & **operator--** ()

  Decrements the address by one.

- unsigned long **getSector** () const

  Returns the logical sector number.

- unsigned **getHead** (**BPB** const ∗bpb) const

  Returns the head of the address (starting at zero).

- unsigned **getTrack** (**BPB** const ∗bpb) const

  Returns the track of the address (starting at zero).

- unsigned **getSectorOnTrack** (**BPB** const ∗bpb) const

  Returns the sector of the address (starting at one).

Private Attributes

- unsigned long **sector**

  The logical sector number.

Detailed Description

Represents a disk address.

Constructor & Destructor Documentation

DiskAddress::DiskAddress (unsigned long *sector*) [inline] Creates a disk address.

Parameters

sector	The logical sector number (starting at zero).

Member Function Documentation

DiskAddress& DiskAddress::operator+= (unsigned const *delta*) [inline] Increments the address by some value.

Parameters

delta	The number of sectors to add.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

References sector.

DiskAddress& DiskAddress::operator-= (unsigned const *delta*) [inline] Decrements the address by some value.

Parameters

delta	The number of sectors to subtract.

@

@plus@

@plus -@

@

@

@skip
Returns

∗this

References sector.

DiskAddress& DiskAddress::operator++ ( ) [inline] Increments the address by one.

@

@plus@

@plus -@

@

@

@skip
Returns

∗this

DiskAddress& DiskAddress::operator-- ( ) [inline] Decrements the address by one.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

unsigned DiskAddress::getHead (BPB const * *bpb*) const [inline]
Returns the head of the address (starting at zero).

Parameters

bpb	The **BPB** (p. 375) to use for LBA –> CHS translation.

@

@plus@

@plus -@

@

@

@skip
Returns

The head of the CHS address.

References BPB::numHeads, BPB::numSectorsPerTrack, and sector.

unsigned DiskAddress::getTrack (BPB const * *bpb*) const [inline]
Returns the track of the address (starting at zero).

Parameters

bpb	The **BPB** (p. 375) to use for LBA –> CHS translation.

@

@plus@

@plus -@

@

@

@skip
Returns

The track (cylinder) of the CHS address.

References BPB::numHeads, BPB::numSectorsPerTrack, and sector.

unsigned DiskAddress::getSectorOnTrack (BPB const * *bpb*) const [inline] Returns the sector of the address (starting at one).

Parameters

bpb	The **BPB** (p. 375) to use for LBA –> CHS translation.

@

@plus@

@plus -@

@

@

@skip

Returns

The sector of the CHS address.

References BPB::numSectorsPerTrack, and sector.

The documentation for this class was generated from the following file:

- **disk.h**

A.4.101 io::driver::Driver Class Reference

Abstract class **Driver** (p. 455) represents all drivers in a system.

Inherited by **io::driver::block::ata::AtaBusDriver**, **io::driver::block::ata::-AtaDriver**, **io::driver::charlayout::CharLayoutDriver**, **io::driver::graphics-::vga::VgaDriver**, **io::driver::interrupt::PICDriver**, **io::driver::keycode::-KeycodeDriver**, **io::driver::pstwo::PS2Driver**, **io::driver::pstwokeyboard-::PS2KeyboardDriver**, **io::driver::rtc::RTCDriver**, **io::driver::speaker::Speaker-Driver**, **io::driver::test::TestDriver**, **io::driver::test::TestDriver2**, and **io-::driver::timer::PIT8254Driver**.

Public Types

- typedef
  tool::collection::ArrayList
  < int > **Dependencies**

Public Member Functions

- virtual void **checkDev** ()=0

  Checks if a new device for this driver is available and initiates a corresponding device-object.

- **Dependencies** const & **getDependencies** () const

  Getter for list of dependencies of this driver.

- int **getMyClassId** () const

  *Getter for ClassID of this **Driver** (p. 455).*

Protected Member Functions

- **Driver** (int classId)

  constructor with id and dependencies, if it has no dependencies use NULL

Protected Attributes

- **Dependencies dependencies**

Private Attributes

- int **myClassId**

Detailed Description

Abstract class **Driver** (p. 455) represents all drivers in a system.

A driver only provides the capability for creating a device and manage dependencies for this device. In class **Device** (p. 436) all operations are specified to use a hardware device.

Member Function Documentation

int io::driver::Driver::getMyClassId ( ) const [inline] Getter for ClassID of this **Driver** (p. 455).

@

@plus@

@plus -@

@

@

@skip
Returns

The documentation for this class was generated from the following file:

- Driver.h

A.4.102 io::driver::DriverManager::DriverDescriptor Class Reference

Describes a **Driver** (p. 455) within the **DriverManager** (p. 459).

Public Member Functions

- **DriverDescriptor** (**Driver** ∗**driver**)

  Constructor.

- **Driver** ∗ **getDriver** () const

  *Returns a pointer to the encapsulated **Driver** (p. 455) object.*

- bool **isDriverInitialized** () const

  *Returns true if the associated **Driver** (p. 455) has already been initialized.*

- void **setDriverInitialized** ()

  *Marks the associated **Driver** (p. 455) as initialized.*

- bool **operator==** (**DriverDescriptor** const &other) const

  *Compares this object with another **DriverDescriptor** (p. 457) object.*

Private Attributes

- **Driver** ∗ **driver**

  *The encapsulated **Driver** (p. 455).*

- bool **isInitialized**

  *True if the associated **Driver** (p. 455) has already been initialized.*

Detailed Description

Describes a **Driver** (p. 455) within the **DriverManager** (p. 459).

Constructor & Destructor Documentation

io::driver::DriverManager::DriverDescriptor::DriverDescriptor (Driver * *driver*) [inline] Constructor.

Marks the driver as not initialized.

Parameters

driver	A pointer to the **Driver** (p. 455) object.

Member Function Documentation

bool io::driver::DriverManager::DriverDescriptor::operator== (DriverDescriptor const & *other*) const [inline] Compares this object with another **DriverDescriptor** (p. 457) object.

Parameters

other	The other **DriverDescriptor** (p. 457).

@

@plus@

@plus -@

@

@

@skip

Returns

True if both objects refer to the same **Driver** (p. 455), else false.

References driver.

The documentation for this class was generated from the following file:

- DriverManager.h

A.4.103 io::driver::test::DriverFrameworkTestCase Class Reference

Test of test.

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()

  Tests the functionality of the driver framework.

- virtual const char ∗ **getName** ()

Additional Inherited Members

Detailed Description

Test of test.

The documentation for this class was generated from the following file:

- DriverFrameworkTestCase.h

A.4.104 io::driver::DriverManager Class Reference

As a Singleton the **DriverManager** (p. 459) is responsible for handling the addition and removing of drivers.

Data Structures

- class **DriverDescriptor**

  *Describes a **Driver** (p. 455) within the **DriverManager** (p. 459).*

Public Member Functions

- void **add** (**Driver** ∗drv)

  *Registers the given **Driver** (p. 455) at the **DriverManager** (p. 459).*

- void **remove** (**Driver** ∗drv)

*Deregisters the given **Driver** (p. 455) from the **DriverManager** (p. 459).*

- void **initializeDrivers** ()

  Checks the dependencies for every registered but uninitialized driver and tries to create a device for the driver.

Static Public Member Functions

- static **DriverManager** & **getInstance** ()

  Returns a reference to the singleton instance of this class.

Private Types

- typedef
  tool::collection::ArrayList
  < **DriverDescriptor** > **Drivers**

Private Member Functions

- **DriverManager** ()

  *Private constructor for class **DriverManager** (p. 459).*

Static Private Member Functions

- static void **init** ()

  Initializes the singleton instance of this class.

Private Attributes

- **Drivers registeredDrivers**

  The registered Drivers.

Static Private Attributes

- static **DriverManager theInstance**

  The singleton instance of this class.

Friends

- class **boot::BootManager**

Detailed Description

As a Singleton the **DriverManager** (p. 459) is responsible for handling the addition and removing of drivers.

Takes care for dependencies in the drivers an initializes a device for a corresponding driver.

Constructor & Destructor Documentation

io::driver::DriverManager::DriverManager ( ) [inline], [private] Private constructor for class **DriverManager** (p. 459).

Use **DriverManager::getInstance()** (p. 460) instead.

Referenced by init().

Member Function Documentation

void io::driver::DriverManager::add (Driver * *drv*) [inline] Registers the given **Driver** (p. 455) at the **DriverManager** (p. 459).

Parameters

drv	The **Driver** (p. 455) to register.

References tool::collection::ArrayList< T >::add(), and registeredDrivers.

void io::driver::DriverManager::remove (Driver * *drv*) [inline] Deregisters the given **Driver** (p. 455) from the **DriverManager** (p. 459).

Parameters

drv	The **Driver** (p. 455) to deregister.

References registeredDrivers, and tool::collection::ArrayList< T >::remove().

void io::driver::DriverManager::initializeDrivers ( ) Checks the dependencies for every registered but uninitialized driver and tries to create a device for the driver.

This operation can be called multiple times if drivers are registered at different points of time. An initialized driver is never reinitialized again.

The documentation for this class was generated from the following file:

- DriverManager.h

A.4.105 io::vfs::test::DummyDirectory Class Reference

Inherits **io::vfs::Directory**.

Public Member Functions

- **DummyDirectory** (**Mount** ∗m, **DummyNode** ∗node)
- virtual const char ∗ **getName** () const
- virtual RemoveError::enum_t **removeNode** (const char ∗name)

  Removes the given node from the file system.
- virtual ::**object::Ref**< **Directory** > **getParent** ()
- virtual ::**tool::collection::List** < ::**object::Ref** < **FileSystemNode** > > ∗ **getNodes** ()

  Returns a list of all nodes in this directory.
- virtual ::**object::Ref** < **FileSystemNode** > **getNode** (const char ∗name)

  Returns the node with the given name.
- virtual ::**object::Ref**< **File** > **createFile** (const char ∗name)

  Creates a new file with the given name.
- virtual ::**object::Ref**< **Directory** > **createDirectory** (const char ∗name)

  Creates a new directory with the given name.

Private Member Functions

- **AbstractDummyNode** ∗ **findNode** (const char ∗name)

Private Attributes

- **DummyNode** ∗ **node**

Additional Inherited Members

Member Function Documentation

virtual const char∗ io::vfs::test::DummyDirectory::getName ( ) const [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getName() (p. 536)

Implements **io::vfs::Directory** (p. 442).

virtual RemoveError::enum_t io::vfs::test::DummyDirectory::removeNode (const char ∗ *name*) [virtual] Removes the given node from the file system.

Only nodes within this directory can be removed. @

@plus@

@plus -@

@

@

@skip

Returns

SUCCESS if the node was removed INUSE if the node is still in use DIRETORYNOTEMPTY if the node is a directory and is not empty NOTFOUND if no node with the given name could be found

Implements **io::vfs::Directory** (p. 443).

virtual ::object::Ref<Directory> io::vfs::test::DummyDirectory::getParent ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::Directory** (p. 444).

virtual ::tool::collection::List< ::object::Ref<FileSystemNode> >* io::vfs::test::DummyDirectory::getNodes ( ) [virtual] Returns a list of all nodes in this directory.

@

@plus@

@plus -@

@

@

@skip
Returns

A list of nodes.

Implements **io::vfs::Directory** (p. 444).

virtual ::object::Ref<FileSystemNode> io::vfs::test::DummyDirectory-::getNode (const char * *name*) [virtual] Returns the node with the given name.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip

Returns

The reference to the node. Null, if no node was found.

Implements **io::vfs::Directory** (p. 445).

virtual ::object::Ref<File> io::vfs::test::DummyDirectory::createFile (const char * *name*) [virtual] Creates a new file with the given name.

Parameters

name	The name of the new file.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created file. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 446).

virtual ::object::Ref<Directory> io::vfs::test::DummyDirectory::create-Directory (const char * *name*) [virtual] Creates a new directory with the given name.

Parameters

name	The name of the new directory.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created directory. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 447).

The documentation for this class was generated from the following file:

- DummyDirectory.h

A.4.106 io::vfs::test::DummyFile Class Reference

Inherits **io::vfs::File**.

Public Member Functions

- **DummyFile** (**Mount** ∗m, **DummyLeaf** ∗leaf)
- virtual const char ∗ **getName** () const
- virtual uint32_t **getSize** ()
- virtual ::**object::Ref**< **Directory** > **getParent** ()
- virtual **FileStream** ∗ **open** (ReadWrite::enum_t rw)

  Opens the file and creates a file stream.

Private Attributes

- **DummyLeaf** ∗ **leaf**

Friends

- class **DummyDirectory**

Additional Inherited Members

Member Function Documentation

virtual const char* io::vfs::test::DummyFile::getName ( ) const [virtual]

@

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getName() (p. 536)

Implements **io::vfs::File** (p. 527).

virtual uint32_t io::vfs::test::DummyFile::getSize ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getSize() (p. 536)

Implements **io::vfs::File** (p. 527).

virtual ::object::Ref<Directory> io::vfs::test::DummyFile::getParent ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::File** (p. 527).

virtual FileStream∗ io::vfs::test::DummyFile::open (ReadWrite::enum_t *rw*) [virtual] Opens the file and creates a file stream.

Parameters

rw	Permission level.

@

@plus@

@plus -@

@

@

@skip

Returns

An open file stream.

Implements **io::vfs::File** (p. 528).

The documentation for this class was generated from the following file:

- DummyFile.h

A.4.107 io::vfs::test::DummyFileStream Class Reference

Inherits **io::vfs::FileStream**.

Public Member Functions

- **DummyFileStream** (::**object::Ref**< **File** > f, ReadWrite::enum_t **rw**, **DummyLeaf** ∗leaf)

- virtual uint32_t **seek** (int32_t offset, SeekOrigin::enum_t origin)

  Change the cursor in the stream.

- virtual uint32_t **read** (unsigned char ∗buffer, uint32_t length)

  Reads bytes from the stream.

Protected Member Functions

- virtual void **doWrite** (unsigned char ∗buffer, uint32_t length)

  Writes bytes to the stream.

Private Attributes

- uint32_t **position**
- **DummyLeaf** ∗ **leaf**

Member Function Documentation

virtual uint32_t io::vfs::test::DummyFileStream::seek (int32_t *offset*, SeekOrigin::enum_t *origin*) [virtual] Change the cursor in the stream.

Parameters

offset	Count of bytes to move the cursor. Can be positive or negative.
origin	From which position should the cursor be moved.

@

@plus@

@plus -@

@

@

@skip

Returns

The current Position in the stream.

Implements **io::vfs::FileStream** (p. 530).

virtual uint32_t io::vfs::test::DummyFileStream::read (unsigned char * *buffer,* uint32_t *length*) [virtual] Reads bytes from the stream.

Parameters

buffer	This buffer will be filled with read bytes. Should be as large as length
length	The count of bytes to read.

@

@plus@

@plus -@

@

@

@skip

Returns

The number of read bytes. Can be less than length if the end of the file is reached.

Implements **io::vfs::FileStream** (p. 530).

virtual void io::vfs::test::DummyFileStream::doWrite (unsigned char * *buffer,* uint32_t *length*) `[protected]`**,** `[virtual]` Writes bytes to the stream.

Can only be called if this stream has write permissions.

Parameters

buffer	This buffer will be written to the file.
length	The length of the buffer.

Implements **io::vfs::FileStream** (p. 531).

The documentation for this class was generated from the following file:

- DummyFileStream.h

A.4.108 io::vfs::test::DummyFileSystem Class Reference

Inherits **io::vfs::FileSystem**.

Public Member Functions

- virtual bool **canMount** (**Volume** *v)

Determines if this file system can mount the given volume.

- virtual **Mount** ∗ **mount** (uint8_t mountID, **Volume** ∗v)

  Mounts the given volume.

Member Function Documentation

virtual bool io::vfs::test::DummyFileSystem::canMount (Volume ∗ *v*) [inline], [virtual] Determines if this file system can mount the given volume.

An actual mount is not performed.

Parameters

v	The volume that should be checked

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the volume could be mounted with this file system. Otherwise false.

Implements **io::vfs::FileSystem** (p. 532).

virtual Mount∗ io::vfs::test::DummyFileSystem::mount (uint8_t *mountID*, Volume ∗ *v*) [inline], [virtual] Mounts the given volume.

Parameters

v	The volume to mount

@

@plus@

@plus -@

@

@

@skip

Returns

The created mount

Implements **io::vfs::FileSystem** (p. 533).

The documentation for this class was generated from the following file:

- DummyFileSystem.h

A.4.109 io::vfs::test::DummyLeaf Class Reference

Inherits **io::vfs::test::AbstractDummyNode**.

Public Member Functions

- **DummyLeaf** (const char ∗name)
- char **getByte** (uint32_t pos) const
- void **setByte** (uint32_t pos, char c)
- uint32_t **getSize** () const
- virtual bool **isFile** () const
- bool **isOpened** () const
- void **setOpened** (bool opened)

Private Attributes

- char ∗ **content**
- uint32_t **length**
- bool **opened**

The documentation for this class was generated from the following file:

- DummyComposite.h

A.4.110 io::vfs::test::DummyMount Class Reference

Inherits **io::vfs::Mount**.

Public Member Functions

- **DummyMount** (uint8_t **mountID**)
- virtual ::**object::Ref**< **Directory** > **getRoot** ()

Protected Member Functions

- virtual void **doUmount** ()

  This method does the actual umounting.

Private Attributes

- **DummyNode** ∗ **root**

Member Function Documentation

virtual ::object::Ref<Directory> io::vfs::test::DummyMount::getRoot ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

AbstractMount::getRoot() (p. 265)

Implements **io::vfs::Mount** (p. 657).

The documentation for this class was generated from the following file:

- DummyMount.h

A.4.111 io::vfs::test::DummyNode Class Reference

Inherits **io::vfs::test::AbstractDummyNode**.

Public Member Functions

- **DummyNode** (const char *name)
- virtual bool **isFile** () const
- ::**tool::collection::List**
  < **AbstractDummyNode** * > * **getNodes** () const

Private Attributes

- ::**tool::collection::List**
  < **AbstractDummyNode** * > * **nodes**

The documentation for this class was generated from the following file:

- DummyComposite.h

A.4.112 memory::E820_entry Struct Reference

Describes a memory block.

Public Types

- enum **Type** { **Free** = 1, **ReservedBySystem**, **ReservedByACPIReclaimable**, **ReservedByACPI_NVS** }

  Possible types of a memory block.

Data Fields

- enum **memory::E820_entry::Type __attribute__**
- unsigned long long **base**

  The base address of the memory block.
- unsigned long long **length**

  The length of the memory block in bytes.
- unsigned long **type**

  The type of the memory block.
- **E820_entry** * **next**

Points to the next entry if available.

Detailed Description

Describes a memory block.

Member Enumeration Documentation

enum memory::E820_entry::Type Possible types of a memory block.

Enumerator

- ***Free*** memory can be used by the OS unconditionally
- ***ReservedBySystem*** memory is reserved and must not be used by the OS
- ***ReservedByACPIReclaimable*** memory can be used by the OS after processing ACPI tables
- ***ReservedByACPI_NVS*** memory must be saved during ACPI NVS (non-volatile sleeping) sessions

Field Documentation

enum memory::E820_entry::Type memory::E820_entry::__attribute__

unsigned long long memory::E820_entry::base The base address of the memory block.

unsigned long long memory::E820_entry::length The length of the memory block in bytes.

unsigned long memory::E820_entry::type The type of the memory block.

E820_entry∗ memory::E820_entry::next Points to the next entry if available.

The documentation for this struct was generated from the following file:

- E820.h

A.4.113 fosCli::commands::EchoCliCommand Class Reference

Writes the given parameter separated with a space to the console.

Inherits **fosCli::parser::CliCommand**.

Public Member Functions

- virtual void **execute** ()
- virtual void **addParameter** (char const ∗parameter)

Private Attributes

- const char ∗ **output**

Detailed Description

Writes the given parameter separated with a space to the console.

The documentation for this class was generated from the following file:

- EchoCliCommand.h

A.4.114 fosCli::commands::EchoCommandCreator Class Reference

Inherits **fosCli::parser::CliCommandCreator**.

Public Member Functions

- virtual **fosCli::parser::CliCommand** ∗ **create** ()

The documentation for this class was generated from the following file:

- EchoCommandCreator.h

A.4.115 RootDirectory::Entry Struct Reference

Describes a directory entry.

Data Structures

- struct **Callback**

  Describes a callback. This is an interface.

Public Types

- enum **Attribute** {
  ReadOnly = 0x1, **Hidden** = 0x2, **System** = 0x4, **VolumeID** = 0x8, **Directory** = 0x10, **Archive** = 0x20, **LongName** = ReadOnly | Hidden | System | VolumeID, **Mask** = ReadOnly | Hidden | System | VolumeID | Directory | Archive }

  Describes an entry's attribute.

Public Member Functions

- bool **load** (**FAT** const &fat, void ∗**buffer**, unsigned size, **Callback** &callback) const

  Loads the contents of this entry into memory.

Data Fields

- char **name** [11]

  The entry's name, padded with blanks.

- unsigned char **attributes**

  The entry's attributes (read only, hidden etc.).

- unsigned char **reservedNT**

  Reserved. Do not use.

- unsigned char **createTimeTenth**

  Number of tenth of seconds to add to the time of creation.

- unsigned short **createTime**

The time of creation.

- unsigned short **createDate**

  The date of creation.

- unsigned short **lastAccessDate**

  The date of the last access.

- unsigned short **firstClusterHigh**

  High word of the entry's start cluster.

- unsigned short **lastWriteTime**

  The time of the last write access.

- unsigned short **lastWriteDate**

  The date of the last write access.

- unsigned short **firstClusterLow**

  Low word of the entry's start cluster.

- unsigned long **fileSize**

  The size of the entry in bytes.

Detailed Description

Describes a directory entry.

Member Enumeration Documentation

enum RootDirectory::Entry::Attribute Describes an entry's attribute.

Enumerator

ReadOnly Read-only file.
Hidden Hidden file.
System System file.
VolumeID Volume Identifier.
Directory Directory.
Archive File should be archived.
LongName **Entry** (p. 479) is a VFAT long name entry.
Mask Combines all bit flags.

Member Function Documentation

bool RootDirectory::Entry::load (FAT const & *fat,* void ∗ *buffer,* unsigned *size,* Callback & *callback*) const Loads the contents of this entry into memory.

Parameters

fat	The **FAT** (p. 497) object to use.
buffer	The buffer to use.
size	The size of the buffer. At least one full cluster needs to fit into the buffer.
callback	The callback which will have its run() operation called whenever the buffer is full or at the end of the loading process.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false otherwise.

The documentation for this struct was generated from the following file:

- **dir.h**

A.4.116 memory::allocator::Environment Class Reference

Allows the **Allocator** (p. 286) to communicate with its environment.

Inherited by **lib::runtime::UserEnvironment**, and **memory::KernelEnvironment**.

Public Member Functions

- virtual ∼**Environment** ()

  Destructor.

Locking

- virtual void **enterCriticalSection** ()=0

  *Enters the **Allocator** (p. 286)'s critical section.*

- virtual void **leaveCriticalSection** ()=0

  *Leaves the **Allocator** (p. 286)'s critical section.*

Memory management

- virtual char * **allocateBlock** (uint32_t pages)=0

  Allocates a memory block.

- virtual bool **freeBlock** (void *ptr, uint32_t pages)=0

  Frees a previously allocated memory block.

Detailed Description

Allows the **Allocator** (p. 286) to communicate with its environment.

Member Function Documentation

virtual char* memory::allocator::Environment::allocateBlock (uint32_t *pages*) [pure virtual] Allocates a memory block.

Parameters

pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the allocated block or NULL if allocation failed.

Implemented in **memory::KernelEnvironment** (p. 583), and **lib::runtime::UserEnvironment** (p. 894).

virtual bool memory::allocator::Environment::freeBlock (void * *ptr*, uint32_t *pages*) [pure virtual] Frees a previously allocated memory block.

Parameters

ptr	The pointer to the previously allocated block.
pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the block could be freed, false otherwise.

Implemented in **memory::KernelEnvironment** (p. 583), and **lib::runtime::-UserEnvironment** (p. 894).

The documentation for this class was generated from the following file:

- **Environment.h**

A.4.117 io::driver::keycode::scancodestates::EOnePhase-OneScanCodeConverterState Class Reference

handles the extended scancodes which are introduced through e1.

Inherits **io::driver::keycode::scancodestates::AbstractScanCodeConverter-State**.

Private Member Functions

- virtual bool **testIfScancodeIsBreakCode** (int scancode) const

  test state specific if a scancode is a breakcode.

- virtual **AbstractScanCodeConverterState** ∗ **handle** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const

  handles the scancode.

Additional Inherited Members

Detailed Description

handles the extended scancodes which are introduced through e1.

The State is divided in two phases. The second phase uses the passed scancode and the next scancode to evaluate the corresponding keycode.

Member Function Documentation

virtual bool io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverterState::testIfScancodeIsBreakCode (int *scancode*) const [private], [virtual] test state specific if a scancode is a breakcode.

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the passed scancode is a breakcode.

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

virtual AbstractScanCodeConverterState∗ io::driver::keycode::scancodestates::EOnePhaseOneScanCodeConverterState::handle (int *scancode,* task-

::pipeandfilter::Pipe< Keycode > & *outPipe*) const [private], [virtual]
handles the scancode.

the case of a breakcode is resolved before in the abstract state.

Parameters

scancode	scancode
&outPipe	outPipe

@

@plus@

@plus -@

@

@

@skip
Returns

the next state

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverter-State** (p. 268).

The documentation for this class was generated from the following file:

- EOnePhaseOneScanCodeConverterState.h

A.4.118 io::driver::keycode::scancodestates::EOnePhase-TwoScanCodeConverterState Class Reference

handles the extended scancodes which are introduced through e1.

Inherits **io::driver::keycode::scancodestates::AbstractScanCodeConverter-State**.

Public Member Functions

- **EOnePhaseTwoScanCodeConverterState** (int paramScancode)

Private Member Functions

- virtual bool **testIfScancodeIsBreakCode** (int scancode) const

  test state specific if a scancode is a breakcode.

- virtual **AbstractScanCodeConverterState** ∗ **handle** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const

  handles the scancode.

Private Attributes

- int **scancodeOld**

Additional Inherited Members

Detailed Description

handles the extended scancodes which are introduced through e1.

The State is divided in two phases. The first passes the scancode two the second phase.

Member Function Documentation

virtual bool io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverterState::testIfScancodeIsBreakCode (int *scancode*) const `[private]`, `[virtual]` test state specific if a scancode is a breakcode.

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the passed scancode is a breakcode.

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

virtual AbstractScanCodeConverterState∗ io::driver::keycode::scancodestates::EOnePhaseTwoScanCodeConverterState::handle (int *scancode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) const [private], [virtual]
handles the scancode.

the case of a breakcode is resolved before in the abstract state.

Parameters

scancode	scancode
&outPipe	outPipe

@

@plus@

@plus -@

@

@

@skip
Returns

the next state

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

The documentation for this class was generated from the following file:

- EOnePhaseTwoScanCodeConverterState.h

A.4.119 runtime::test::ExceptionTestCase Class Reference

Tests exception handling in kernel mode.

Inherits **test::TestCase**.

Public Member Functions

- virtual char const * **getName** ()

Protected Member Functions

- virtual void **run** ()

Private Member Functions

- void **testPrimitiveExceptions** ()

  Tests throwing and catching objects of primitive types.

- void **testClassExceptions** ()

  Tests throwing and catching class objects where the catch handler type exactly matches the object type.

- void **testPolymorphicExceptions** ()

  Tests throwing and catching class objects where the catch handler type is a superclass of the object type.

- void **testMultipleCatchHandlers** ()

  Tests throwing and catching class objects where multiple catch handlers exist such that the best suited has to be chosen.

- void **testCatchAllHandlers** ()

  Tests catch-all handlers.

- void **testUnexpectedExceptions** ()

  Tests unexpected exceptions (and set_unexpected).

Detailed Description

Tests exception handling in kernel mode.

The documentation for this class was generated from the following file:

- ExceptionTestCase.h

A.4.120 api::task::scheduler::ExitThreadRequest Class Reference

Exits the current thread.

Inherits **ipc::Request**.

Public Member Functions

- **ExitThreadRequest** ()

  Constructor.

Static Public Attributes

- static const uint32_t **Id** = 3

  The identifier of this Request.

Additional Inherited Members

Detailed Description

Exits the current thread.

The documentation for this class was generated from the following file:

- **ExitThreadRequest.h**

A.4.121 io::driver::keycode::scancodestates::EZeroScanCodeConverterState Class Reference

handles the extended scancodes(Scancode Set 1) which are introduced through e0.

Inherits **io::driver::keycode::scancodestates::AbstractScanCodeConverterState**.

Private Member Functions

- virtual bool **testIfScancodeIsBreakCode** (int scancode) const

  test state specific if a scancode is a breakcode.

- virtual **AbstractScanCodeConverterState** ∗ **handle** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const

  handles the scancode.

- virtual uint8_t **translateScancode** (int scancode) const

Additional Inherited Members

Detailed Description

handles the extended scancodes(Scancode Set 1) which are introduced through e0.

Member Function Documentation

virtual bool io::driver::keycode::scancodestates::EZeroScanCodeConverterState::testIfScancodeIsBreakCode (int *scancode*) const [private], [virtual] test state specific if a scancode is a breakcode.

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the passed scancode is a breakcode.

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

virtual AbstractScanCodeConverterState∗ io::driver::keycode::scancodestates::EZeroScanCodeConverterState::handle (int *scancode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) const [private], [virtual] handles the scancode.

the case of a breakcode is resolved before in the abstract state.

Parameters

scancode	scancode
&outPipe	outPipe

@

@plus@

@plus -@

@

@

@skip

Returns

the next state

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

virtual uint8_t io::driver::keycode::scancodestates::EZeroScanCodeConverterState::translateScancode (int *scancode*) const [private], [virtual]

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

the keycode which is translated of the scancode

The documentation for this class was generated from the following file:

- EZeroScanCodeConverterState.h

A.4.122 ipc::test::FantasticObject Class Reference

Test interface for IPC.

Inherited by **ipc::test::FantasticObjectImpl**.

Public Member Functions

- virtual ∼**FantasticObject** ()

  Destructor.

- virtual int **getAnswer** ()=0

  Returns 42 ;-)

- virtual int **increment** (int i)=0

  Returns i + 1.

- virtual void **incrementInPlace** (int &i)=0

  Performs i=i+1.

- virtual void **incrementElements** (**Array**< uint32_t > array)=0

  Increments each element in the array once.

- virtual int **add** (int a, int b)=0

  Returns a + b.

- virtual void **print** (**Array**< char const > s)=0

  Prints passed zero-terminated string to the console.

Detailed Description

Test interface for IPC.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.123 ipc::test::FantasticObjectImpl Class Reference

Implementation of the IPC interface.

Inherits **ipc::test::FantasticObject**.

Public Member Functions

- virtual int **getAnswer** ()

  Returns 42 ;-)

- virtual int **increment** (int i)

  Returns i + 1.

- virtual void **incrementInPlace** (int &i)

  Performs i=i+1.

- virtual void **incrementElements** (**Array**< uint32_t > array)

  Increments each element in the array once.

- virtual int **add** (int a, int b)

  Returns a + b.

- virtual void **print** (**Array**< char const > s)

  Prints passed zero-terminated string to the console.

Detailed Description

Implementation of the IPC interface.

The documentation for this class was generated from the following file:

- FantasticObjectImpl.h

A.4.124 ipc::test::FantasticObjectProxy Class Reference

Data Structures

- class **Command**
- class **Command**< **FantasticObjectProxy::Id_add** >
- class **Command**< **FantasticObjectProxy::Id_getAnswer** >
- class **Command**< **FantasticObjectProxy::Id_increment** >

- class **Command**< **FantasticObjectProxy::Id_incrementElements** >
- class **Command**< **FantasticObjectProxy::Id_incrementInPlace** >
- class **Command**< **FantasticObjectProxy::Id_print** >
- class **CommandBase**
- class **Request**
- class **Request**< **0** >
- class **Request**< **FantasticObjectProxy::Id_add** >
- class **Request**< **FantasticObjectProxy::Id_getAnswer** >
- class **Request**< **FantasticObjectProxy::Id_increment** >
- class **Request**< **FantasticObjectProxy::Id_incrementElements** >
- class **Request**< **FantasticObjectProxy::Id_incrementInPlace** >
- class **Request**< **FantasticObjectProxy::Id_print** >
- class **RequestBase**

Public Types

- typedef int **RetType_getAnswer**
- typedef int **RetType_increment**
- typedef int **P1Type_increment**
- typedef int & **P1Type_incrementInPlace**
- typedef **Array**< uint32_t > **P1Type_incrementElements**
- typedef int **RetType_add**
- typedef int **P1Type_add**
- typedef int **P2Type_add**
- typedef **Array**< char const > **P1Type_print**

Public Member Functions

- **FantasticObjectProxy** (::**ipc::CommandRelay** &relay,::**ipc::RemoteObject** const &target)
- ::**ipc::Result**< int > **getAnswer** ()
- ::**ipc::Result**< int > **increment** (int i)
- ::**ipc::Result**< void > **incrementInPlace** (int &i)

- ::**ipc::Result**< void > **incrementElements** (**Array**< uint32_t > array)
- ::**ipc::Result**< int > **add** (int a, int b)
- ::**ipc::Result**< void > **print** (**Array**< char const > s)

Static Public Attributes

- static uint32_t const **NumRequests** = 6
- static uint32_t const **Id_getAnswer** = 1
- static uint32_t const **Id_increment** = 2
- static uint32_t const **Id_incrementInPlace** = 3
- static uint32_t const **Id_incrementElements** = 4
- static uint32_t const **Id_add** = 5
- static uint32_t const **Id_print** = 6

Private Attributes

- ::**ipc::CommandRelay** & **relay**
- ::**ipc::RemoteObject target**

Static Private Attributes

- static **CommandBase** ∗ **commands** [ ]

Data Structure Documentation

class ipc::test::FantasticObjectProxy::Command

template<uint32_t Index>class ipc::test::FantasticObjectProxy::Command< Index >

class ipc::test::FantasticObjectProxy::Request

template<uint32_t Index>class ipc::test::FantasticObjectProxy::Request< Index >

Constructor & Destructor Documentation

ipc::test::FantasticObjectProxy::FantasticObjectProxy (::ipc::CommandRelay & *relay,* ::ipc::RemoteObject const & *target*) [inline]

- Constructor. ∗

  Parameters

relay	∗ The **CommandRelay** (p. 405) to use. ∗
target	∗ The **RemoteObject** (p. 773) to call.

Field Documentation

uint32_t const ipc::test::FantasticObjectProxy::NumRequests = 6 [static]

- The number of requests this proxy can handle.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.125 memory::FarPointer Struct Reference

Struct to access far jumps.

Data Fields

- uint32_t **offset**
- uint16_t **selector**

Detailed Description

Struct to access far jumps.

Field Documentation

uint32_t memory::FarPointer::offset

uint16_t memory::FarPointer::selector The documentation for this struct was generated from the following file:

- FarPointer.h

A.4.126 FAT Class Reference

Represents a **FAT** (p. 497) (File Allocation Table).

Data Structures

- class **FAT12Type**

  Handles FAT12.

- class **FAT16Type**

  Handles FAT16.

- class **FATType**

  FAT *(p. 497) type abstraction.*

Public Member Functions

- **FAT** (**Disk** const ∗**disk**)

  *Creates a **FAT** (p. 497) object.*

- bool **isOK** () const

  *Returns true if the **FAT** (p. 497) has been created successfully.*

- **Disk** const ∗ **getDisk** () const

  *Returns the underlying **Disk** (p. 448) object.*

- bool **getNextCluster** (unsigned &cluster) const

  *Returns the following cluster or zero if the **FAT** (p. 497) object is invalid or if the passed cluster is the last one in its chain.*

- unsigned long **mapClusterToSector** (unsigned long cluster) const

  Maps a cluster to its first logical sector.

Private Member Functions

- bool **load** (unsigned long sector)

  *Loads a **FAT** (p. 497) sector.*

- **FATType** ∗ **determineFATType** (**BPB** const ∗bpb)

  *Determines the type of the **FAT** (p. 497).*

Private Attributes

- **Disk** const ∗const **disk**

  The underlying disk.

- char ∗ **buffer**

  A buffer containing room for two sectors (needed for FAT12).

- unsigned long **firstFATSector**

  *The first sector of the first **FAT** (p. 497).*

- unsigned long **lastSectorLoaded**

  *The last **FAT** (p. 497) sector loaded.*

- **FATType** ∗ **fatType**

  *The **FAT** (p. 497) type.*

- unsigned long **firstDataSector**

  The first sector of the data area.

Detailed Description

Represents a **FAT** (p. 497) (File Allocation Table).

Constructor & Destructor Documentation

FAT::FAT (Disk const ∗ *disk*) Creates a **FAT** (p. 497) object.

Parameters

disk	The underlying disk. The lifetime of the **Disk** (p. 448) object must exceed the lifetime of this **FAT** (p. 497) object.

Member Function Documentation

bool FAT::isOK ( ) const [inline] Returns true if the **FAT** (p. 497) has been created successfully.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the **FAT** (p. 497) object is valid, false otherwise.

References fatType.

Referenced by getNextCluster().

Disk const* FAT::getDisk ( ) const [inline] Returns the underlying **Disk** (p. 448) object.

@

@plus@

@plus -@

@

@

@skip
Returns

The underlying disk object.

References disk.

bool FAT::getNextCluster (unsigned & *cluster*) const [inline] Returns the following cluster or zero if the **FAT** (p. 497) object is invalid or if the passed cluster is the last one in its chain.

Parameters

cluster	A cluster on entry. On exit, it is set to the following cluster in the chain or to zero if this is the last cluster in its chain.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false on error.

References fatType, FAT::FATType::getNextCluster(), and isOK().

unsigned long FAT::mapClusterToSector (unsigned long *cluster*) const [inline] Maps a cluster to its first logical sector.

Parameters

cluster	The cluster to map.

@

@plus@

@plus -@

@

@

@skip

Returns

The first logical sector of the cluster.

References disk, firstDataSector, Disk::getBPB(), and BPB::numSectorsPerCluster.

bool FAT::load (unsigned long *sector*) [private] Loads a **FAT** (p. 497) sector.

Parameters

sector	The **FAT** (p. 497) sector, The first sector of the **FAT** (p. 497) has index zero.

@

@plus@

@plus -@

@

@

@skip

Returns

True if loading the sector succeeded, false otherwise.

FATType∗ FAT::determineFATType (BPB const ∗ *bpb*) [private] Determines the type of the **FAT** (p. 497).

Parameters

bpb	The underlying **BPB** (p. 375).

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to a **FATType** (p. 524) object or null if the type could not be determined.

The documentation for this class was generated from the following file:

- **fat.h**

A.4.127 FAT::FAT12Type Class Reference

Handles FAT12.

Inherits **FAT::FATType**.

Public Member Functions

- **FAT12Type** (**FAT** &**fat**)

  *Creates a **FAT12Type** (p. 502) object.*

- virtual bool **getNextCluster** (unsigned &cluster) const

  Returns the following cluster or zero if passed cluster is the last one in its chain.

Private Attributes

- **FAT** & **fat**

  *The **FAT** (p. 497) object.*

Static Private Attributes

- static const unsigned short **EndOfChain** = 0xFFF

  The end-of-chain marker.

Detailed Description

Handles FAT12.

Constructor & Destructor Documentation

FAT::FAT12Type::FAT12Type (FAT & *fat*) [inline] Creates a **FAT12-Type** (p. 502) object.

Parameters

fat	A reference to the **FAT** (p. 497) object.

Member Function Documentation

virtual bool FAT::FAT12Type::getNextCluster (unsigned & *cluster*) const [virtual] Returns the following cluster or zero if passed cluster is the last one in its chain.

Parameters

cluster	A cluster on entry. On exit, it is set to the following cluster in the chain or to zero if this is the last cluster in its chain.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false on error.

Implements **FAT::FATType** (p. 524).

The documentation for this class was generated from the following file:

- **fat.h**

A.4.128 FAT::FAT16Type Class Reference

Handles FAT16.

Inherits **FAT::FATType**.

Public Member Functions

- **FAT16Type** (**FAT** &**fat**)

  *Creates a **FAT16Type** (p. 504) object.*

- virtual bool **getNextCluster** (unsigned &cluster) const

  Returns the following cluster or zero if passed cluster is the last one in its chain.

Private Attributes

- **FAT** & **fat**

  *The **FAT** (p. 497) object.*

Static Private Attributes

- static const unsigned short **EndOfChain** = 0xFFFF

  The end-of-chain marker.

Detailed Description

Handles FAT16.

Constructor & Destructor Documentation

FAT::FAT16Type::FAT16Type (FAT & *fat*) [inline] Creates a **FAT16-Type** (p. 504) object.

Parameters

fat	A reference to the **FAT** (p. 497) object.

Member Function Documentation

virtual bool FAT::FAT16Type::getNextCluster (unsigned & *cluster*) const [virtual] Returns the following cluster or zero if passed cluster is the last one in its chain.

Parameters

cluster	A cluster on entry. On exit, it is set to the following cluster in the chain or to zero if this is the last cluster in its chain.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false on error.

Implements **FAT::FATType** (p. 524).

The documentation for this class was generated from the following file:

- **fat.h**

A.4.129 io::vfs::fat::FatAccess Class Reference

Access to Hard **Disk** (p. 448) for FAT16.

Public Member Functions

- uint16_t **readCluster** (uint16_t clusterNumber)

  Reads a cluster with the number clusterNumber.

- virtual **driver::block::Block** ∗ **readByte** (uint32_t byte, **BlockDeviceVolume** ∗bDevice)

  Return the Block where the byte with number byte is in.

- virtual uint16_t **calcBytePosition** (uint32_t byte)

  Calculates the position of a byte in a block.

- virtual **Bootsector** ∗ **getBootSector** ()

  *Returns the **Bootsector** (p. 367) from the hard disk.*

Static Public Member Functions

- static **FatAccess** & **instance** ()

  Singleton Pattern.

Private Attributes

- **Bootsector bSector**

Detailed Description

Access to Hard **Disk** (p. 448) for FAT16.

Member Function Documentation

static FatAccess& io::vfs::fat::FatAccess::instance ( ) [static] Singleton Pattern.

@

@plus@

@plus -@

@

@

@skip
Returns

uint16_t io::vfs::fat::FatAccess::readCluster (uint16_t *clusterNumber*) Reads a cluster with the number clusterNumber.

Parameters

clusterNumber	

@

@plus@

@plus -@

@

@

@skip
Returns

virtual driver::block::Block* io::vfs::fat::FatAccess::readByte (uint32_t *byte,* BlockDeviceVolume * *bDevice*) [virtual] Return the Block where the byte with number byte is in.

Use calcBytePosition to get the exact position of the byte in the block.

Parameters

byte	

@

@plus@

@plus -@

@

@

@skip
Returns

virtual uint16_t io::vfs::fat::FatAccess::calcBytePosition (uint32_t *byte*) [virtual] Calculates the position of a byte in a block.

Parameters

byte	

@

@plus@

@plus -@

@

@

@skip
Returns

virtual Bootsector* io::vfs::fat::FatAccess::getBootSector ( ) [virtual]
Returns the **Bootsector** (p. 367) from the hard disk.

@

@plus@

@plus -@

@

@

@skip
Returns

The documentation for this class was generated from the following file:

- FatAccess.h

A.4.130 io::vfs::fat::FatDirectory Class Reference

Inherits **io::vfs::Directory**, and **FatObject**.

Public Member Functions

- **FatDirectory** (**Mount** ∗m)
- virtual const char ∗ **getName** ()
- virtual uint32_t **getSize** ()
- virtual RemoveError::enum_t **removeNode** (const char ∗name)

  Removes the given node from the file system.
- virtual **object::Ref**< **Directory** > **getParent** ()
- virtual **tool::collection::List**
  < **object::Ref**< **FileSystemNode** > > ∗ **getNodes** ()

  Returns a list of all nodes in this directory.
- virtual **object::Ref**
  < **FileSystemNode** > **getNode** (const char ∗name)

  Returns the node with the given name.
- virtual **object::Ref**< **File** > **createFile** (const char ∗name)

  Creates a new file with the given name.
- virtual **object::Ref**< **Directory** > **createDirectory** (const char ∗name)

  Creates a new directory with the given name.

Private Attributes

- ::**tool::collection::List**
  < **DirectoryEntry** > ∗ **dirEntries**

Additional Inherited Members

Member Function Documentation

virtual uint32_t io::vfs::fat::FatDirectory::getSize ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getSize() (p. 536)

Reimplemented from **io::vfs::Directory** (p. 443).

virtual RemoveError::enum_t io::vfs::fat::FatDirectory::removeNode (const char * *name*) [virtual] Removes the given node from the file system.

Only nodes within this directory can be removed. @

@plus@

@plus -@

@

@

@skip
Returns

SUCCESS if the node was removed INUSE if the node is still in use DIRETORYNOTEMPTY if the node is a directory and is not empty NOTFOUND if no node with the given name could be found

Implements **io::vfs::Directory** (p. 443).

virtual object::Ref<Directory> io::vfs::fat::FatDirectory::getParent ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::Directory** (p. 444).

virtual tool::collection::List<object::Ref<FileSystemNode> >* io::vfs::fat::FatDirectory::getNodes ( ) [virtual] Returns a list of all nodes in this directory.

@

@plus@

@plus -@

@

@

@skip
Returns

A list of nodes.

Implements **io::vfs::Directory** (p. 444).

virtual object::Ref<FileSystemNode> io::vfs::fat::FatDirectory::getNode (const char * *name*) [virtual] Returns the node with the given name.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip

Returns

The reference to the node. Null, if no node was found.

Implements **io::vfs::Directory** (p. 445).

virtual object::Ref<File> io::vfs::fat::FatDirectory::createFile (const char * *name*) [virtual] Creates a new file with the given name.

Parameters

name	The name of the new file.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created file. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 446).

virtual object::Ref<Directory> io::vfs::fat::FatDirectory::createDirectory (const char * *name*) [virtual] Creates a new directory with the given name.

Parameters

name	The name of the new directory.

@

@plus@

@plus -@

@

@

@skip

Returns

A reference to the newly created directory. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 447).

The documentation for this class was generated from the following file:

- FatDirectory.h

A.4.131 io::vfs::fat::FatFile Class Reference

Inherits **io::vfs::File**.

Public Member Functions

- **FatFile** (**Mount** ∗m)
- virtual const char ∗ **getName** ()
- virtual uint32_t **getSize** ()
- virtual RemoveError::enum_t **remove** ()
- virtual **object::Ref**< **Directory** > **getParent** ()
- virtual **FileStream** ∗ **open** (ReadWrite::enum_t rw)

  Opens the file and creates a file stream.

Additional Inherited Members

Member Function Documentation

virtual uint32_t io::vfs::fat::FatFile::getSize ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getSize() (p. 536)

Implements **io::vfs::File** (p. 527).

virtual object::Ref<Directory> io::vfs::fat::FatFile::getParent ( ) [virtual]

@

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::File** (p. 527).

virtual FileStream* io::vfs::fat::FatFile::open (ReadWrite::enum_t *rw*) [virtual] Opens the file and creates a file stream.

Parameters

rw	Permission level.

@

@plus@

@plus -@

@

@

@skip

Returns

An open file stream.

Implements **io::vfs::File** (p. 528).

The documentation for this class was generated from the following file:

- FatFile.h

A.4.132 io::vfs::fat::FatFileStream Class Reference

Inherits **io::vfs::FileStream**.

Public Member Functions

- **FatFileStream** (**File** ∗f, ReadWrite::enum_t **rw**)
- virtual uint32_t **seek** (int32_t offset, SeekOrigin::enum_t origin)

  Change the cursor in the stream.
- virtual uint32_t **read** (unsigned char ∗buffer, uint32_t length)

  Reads bytes from the stream.
- virtual void **doWrite** (unsigned char ∗buffer, uint32_t length)

  Writes bytes to the stream.

Additional Inherited Members

Member Function Documentation

virtual uint32_t io::vfs::fat::FatFileStream::seek (int32_t *offset*, SeekOrigin::enum_t *origin*) `[virtual]` Change the cursor in the stream.

Parameters

offset	Count of bytes to move the cursor. Can be positive or negative.
origin	From which position should the cursor be moved.

@

@plus@

@plus -@

@

@

@skip

Returns

The current Position in the stream.

Implements **io::vfs::FileStream** (p. 530).

virtual uint32_t io::vfs::fat::FatFileStream::read (unsigned char * *buffer,* uint32_t *length*) [virtual] Reads bytes from the stream.

Parameters

buffer	This buffer will be filled with read bytes. Should be as large as length
length	The count of bytes to read.

@

@plus@

@plus -@

@

@

@skip

Returns

The number of read bytes. Can be less than length if the end of the file is reached.

Implements **io::vfs::FileStream** (p. 530).

virtual void io::vfs::fat::FatFileStream::doWrite (unsigned char * *buffer*, uint32_t *length*) [virtual] Writes bytes to the stream.

Can only be called if this stream has write permissions.

Parameters

buffer	This buffer will be written to the file.
length	The length of the buffer.

Implements **io::vfs::FileStream** (p. 531).

The documentation for this class was generated from the following file:

- FatFileStream.h

A.4.133 io::vfs::fat::FatFileSystem Class Reference

Inherits **io::vfs::FileSystem**.

Public Member Functions

- virtual bool **canMount** (**Volume** *v)

Determines if this file system can mount the given volume.

- virtual **Mount** ∗ **mount** (**Volume** ∗v)

Member Function Documentation

virtual bool io::vfs::fat::FatFileSystem::canMount (Volume ∗ *v*) [virtual]

Determines if this file system can mount the given volume.

An actual mount is not performed.

Parameters

v	The volume that should be checked

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the volume could be mounted with this file system. Otherwise false.

Implements **io::vfs::FileSystem** (p. 532).

The documentation for this class was generated from the following file:

- FatFileSystem.h

A.4.134 io::vfs::fat::FatMount Class Reference

Inherits **io::vfs::Mount**.

Public Member Functions

- **FatMount** (uint8_t mountId, **Volume** ∗device)
- virtual **Directory** ∗ **getRoot** () const
- virtual **Volume** ∗ **getDevice** ()
- virtual void **doUmount** ()

This method does the actual umounting.

- virtual void **readBootSector** ()

  Reads the bootsector from the hard disk.

- virtual void **readRootDirectory** ()

  Reads the root directory from the disk.

Private Attributes

- **Bootsector bLoader**
- **Volume** ∗ **bDevice**
- **Directory** ∗ **rootDir**

Additional Inherited Members

The documentation for this class was generated from the following file:

- FatMount.h

A.4.135 FatObject Class Reference

Abstraction for FATFile and FATDirectory.

Inherited by **io::vfs::fat::FatDirectory**[private], and **io::vfs::fat::FatRootDirectory**[private].

Detailed Description

Abstraction for FATFile and FATDirectory.

The documentation for this class was generated from the following file:

- FatObject.h

A.4.136 io::vfs::fat::FatRootDirectory Class Reference

Class for the root directory of a FAT16.

Inherits **io::vfs::Directory**, and **FatObject**.

Public Member Functions

- **FatRootDirectory** (::**io::vfs::Mount** ∗m)
- const char ∗ **getName** () const
- uint32_t **getSize** ()
- RemoveError::enum_t **removeNode** (const char ∗name)

  Removes the given node from the file system.
- **object::Ref**< **Directory** > **getParent** ()
- **tool::collection::List**
  < **object::Ref**< **FileSystemNode** > > ∗ **getNodes** ()

  Returns a list of all nodes in this directory.
- **object::Ref**< **FileSystemNode** > **getNode** (const char ∗name)

  Returns the node with the given name.
- **object::Ref**< **File** > **createFile** (const char ∗name)

  Creates a new file with the given name.
- **object::Ref**< **Directory** > **createDirectory** (const char ∗name)

  Creates a new directory with the given name.

Private Attributes

- ::**tool::collection::List**
  < **DirectoryEntry** > ∗ **dirEntries**

Additional Inherited Members

Detailed Description

Class for the root directory of a FAT16.

Member Function Documentation

const char∗ io::vfs::fat::FatRootDirectory::getName ( ) const [virtual]

@

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getName() (p. 536)

Implements **io::vfs::Directory** (p. 442).

uint32_t io::vfs::fat::FatRootDirectory::getSize ( ) [virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getSize() (p. 536)

Reimplemented from **io::vfs::Directory** (p. 443).

RemoveError::enum_t io::vfs::fat::FatRootDirectory::removeNode (const char * *name*) [virtual] Removes the given node from the file system.

Only nodes within this directory can be removed. @

@plus@

@plus -@

@

@

@skip
Returns

SUCCESS if the node was removed INUSE if the node is still in use DIRETORYNOTEMPTY if the node is a directory and is not empty NOTFOUND if no node with the given name could be found

Implements **io::vfs::Directory** (p. 443).

object::Ref<Directory> io::vfs::fat::FatRootDirectory::getParent ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::Directory** (p. 444).

tool::collection::List<object::Ref<FileSystemNode> >* io::vfs::fat::FatRootDirectory::getNodes ( ) [virtual] Returns a list of all nodes in this directory.

@

@plus@

@plus -@

@

@

@skip

Returns

A list of nodes.

Implements **io::vfs::Directory** (p. 444).

object::Ref< FileSystemNode> io::vfs::fat::FatRootDirectory::getNode (const char * *name*) [virtual] Returns the node with the given name.

Parameters

name	The name to search for.

@

@plus@

@plus -@

@

@

@skip
Returns

The reference to the node. Null, if no node was found.

Implements **io::vfs::Directory** (p. 445).

object::Ref< File> io::vfs::fat::FatRootDirectory::createFile (const char * *name*) [virtual] Creates a new file with the given name.

Parameters

name	The name of the new file.

@

@plus@

@plus -@

@

@

@skip
Returns

A reference to the newly created file. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 446).

object::Ref< Directory> io::vfs::fat::FatRootDirectory::createDirectory (const char * *name*) [virtual] Creates a new directory with the given name.

Parameters

name	The name of the new directory.

@

@plus@

@plus -@

@

@

@skip

Returns

> A reference to the newly created directory. Null, if a node with the given name is already present.

Implements **io::vfs::Directory** (p. 447).

The documentation for this class was generated from the following file:

- FatRootDirectory.h

A.4.137 FAT::FATType Class Reference

FAT (p. 497) type abstraction.

Inherited by **FAT::FAT12Type**, and **FAT::FAT16Type**.

Public Member Functions

- virtual bool **getNextCluster** (unsigned &cluster) const =0

  Returns the following cluster or zero if passed cluster is the last one in its chain.

Detailed Description

FAT (p. 497) type abstraction.

Member Function Documentation

virtual bool FAT::FATType::getNextCluster (unsigned & *cluster*) const `[pure virtual]` Returns the following cluster or zero if passed cluster is the last one in its chain.

Parameters

cluster	A cluster on entry. On exit, it is set to the following cluster in the chain or to zero if this is the last cluster in its chain.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false on error.

Implemented in **FAT::FAT16Type** (p. 505), and **FAT::FAT12Type** (p. 503).

Referenced by FAT::getNextCluster().

The documentation for this class was generated from the following file:

- **fat.h**

A.4.138 io::vfs::File Class Reference

Abstract file.

Inherits **io::vfs::FileSystemNode**.

Inherited by **io::vfs::fat::FatFile**, and **io::vfs::test::DummyFile**.

Public Member Functions

- **File** (**Mount** ∗m)
- virtual void **accept** (**FileSystemNodeVisitor** ∗v)
- virtual bool **isFile** ()
- virtual const char ∗ **getName** () const =0
- virtual uint32_t **getSize** ()=0
- virtual **object::Ref**< **Directory** > **getParent** ()=0
- virtual **FileStream** ∗ **open** (ReadWrite::enum_t rw)=0

  Opens the file and creates a file stream.

Additional Inherited Members

Detailed Description

Abstract file.

Constructor & Destructor Documentation

io::vfs::File::File (Mount * *m*) @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::FileSystemNode() (p. 535)

Member Function Documentation

virtual void io::vfs::File::accept (FileSystemNodeVisitor * *v*) [virtual]

@

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::accept() (p. 535)

Implements **io::vfs::FileSystemNode** (p. 535).

virtual bool io::vfs::File::isFile ( ) [virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::isFile() (p. 535)

Implements **io::vfs::FileSystemNode** (p. 535).

virtual const char* io::vfs::File::getName ( ) const [pure virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getName() (p. 536)

Implements **io::vfs::FileSystemNode** (p. 536).

Implemented in **io::vfs::test::DummyFile** (p. 468).

virtual uint32_t io::vfs::File::getSize ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
See Also

FileSystemNode::getSize() (p. 536)

Implements **io::vfs::FileSystemNode** (p. 536).

Implemented in **io::vfs::fat::FatFile** (p. 513), and **io::vfs::test::DummyFile** (p. 468).

virtual object::Ref<Directory> io::vfs::File::getParent ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip

See Also

FileSystemNode::getParent() (p. 537)

Implements **io::vfs::FileSystemNode** (p. 537).

Implemented in **io::vfs::fat::FatFile** (p. 514), and **io::vfs::test::DummyFile** (p. 468).

virtual FileStream* io::vfs::File::open (ReadWrite::enum_t *rw*) [pure virtual] Opens the file and creates a file stream.

Parameters

rw	Permission level.

@

@plus@

@plus -@

@

@

@skip

Returns

An open file stream.

Implemented in **io::vfs::fat::FatFile** (p. 514), and **io::vfs::test::DummyFile** (p. 469).

The documentation for this class was generated from the following file:

- File.h

A.4.139 io::vfs::FileStream Class Reference

Read and write files.

Inherited by **io::vfs::fat::FatFileStream**, and **io::vfs::test::DummyFileStream**.

Public Member Functions

- **FileStream** (**object::Ref**< **File** > f, ReadWrite::enum_t **rw**)

  Constructor.
- virtual uint32_t **seek** (int32_t offset, SeekOrigin::enum_t origin)=0

  Change the cursor in the stream.
- virtual uint32_t **read** (unsigned char *buffer, uint32_t length)=0

  Reads bytes from the stream.
- virtual void **write** (unsigned char *buffer, uint32_t length)

  Writes bytes to the stream.

Protected Member Functions

- virtual **object::Ref**< **File** > **getFile** () const

  Returns the underlying file.
- virtual void **doWrite** (unsigned char *buffer, uint32_t length)=0

  Writes bytes to the stream.

Private Attributes

- ReadWrite::enum_t **rw**

  Read write permission.
- **object::Ref**< **File** > **file**

  underlying file

Detailed Description

Read and write files.

Constructor & Destructor Documentation

io::vfs::FileStream::FileStream (object::Ref< **File** > ***f*, ReadWrite::enum_t *rw*)** Constructor.

Parameters

f	Opened file.
rw	Read/write permission.

Member Function Documentation

virtual uint32_t io::vfs::FileStream::seek (int32_t *offset*, SeekOrigin::enum_t *origin*) [pure virtual] Change the cursor in the stream.

Parameters

offset	Count of bytes to move the cursor. Can be positive or negative.
origin	From which position should the cursor be moved.

@

@plus@

@plus -@

@

@

@skip

Returns

The current Position in the stream.

Implemented in **io::vfs::fat::FatFileStream** (p. 515), and **io::vfs::test::DummyFileStream** (p. 470).

virtual uint32_t io::vfs::FileStream::read (unsigned char * *buffer*, uint32_t *length*) [pure virtual] Reads bytes from the stream.

Parameters

buffer	This buffer will be filled with read bytes. Should be as large as length
length	The count of bytes to read.

@

@plus@

@plus -@

@

@

@skip
Returns

The number of read bytes. Can be less than length if the end of the file is reached.

Implemented in **io::vfs::fat::FatFileStream** (p. 516), and **io::vfs::test::DummyFileStream** (p. 471).

virtual void io::vfs::FileStream::write (unsigned char * *buffer*, uint32_t *length*) [virtual] Writes bytes to the stream.

Can only be called if this stream has write permissions.

Parameters

buffer	This buffer will be written to the file.
length	The length of the buffer.

virtual object::Ref<File> io::vfs::FileStream::getFile ( ) const [inline], [protected], [virtual] Returns the underlying file.

@

@plus@

@plus -@

@

@

@skip
Returns

The file.

References file.

virtual void io::vfs::FileStream::doWrite (unsigned char * *buffer*, uint32_t *length*) [protected], [pure virtual] Writes bytes to the stream.

Can only be called if this stream has write permissions.

Parameters

buffer	This buffer will be written to the file.
length	The length of the buffer.

Implemented in **io::vfs::fat::FatFileStream** (p. 517), and **io::vfs::test::DummyFileStream** (p. 472).

The documentation for this class was generated from the following file:

- FileStream.h

A.4.140 io::vfs::FileSystem Class Reference

The abstract representation of a file system.

Inherited by **io::vfs::fat::FatFileSystem**, and **io::vfs::test::DummyFileSystem**.

Public Member Functions

- virtual bool **canMount** (**Volume** ∗v)=0

  Determines if this file system can mount the given volume.

- virtual **Mount** ∗ **mount** (uint8_t mountID, **Volume** ∗v)=0

  Mounts the given volume.

Detailed Description

The abstract representation of a file system.

All methods should only be called by the vfs manager.

Member Function Documentation

virtual bool io::vfs::FileSystem::canMount (Volume ∗ *v*) `[pure virtual]`

Determines if this file system can mount the given volume.

An actual mount is not performed.

Parameters

v	The volume that should be checked

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the volume could be mounted with this file system. Otherwise false.

Implemented in **io::vfs::test::DummyFileSystem** (p. 473), and **io::vfs::fat::FatFileSystem** (p. 518).

virtual Mount* io::vfs::FileSystem::mount (uint8_t *mountID,* Volume * *v*) `[pure virtual]` Mounts the given volume.

Parameters

v	The volume to mount

@

@plus@

@plus -@

@

@

@skip

Returns

The created mount

Implemented in **io::vfs::test::DummyFileSystem** (p. 473).

The documentation for this class was generated from the following file:

- FileSystem.h

A.4.141 io::vfs::FileSystemNode Class Reference

The abstract representation of a file system node.

Inherits **object::RefCountedObject**.

Inherited by **io::vfs::Directory**, and **io::vfs::File**.

Public Member Functions

- **FileSystemNode** (**Mount** ∗m)

  Creates a new file system node with the given mount as parent.
- virtual void **accept** (**FileSystemNodeVisitor** ∗v)=0

  Accepts a visitor.
- virtual bool **isFile** ()=0

  Determines, if this node is a file.
- virtual bool **isDirectory** ()

  Determines, if this node is a directory.
- virtual const char ∗ **getName** () const =0

  Returns the file/directory name of this node.
- virtual uint32_t **getSize** ()=0

  Returns the size of this file/directory in bytes.
- virtual **object::Ref**< **Directory** > **getParent** ()=0

  Returns the parent directory.

Protected Member Functions

- virtual **Mount** ∗ **getMount** () const

  Returns the mount, to which this node belongs.

Private Attributes

- **Mount** ∗ **mount**

  the reference to the original mount

Detailed Description

The abstract representation of a file system node.

Constructor & Destructor Documentation

io::vfs::FileSystemNode::FileSystemNode (Mount * *m*) Creates a new file system node with the given mount as parent.

Class is abstract, so this constructor can not be called directly.

Parameters

m	The original mount

Member Function Documentation

virtual void io::vfs::FileSystemNode::accept (FileSystemNodeVisitor * *v*) `[pure virtual]` Accepts a visitor.

See visitor pattern.

Parameters

v	The visitor

Implemented in **io::vfs::File** (p. 526), and **io::vfs::Directory** (p. 442).

virtual bool io::vfs::FileSystemNode::isFile ( ) `[pure virtual]` Determines, if this node is a file.

@

@plus@

@plus -@

@

@

@skip

Returns

true, if this node is a file. false otherwise.

Implemented in **io::vfs::File** (p. 526), and **io::vfs::Directory** (p. 442).

virtual bool io::vfs::FileSystemNode::isDirectory ( ) [virtual] Determines, if this node is a directory.

@

@plus@

@plus -@

@

@

@skip
Returns

true, if this node is a directory. false otherwise.

virtual const char* io::vfs::FileSystemNode::getName ( ) const [pure virtual] Returns the file/directory name of this node.

@

@plus@

@plus -@

@

@

@skip
Returns

The name of this node

Implemented in **io::vfs::File** (p. 527), **io::vfs::Directory** (p. 442), **io::vfs::fat::FatRootDirectory** (p. 520), **io::vfs::test::DummyFile** (p. 468), and **io::vfs::test::DummyDirectory** (p. 463).

virtual uint32_t io::vfs::FileSystemNode::getSize ( ) [pure virtual] Returns the size of this file/directory in bytes.

If this node is a directory, the size of all sub directories and files is summed. @

@plus@

@plus -@

@

@

@skip
Returns

The size in bytes.

Implemented in **io::vfs::File** (p. 527), **io::vfs::Directory** (p. 443), **io::vfs::fat::FatRootDirectory** (p. 521), **io::vfs::fat::FatDirectory** (p. 510), **io::vfs::fat::FatFile** (p. 513), and **io::vfs::test::DummyFile** (p. 468).

virtual object::Ref<Directory> io::vfs::FileSystemNode::getParent ( ) [pure virtual] Returns the parent directory.

If this node is the root directory of a mount, NULL is returned. @

@plus@

@plus -@

@

@

@skip
Returns

A reference to the parent directory or NULL

Implemented in **io::vfs::Directory** (p. 444), **io::vfs::File** (p. 527), **io::vfs::fat::FatRootDirectory** (p. 522), **io::vfs::fat::FatDirectory** (p. 510), **io::vfs::fat::FatFile** (p. 514), **io::vfs::test::DummyFile** (p. 468), and **io::vfs::test::DummyDirectory** (p. 464).

virtual Mount* io::vfs::FileSystemNode::getMount ( ) const [inline], [protected], [virtual] Returns the mount, to which this node belongs.

@

@plus@

@plus -@

@

@

@skip

Returns

The original mount

References mount.

The documentation for this class was generated from the following file:

- FileSystemNode.h

A.4.142 io::vfs::FileSystemNodeVisitor Class Reference

Visitor for file system nodes.

Public Member Functions

- virtual void **visit** (**object::Ref**< **File** > f)=0

  Visiting a file.

- virtual void **visit** (**object::Ref**< **Directory** > d)=0

  Visiting a directory.

Detailed Description

Visitor for file system nodes.

See visitor pattern.

Member Function Documentation

virtual void io::vfs::FileSystemNodeVisitor::visit (object::Ref< File > *f*) `[pure virtual]` Visiting a file.

Parameters

f	Reference to the file

virtual void io::vfs::FileSystemNodeVisitor::visit (object::Ref< Directory > *d*) `[pure virtual]` Visiting a directory.

Parameters

d	Reference to the directory

The documentation for this class was generated from the following file:

- FileSystemNodeVisitor.h

A.4.143 task::pipeandfilter::FilterThread< T, V > Class Template Reference

Inherits **task::Thread**.

Public Member Functions

- **FilterThread** (**task::Process** &parent, **Pipe**< T > &paramInPipe, **Pipe**< V > &paramOutPipe, **AbstractFilter**< T, V > *paramfilter, ThreadPriority priority=PRIO_NORMAL)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **AbstractFilter**< T, V > * **filter**
- **Pipe**< T > & **inPipe**
- **Pipe**< V > & **outPipe**

Additional Inherited Members

Member Function Documentation

template<typename T , typename V > void task::pipeandfilter::FilterThread< T, V >::run ( ) `[virtual]` IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- FilterThread.h

A.4.144 fosCli::FosCli Class Reference

FHDW-OS Super Command Line Interpreter.

Public Member Functions

- void **start** (**ipc::CommandRelay** &relay)

  Starting all required services for the FHDW-OS super command line interpreter.
- **scanner::FosCliScanner** ∗ **getScanner** ()

Static Public Member Functions

- static **FosCli** & **getInstance** ()

Private Member Functions

- void **run** ()

  Starting scanning and parsing the input from the keyboard.

Private Attributes

- **ipc::CommandRelay** ∗ **relay**
- **scanner::FosCliScanner** ∗ **scanner**

Static Private Attributes

- static **FosCli instance**

Detailed Description

FHDW-OS Super Command Line Interpreter.

Member Function Documentation

static FosCli& fosCli::FosCli::getInstance ( ) [inline], [static] @

@plus@

@plus -@

@

@

@skip
Returns

Returns the singleton instance of the **FosCli** (p. 540)

scanner::FosCliScanner* fosCli::FosCli::getScanner ( ) @

@plus@

@plus -@

@

@

@skip
Returns

Returns the scanner of the **FosCli** (p. 540)

The documentation for this class was generated from the following file:

- FosCli.h

A.4.145 fosCli::scanner::elements::FosCliElement Class Reference

Inherited by **fosCli::scanner::elements::FosCliStringElement**.

Public Member Functions

- virtual void **accept** (**FosCliElementVisitor** *visitor)=0

  Accept method for visitor-pattern.

Member Function Documentation

virtual void fosCli::scanner::elements::FosCliElement::accept (FosCliElementVisitor * *visitor*) [pure virtual] Accept method for visitor-pattern.

Parameters

visitor	The visitor which should visit this object.

Implemented in **fosCli::scanner::elements::FosCliStringElement** (p. 550).

The documentation for this class was generated from the following file:

- FosCliElement.h

A.4.146 fosCli::scanner::elements::FosCliElementVisitor Class Reference

Visitor interface for FosCliElements.

Inherited by **fosCli::parser::Parser** [private].

Public Member Functions

- virtual void **handle** (**FosCliStringElement** *element)=0

Detailed Description

Visitor interface for FosCliElements.

The documentation for this class was generated from the following file:

- FosCliElementVisitor.h

A.4.147 fosCli::scanner::states::FosCliExecuteScannerState Class Reference

Inherits **fosCli::scanner::FosCliScannerState**.

Public Member Functions

- **FosCliExecuteScannerState** (**tool::collection::List**< **elements::FosCliElement** * > ***elementList**)

Protected Member Functions

- virtual **FosCliScannerStateResult scanInternal** (char character)

  Implementation of abstract method from.

Member Function Documentation

virtual FosCliScannerStateResult fosCli::scanner::states::FosCliExecuteScannerState::scanInternal (char *character*) [protected], [virtual]
Implementation of abstract method from.

@

@plus@

@plus -@

@

@

@skip
See Also

fosCli::scanner::FosCliScannerState (p. 545)

Implements **fosCli::scanner::FosCliScannerState** (p. 547).

The documentation for this class was generated from the following file:

- FosCliExecuteScannerState.h

A.4.148 fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState Class Reference

Inherits **fosCli::scanner::FosCliScannerState**.

Public Member Functions

- **FosCliReadSpaceDemelitedSignScannerState** (**tool::collection::List**< **elements::FosCliElement** ∗ > ∗**elementList**)

Protected Member Functions

- virtual **FosCliScannerStateResult scanInternal** (char character)

  Implementation of abstract method from.

Private Attributes

- char const ∗ **string**

Member Function Documentation

virtual FosCliScannerStateResult fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState::scanInternal (char *character*) [protected], [virtual] Implementation of abstract method from.

@

@plus@

@plus -@

@

@

@skip

See Also

fosCli::scanner::FosCliScannerState (p. 545)

Implements **fosCli::scanner::FosCliScannerState** (p. 547).

The documentation for this class was generated from the following file:

- FosCliReadSpaceDemelitedSignScannerState.h

A.4.149 fosCli::scanner::FosCliScanner Class Reference

Scanning the input of the keyboard.

Public Member Functions

- char const ∗ **scan** (char character)

  Scanning the input of the keyboard and sends it to the parser if the line was executed.

Private Attributes

- **FosCliScannerState** ∗ **state**
- **tool::collection::List**
  < **elements::FosCliElement** ∗ > ∗ **elementList**

Detailed Description

Scanning the input of the keyboard.

Member Function Documentation

char const∗ fosCli::scanner::FosCliScanner::scan (char *character*)
Scanning the input of the keyboard and sends it to the parser if the line was executed.

Parameters

character	the sign which should be scanned

The documentation for this class was generated from the following file:

- FosCliScanner.h

A.4.150 fosCli::scanner::FosCliScannerState Class Reference

A state of the scanner.

Inherited by **fosCli::scanner::states::FosCliExecuteScannerState**, and **fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState**.

Public Member Functions

- **FosCliScannerState** (**tool::collection::List**< **elements::FosCliElement** ∗ > ∗**elementList**)

  Constructor.

- **FosCliScannerStateResult scan** (char character)

  The corresponding scanning function for this state.

Protected Member Functions

- virtual **FosCliScannerStateResult scanInternal** (char character)=0

  Internal abstract scan method of template method pattern.

- **tool::collection::List** < **elements::FosCliElement** ∗ > ∗ **getElementList** ()

Private Attributes

- **tool::collection::List** < **elements::FosCliElement** ∗ > ∗ **elementList**

  The list which contains the scanned not parsed elements.

Detailed Description

A state of the scanner.

Constructor & Destructor Documentation

fosCli::scanner::FosCliScannerState::FosCliScannerState (tool::collection-::List< **elements::FosCliElement** ∗ > ∗ ***elementList*** **)** Constructor.

Parameters

elementList	The list which contains the scanned not parsed elements

Member Function Documentation

FosCliScannerStateResult fosCli::scanner::FosCliScannerState::scan (char *character*) The corresponding scanning function for this state.

Parameters

character	The character which should be scanned

@

@plus@

@plus -@

@

@

@skip

Returns

The next state and a message which should be printed

virtual FosCliScannerStateResult fosCli::scanner::FosCliScannerState::scanInternal (char *character*) [protected], [pure virtual] Internal abstract scan method of template method pattern.

Template method is @

@plus@

@plus -@

@

@

@skip

See Also

scan (p. 546)

Implemented in **fosCli::scanner::states::FosCliExecuteScannerState** (p. 543), and **fosCli::scanner::states::FosCliReadSpaceDemelitedSignScannerState** (p. 544).

tool::collection::List<elements::FosCliElement*>* fosCli::scanner::FosCliScannerState::getElementList ( ) [protected] @

@plus@

@plus -@

@

@

@skip

Returns

Returns the list which contains the scanned not parsed elements

The documentation for this class was generated from the following file:

- FosCliScannerState.h

A.4.151 fosCli::scanner::FosCliScannerStateResult Class Reference

Public Member Functions

- **FosCliScannerStateResult** (**FosCliScannerState** ∗nextState, char const ∗consoleOutput)
- char const ∗ **getConsoleOutput** ()
- **FosCliScannerState** ∗ **getNextState** ()

Private Attributes

- **FosCliScannerState** ∗ **nextState**
- char const ∗const **consoleOutput**

The documentation for this class was generated from the following file:

- FosCliScannerStateResult.h

A.4.152 fosCli::scanner::elements::FosCliStringElement Class Reference

Inherits **fosCli::scanner::elements::FosCliElement**.

Public Member Functions

- **FosCliStringElement** ()

  Default Constructor with empty string.

- **FosCliStringElement** (char const ∗startString)

Constructor which sets the string to.

- char const ∗ **getString** ()

  getter for the string value

- virtual ∼**FosCliStringElement** ()

  Destructor.

- virtual void **accept** (**FosCliElementVisitor** ∗visitor)

  Implementation of abstract method.

Private Attributes

- char const ∗ **string**

Constructor & Destructor Documentation

fosCli::scanner::elements::FosCliStringElement::FosCliStringElement (char const ∗ *startString*) Constructor which sets the string to.

Parameters

startString	

Member Function Documentation

char const∗ fosCli::scanner::elements::FosCliStringElement::getString ( ) [inline] getter for the string value

@

@plus@

@plus -@

@

@

@skip

Returns

string

virtual void fosCli::scanner::elements::FosCliStringElement::accept (FosCliElementVisitor * *visitor*) `[virtual]` Implementation of abstract method.

@

@plus@

@plus -@

@

@

@skip
See Also

fosCli::scanner::elements::FosCliElement (p. 541)

Implements **fosCli::scanner::elements::FosCliElement** (p. 542).

The documentation for this class was generated from the following file:

- FosCliStringElement.h

A.4.153 api::memory::lmm::FreeBlockRequest Class Reference

A **FreeBlockRequest** (p. 550) represents the desire of freeing an amount of memory.

Inherits **ipc::Request**.

Public Member Functions

- **FreeBlockRequest** (void *__startAddr__, uint32_t **sizeInPages**)

  Constructor.

- uint32_t **getSizeInPages** () const

  Getter for sizeInPages.

- void * **getStartAddr** () const

  Getter for startAddr.

Private Attributes

- uint32_t const **sizeInPages**

  Amount of pages that shall be freed.

- void * **startAddr**

  Start address of the be freed memory space.

Static Private Attributes

- static const uint32_t **Id** = 10

  Identifier.

Additional Inherited Members

Detailed Description

A **FreeBlockRequest** (p. 550) represents the desire of freeing an amount of memory.

Constructor & Destructor Documentation

api::memory::lmm::FreeBlockRequest::FreeBlockRequest (void * *startAddr*, uint32_t *sizeInPages*) `[inline]` Constructor.

Parameters

startAddr	is the start address of the memory block, that shall be freed.
sizeInPages	is the amount of pages that belongs to the memory

Member Function Documentation

uint32_t api::memory::lmm::FreeBlockRequest::getSizeInPages ( ) const `[inline]` Getter for sizeInPages.

@

@plus@

@plus -@

@

@

@skip
Returns

the amount of pages that shall be freed

References sizeInPages.

void* api::memory::lmm::FreeBlockRequest::getStartAddr ( ) const [inline] Getter for startAddr.

@

@plus@

@plus -@

@

@

@skip
Returns

the start address of the to be freed memory space.

References startAddr.

The documentation for this class was generated from the following file:

- FreeBlockRequest.h

A.4.154 memory::GateDescriptor Class Reference

Describes a gate descriptor.

Inherits **memory::SystemSegmentDescriptor**.

Public Member Functions

- uint32_t **getOffset** () const

  Returns the gate's offset.

- **GateDescriptor** & **setOffset** (uint32_t offset)

  Sets the gate's offset.

- **Selector getSelector** () const

  Returns the gate's selector.

- **GateDescriptor** & **setSelector** (**Selector** selector)

  Sets the gate's selector.

- uint32_t **getParamCount** () const

  Returns the gate's parameter count.

- **GateDescriptor** & **setParamCount** (uint32_t paramCount)

  Sets the gate's parameter count.

Private Types

- typedef **tool::BitField**
  < uint32_t, 0, 0, 16 > **OffsetLowField**

  Describes the bits 0..15 of the offset field.

- typedef **tool::BitField**
  < uint32_t, 0, 16, 16 > **SelectorField**

  Describes the selector field.

- typedef **tool::BitField**
  < uint32_t, 1, 0, 5 > **ParamCountField**

  Describes the parameter count field.

- typedef **tool::BitField**
  < uint32_t, 1, 16, 16 > **OffsetHighField**

  Describes the bits 16..31 of the offset field.

Additional Inherited Members

Detailed Description

Describes a gate descriptor.

Member Function Documentation

GateDescriptor& memory::GateDescriptor::setOffset (uint32_t *offset*) [inline] Sets the gate's offset.

Parameters

offset	The new offset.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

GateDescriptor& memory::GateDescriptor::setSelector (Selector *selector*) [inline] Sets the gate's selector.

Parameters

selector	The new selector.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this

References memory::Descriptor::data, memory::Selector::getValue(), and tool::BitField< Base, Index, BitPosition, Length >::set().

uint32_t memory::GateDescriptor::getParamCount ( ) const [inline] Returns the gate's parameter count.

Only used by call gates.

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::get().

GateDescriptor& memory::GateDescriptor::setParamCount (uint32_t *paramCount*) `[inline]` Sets the gate's parameter count.

At most 31 32-bit words can be copied. Only used by call gates.

Parameters

paramCount	The new parameter count.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References memory::Descriptor::data, and tool::BitField< Base, Index, BitPosition, Length >::set().

The documentation for this class was generated from the following file:

- Descriptor.h

A.4.155 cpu::GDTManager Class Reference

<<singleton>> Class for managing the Global **Descriptor** (p. **??**) Table (GDT)

Public Member Functions

- **memory::Selector addDescriptor** (**memory::Descriptor** const &desc)

  *Inserts passed **Descriptor** (p. **??**) into the next free slot in the GDT and returns its Selector with RPL = 0.*

- **memory::Selector addTSS** (**task::TaskStateSegment** *tss)

*Inserts a TSS **Descriptor** (p. **??**) into the GDT which is based on passed TaskStateSegment and returns an appropriate Selector.*

- void **free** (**memory::Selector** selector)

  *Sets the Present flag of the **Descriptor** (p. **??**) the Selector passed points to to zero.*

- **memory::Selector getKernelCodeSegementSelector** ()

  Returns the kernel code segment selector.

- **memory::Selector getKernelDataSegementSelector** ()

  Returns the kernel data and stack segment selector.

- **memory::Selector getUserCodeSegementSelector** ()

  Returns the user code segment selector.

- **memory::Selector getUserDataSegementSelector** ()

  Returns the user data and stack segment selector.

- **memory::Selector getTLSSelector** ()

  Returns the selector pointing to the current TCB.

- **memory::Descriptor** & **getDescriptor** (**memory::Selector** sel)

  *Inserts passed **Descriptor** (p. **??**) into the next free slot in the GDT and returns its Selector with RPL = 0.*

- **memory::Descriptor** const & **getDescriptor** (**memory::Selector** sel) const

Static Public Member Functions

- static **GDTManager** & **getInstance** ()

  Returns the singleton instance of this class.

Private Member Functions

- **GDTManager** ()

  Constructor. Initializes and loads a new GDT.

- void **initGDT** ()

  Builds up the initial GDT (copy from loader)

- void **loadGDT** ()

Sets the GDT active (assembly)

- **memory::Descriptor** ∗ **findNextFree** ()

  Finds the next free slot in the GDT.

Static Private Member Functions

- static void **init** ()

  Initializes the singleton instance of this class.

Private Attributes

- **memory::Descriptor gdt** [**NumEntries**]

  The GDT entries.

- **memory::Descriptor** ∗ **lastUsed**

  Points to the last descriptor used in the GDT (for speedup of search for next free selector).

- **memory::Selector kernelCodeSegementSelector**

  Selector for the kernel code segment selector.

- **memory::Selector kernelDataSegementSelector**

  Selector for the kernel data and stack segment selector.

- **memory::Selector userCodeSegementSelector**

  Selector for the user code segment.

- **memory::Selector userDataSegementSelector**

  Selector for the user data and stack segment.

- **memory::Selector tlsSelector**

  Selector pointing to the current TCB.

Static Private Attributes

- static uint32_t const **NumEntries** = 8192

  The number of entries in the GDT.

- static **GDTManager instance**

  Instance for the Singleton Pattern.

Friends

- class **boot::BootManager**

Detailed Description

<<singleton>> Class for managing the Global **Descriptor** (p. **??**) Table (GDT)

Member Function Documentation

memory::Selector cpu::GDTManager::addDescriptor (memory::Descriptor const & *desc*) Inserts passed **Descriptor** (p. **??**) into the next free slot in the GDT and returns its Selector with RPL = 0.

If no free slot is available, a NULL Selector is returned.

memory::Descriptor& cpu::GDTManager::getDescriptor (memory::Selector *sel*) `[inline]` Inserts passed **Descriptor** (p. **??**) into the next free slot in the GDT and returns its Selector with RPL = 0.

If no free slot is available, a NULL Selector is returned.

References gdt, and memory::Selector::getIndex().

memory::Selector cpu::GDTManager::addTSS (task::TaskStateSegment * *tss*) Inserts a TSS **Descriptor** (p. **??**) into the GDT which is based on passed TaskStateSegment and returns an appropriate Selector.

If the GDT is full, a NULL Selector is returned.

Parameters

tss	The TaskStateSegment for which to add a TSS descriptor.

static void cpu::GDTManager::init ( ) `[inline]`, `[static]`, `[private]` Initializes the singleton instance of this class.

@

@plus@

@plus -@

@

@

@skip
Returns

References GDTManager(), and instance.

memory::Descriptor* cpu::GDTManager::findNextFree ( ) [private]
Finds the next free slot in the GDT.

@

@plus@

@plus -@

@

@

@skip
Returns

Returns the pointer of the next free slot or NULL if no free slot found.

The documentation for this class was generated from the following file:

- GDTManager.h

A.4.156 api::kernel::GetVersionRequest Class Reference

Determines the kernel version.

Inherits **ipc::Request**.

Public Member Functions

- uint32_t **getVersion** () const

  Returns the requested kernel version.

- void **setVersion** (uint32_t **version**)

  Sets the requested kernel version.

- **GetVersionRequest** ()

  Constructor.

Static Public Attributes

- static const uint32_t **Id** = 0

  The identifier of this Request.

Private Attributes

- uint32_t **version**

  The version (output).

Additional Inherited Members

Detailed Description

Determines the kernel version.

The documentation for this class was generated from the following file:

- **GetVersionRequest.h**

A.4.157 fosCli::commands::HelpCliCommand Class Reference

Inherits **fosCli::parser::CliCommand**.

Public Member Functions

- virtual void **execute** ()
- virtual void **addParameter** (char const ∗parameter)

The documentation for this class was generated from the following file:

- HelpCliCommand.h

A.4.158 fosCli::commands::HelpCliCommandCreator Class Reference

Inherits **fosCli::parser::CliCommandCreator**.

Public Member Functions

- virtual **parser::CliCommand** ∗ **create** ()

The documentation for this class was generated from the following file:

- HelpCliCommandCreator.h

A.4.159 task::priorityinheritance::test::HighPrioThread Class Reference

A high prioritized thread that wants to access a resource that a lower prioritized thread has locked exclusively.

Inherits **task::Thread**.

Public Member Functions

- **HighPrioThread** (**Process** &**process**, **lock::Semaphore** &**sem**, **lock::-Semaphore** &**exit**)

  Constructor.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **lock::Semaphore** & **sem**

  The semaphore that protects some unspecified resource.

- **lock::Semaphore** & **exit**

The exit semaphore released on thread exit.

Detailed Description

A high prioritized thread that wants to access a resource that a lower prioritized thread has locked exclusively.

Constructor & Destructor Documentation

task::priorityinheritance::test::HighPrioThread::HighPrioThread (Process & *process,* lock::Semaphore & *sem,* lock::Semaphore & *exit*) Constructor.

Parameters

process	The process this thread should attach to.
sem	The semaphore that protects some unspecified resource.
exit	The exit semaphore.

Member Function Documentation

virtual void task::priorityinheritance::test::HighPrioThread::run ( ) `[protected]`**,** `[virtual]` IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **HighPrioThread.h**

A.4.160 task::scheduler::IdleThread Class Reference

This **Thread** (p. 865) runs if no other **Thread** (p. 865) is runnable.

Inherits **task::Thread**.

Protected Member Functions

- virtual void **run** ()

  Halts the processor until an interrupt arrives.

Private Member Functions

- **IdleThread** ()

  Constructor.

Friends

- class **Scheduler**

Additional Inherited Members

Detailed Description

This **Thread** (p. 865) runs if no other **Thread** (p. 865) is runnable.

The documentation for this class was generated from the following file:

- IdleThread.h

A.4.161 io::driver::keycode::scancodestates::InitialScanCodeConverterState Class Reference

standard state handles the most keys like letters and digits and the most other characters (scancode set 1)

Inherits **io::driver::keycode::scancodestates::AbstractScanCodeConverterState**.

Private Member Functions

- virtual bool **testIfScancodeIsBreakCode** (int scancode) const

  test state specific if a scancode is a breakcode.

- virtual **AbstractScanCodeConverterState** ∗ **handle** (int scancode, **task::pipeandfilter::Pipe**< **Keycode** > &outPipe) const

  handles the scancode.

- virtual uint8_t **translateScancode** (int scancode) const

Additional Inherited Members

Detailed Description

standard state handles the most keys like letters and digits and the most other characters (scancode set 1)

Member Function Documentation

virtual bool io::driver::keycode::scancodestates::InitialScanCodeConverterState::testIfScancodeIsBreakCode (int *scancode*) const [private], [virtual] test state specific if a scancode is a breakcode.

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the passed scancode is a breakcode.

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverterState** (p. 268).

virtual AbstractScanCodeConverterState∗ io::driver::keycode::scancodestates::InitialScanCodeConverterState::handle (int *scancode,* task::pipeandfilter::Pipe< Keycode > & *outPipe*) const [private], [virtual] handles the scancode.

the case of a breakcode is resolved before in the abstract state.

Parameters

scancode	scancode
&outPipe	outPipe

@

@plus@

@plus -@

@

@

@skip

Returns

the next state

Implements **io::driver::keycode::scancodestates::AbstractScanCodeConverter-State** (p. 268).

virtual uint8_t io::driver::keycode::scancodestates::InitialScanCodeConverter-State::translateScancode (int *scancode*) const [private], [virtual]

Parameters

scancode	

@

@plus@

@plus -@

@

@

@skip

Returns

the keycode which is translated of the scancode

The documentation for this class was generated from the following file:

- InitialScanCodeConverterState.h

A.4.162 cpu::interrupt::InterruptHandler Class Reference

Inherited by **cpu::interrupt::DefaultExceptionHandler**, **cpu::interrupt::DefaultHandler**, **io::driver::interrupt::PICInterruptHandler**, and **memory::paging::PageFaultExceptionHandler**.

Public Member Functions

- virtual bool **handle** (uint32_t errorCode)=0

  Handles the interrupt.

Member Function Documentation

virtual bool cpu::interrupt::InterruptHandler::handle (uint32_t *errorCode*) [pure virtual] Handles the interrupt.

Has to be implemented by the user.

Parameters

errorCode	An optional error code passed to some exceptions.

@

@plus@

@plus -@

@

@

@skip

Returns

boolean that indicates whether the handling of the interrupt was successful

Implemented in **io::driver::interrupt::PICInterruptHandler** (p. 714), **cpu::interrupt::DefaultHandler** (p. 428), **cpu::interrupt::DefaultExceptionHandler** (p. 427), and **memory::paging::PageFaultExceptionHandler** (p. 685).

The documentation for this class was generated from the following file:

- InterruptHandler.h

A.4.163 cpu::interrupt::InterruptHandlerContainer Class Reference

Public Member Functions

- **InterruptHandlerContainer** (cpu::level::IRQLevel level=cpu::level::IRQ-L_DISPATCH)
- void **setIRQLevel** (cpu::level::IRQLevel level)

  Changes the IRQ level for this IRQ.
- void **insert** (**InterruptHandler** ∗i)

  Inserts an isrHandler into this container.
- void **remove** (**InterruptHandler** ∗i)

  removes the in i specified isrHandler from the container.
- void **handle** (uint32_t errorCode)

  Handles the interrupt.

Private Types

- typedef
  tool::collection::LinkedList
  < **InterruptHandler** ∗ > **Handlers**

  Type of a list of associated interrupt handlers.

Private Attributes

- **Handlers handlers**

  The associated interrupt handlers.
- **task::lock::SpinLock mutex**

  The spin lock raising the IRQ level when entering the handler.

Member Function Documentation

void cpu::interrupt::InterruptHandlerContainer::setIRQLevel (cpu::level-::IRQLevel *level*) [inline] Changes the IRQ level for this IRQ.

Parameters

level	The new IRQ level.

References mutex, and task::lock::SpinLock::setIRQLevel().

Referenced by cpu::interrupt::InterruptManager::setIRQLevel().

void cpu::interrupt::InterruptHandlerContainer::insert (InterruptHandler * *i*) [inline] Inserts an isrHandler into this container.

Parameters

i	isrHandler to be inserted in the container.

References tool::collection::LinkedList< T >::add(), and handlers.

Referenced by cpu::interrupt::InterruptManager::addHandler().

void cpu::interrupt::InterruptHandlerContainer::remove (InterruptHandler * *i*) [inline] removes the in i specified isrHandler from the container.

Parameters

i	isrHandler to be removed in the container

References handlers, and tool::collection::LinkedList< T >::remove().

Referenced by cpu::interrupt::InterruptManager::removeHandler().

void cpu::interrupt::InterruptHandlerContainer::handle (uint32_t *errorCode*) Handles the interrupt.

Calls the methods from the isrHandler in the container. If one isrHandler handled the interrupt the handling stops.

Referenced by cpu::interrupt::InterruptManager::handleInterrupt().

The documentation for this class was generated from the following file:

- InterruptHandlerContainer.h

A.4.164 cpu::interrupt::InterruptManager Class Reference

Manages the handler for the hardware interrupts.

Public Member Functions

- void **setIRQLevel** (uint8_t vectorNumber, cpu::level::IRQLevel level)

  Sets the IRQ level calculated from <vectorNumber>
- void **addHandler** (uint8_t vectorNumber, **InterruptHandler** *isr)

  Adds an handler for an vector.
- void **removeHandler** (uint8_t vectorNumber, **InterruptHandler** *isr)

  Removes an handler for an vector.
- void **handleInterrupt** (uint8_t vectorNumber, uint32_t errorCode)

  Forwards the work to the real handler.

Static Public Member Functions

- static void **init** ()
- static **InterruptManager** & **getInstance** ()

Static Public Attributes

- static int const **NumInterruptVectors** = 256

  Total number of interrupt vectors.
- static int const **NumExceptions** = 32

  Number of CPU exceptions.

Static Private Member Functions

- static void **isrEntryPoint** (unsigned intVector, unsigned errorCode)

  Generic ISR entry point invoked by specific ISR wrappers.

Private Attributes

- **DefaultHandler defaultHandler**

  The handler that will be used for all interrupts if not implemented.
- **DefaultExceptionHandler defaultExceptionHandler**

  The handler that will be used for all exceptions if not implemented.

- **InterruptHandlerContainer isrContainer** [**NumInterruptVectors**]

  The containter for all the handler for each vector.

Static Private Attributes

- static **InterruptManager** ∗ **theInstance**

  Static attribute for the singleton pattern.

- static char **idt** [ ]

  *The Interrupt **Descriptor*** (p. **??**) *Table that we manage.*

Detailed Description

Manages the handler for the hardware interrupts.

Member Function Documentation

void cpu::interrupt::InterruptManager::setIRQLevel (uint8_t *vectorNumber,* cpu::level::IRQLevel *level*) [inline] Sets the IRQ level calculated from <vectorNumber>

Parameters

vectorNumber	The vector number that was executed
level	The level we came from

References isrContainer, and cpu::interrupt::InterruptHandlerContainer::setIRQLevel().

void cpu::interrupt::InterruptManager::addHandler (uint8_t *vectorNumber,* InterruptHandler ∗ *isr*) [inline] Adds an handler for an vector.

Parameters

vectorNumber	The vector number (Interrupt/Exception) for that we inserting the handler

isr	The isr to add for the vector number <vectorNumber>

References cpu::interrupt::InterruptHandlerContainer::insert(), and isrContainer.

void cpu::interrupt::InterruptManager::removeHandler (uint8_t *vectorNumber,* InterruptHandler * *isr*) [inline] Removes an handler for an vector.

Parameters

vectorNumber	The vector number for that we removing the handler
isr	The isr to remove for the vector number <vectorNumber>

References isrContainer, and cpu::interrupt::InterruptHandlerContainer::remove().

void cpu::interrupt::InterruptManager::handleInterrupt (uint8_t *vectorNumber,* uint32_t *errorCode*) [inline] Forwards the work to the real handler.

Parameters

vectorNumber	Which vector number was executed
errorCode	Which error code was raised if it was an exception

References cpu::interrupt::InterruptHandlerContainer::handle(), and isrContainer.

static void cpu::interrupt::InterruptManager::isrEntryPoint (unsigned *intVector,* unsigned *errorCode*) [static], [private] Generic ISR entry point invoked by specific ISR wrappers.

Parameters

intVector	The number of the interrupt.
errorCode	The error code. Only meaningful for certain processor exceptions. For other interrupts, it contains the value zero.

The documentation for this class was generated from the following file:

- **InterruptManager.h**

A.4.165 io::IOPort Class Reference

Represents an I/O port.

Public Member Functions

- **IOPort** (uint16_t **portNumber**)

  Constructor.

- uint16_t **getPortNumber** () const

  Returns the underlying port number.

- void **write8** (uint8_t value) const

  Writes an 8-bit value to the port.

- void **write16** (uint16_t value) const

  Writes a 16-bit value to the port.

- uint8_t **read8** () const

  Reads an 8-bit value from the port.

- uint16_t **read16** () const

  Reads a 16-bit value from the port.

Private Attributes

- uint16_t **portNumber**

  The underlying port number.

Detailed Description

Represents an I/O port.

Constructor & Destructor Documentation

io::IOPort::IOPort (uint16_t *portNumber*) `[inline]` Constructor.

Parameters

portNumber	The underlying I/O port number.

Member Function Documentation

void io::IOPort::write8 (uint8_t *value*) const [inline] Writes an 8-bit value to the port.

Parameters

value	The value to write.

References portNumber.

Referenced by io::driver::interrupt::PICController::doEOI().

void io::IOPort::write16 (uint16_t *value*) const [inline] Writes a 16-bit value to the port.

Parameters

value	The value to write.

References portNumber.

uint8_t io::IOPort::read8 ( ) const [inline] Reads an 8-bit value from the port.

@

@plus@

@plus -@

@

@

@skip
Returns

The value read.

References portNumber.

Referenced by io::driver::block::ata::commands::AtaCommand::acknowledgeIRQ(), and io::driver::pstwo::PS2Device::readStatus().

uint16_t io::IOPort::read16 ( ) const [inline] Reads a 16-bit value from the port.

@

@plus@

@plus -@

@

@

@skip
Returns

The value read.

References portNumber.

The documentation for this class was generated from the following file:

- IOPort.h

A.4.166 ipc::test::IPCReceiverThread Class Reference

IPC receiver.

Inherits **task::Thread**.

Public Member Functions

- **IPCReceiverThread** (::**task::Process** &p,::**task::lock::Semaphore** &notifySem,::**task::lock::Semaphore** &notify2Sem,::**task::lock::Semaphore** &exitSem)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **FantasticObjectImpl impl**
- ::**task::lock::Semaphore** & **notifySem**
- ::**task::lock::Semaphore** & **notify2Sem**
- ::**task::lock::Semaphore** & **exitSem**

Additional Inherited Members

Detailed Description

IPC receiver.

Member Function Documentation

virtual void ipc::test::IPCReceiverThread::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- IPCReceiverThread.h

A.4.167 ipc::test::IPCSenderThread Class Reference

IPC sender.

Inherits **task::Thread**.

Public Member Functions

- **IPCSenderThread** (::**task::Process** &p, **Participant** receiver,::**task::lock::Semaphore** &exitSem, int &answer, int &i, int a, int b, int &addResult, uint32_t *array, uint32_t arrayLen)
- virtual void **run** ()

IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **Participant receiver**
- ::**task::lock::Semaphore** & **exitSem**
- int & **answer**
- int & **i**
- int **a**
- int **b**
- int & **addResult**
- uint32_t * **array**
- uint32_t **arrayLen**

Additional Inherited Members

Detailed Description

IPC sender.

Member Function Documentation

virtual void ipc::test::IPCSenderThread::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- IPCSenderThread.h

A.4.168 ipc::test::IPCTestCase Class Reference

Tests interprocess communication (IPC).

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

Detailed Description

Tests interprocess communication (IPC).

The documentation for this class was generated from the following file:

- IPCTestCase.h

A.4.169 io::driver::interrupt::IRQHandler Class Reference

Encapsulates an IRQ handler.

Inherited by **io::driver::block::ata::AtaBusIrqHandler**, **io::driver::pstwokeyboard-::PS2KeyboardHandler**, **io::driver::rtc::RTCHandler**, and **task::scheduler-::TimerInterruptHandler**.

Public Member Functions

- virtual ∼**IRQHandler** ()

  Destructor.
- virtual bool **handle** ()=0

  Handles the interrupt.

Detailed Description

Encapsulates an IRQ handler.

An **IRQHandler** (p. 578) is invoked by the **PICInterruptHandler** (p. 712) and shall handle the IRQ if it can determine that "its" hardware produced the IRQ. (In some environments, interrupt sharing is possible.)

Member Function Documentation

virtual bool io::driver::interrupt::IRQHandler::handle ( ) [pure virtual]
Handles the interrupt.

@

@plus@

@plus -@

@

@

@skip
Returns

> True if the **IRQHandler** (p. 578) handled the IRQ, else false. In the latter case, the **PICInterruptHandler** (p. 712) will call the next **IRQHandler** (p. 578) in the chain (if available).

Implemented in **io::driver::block::ata::AtaBusIrqHandler** (p. 310), **io::driver::rtc::RTCHandler** (p. 798), **io::driver::pstwokeyboard::PS2KeyboardHandler** (p. 761), and **task::scheduler::TimerInterruptHandler** (p. 885).

The documentation for this class was generated from the following file:

- **IRQHandler.h**

A.4.170 tool::collection::Iterator< T > Class Template Reference

Basic definition of an iterator over a modifiable collection.

Inherited by **tool::collection::ArrayListIterator**< **T** >, and **tool::collection::LinkedListIterator**< **T** >.

Public Member Functions

- virtual bool **moveNext** ()=0

  moves the iterator to the next element

- virtual T & **current** ()=0

  *returns the current element **moveNext()** (p. 580) has to be called before this function is used the first time.*

- virtual void **reset** ()=0

  resets the pointer, as if he was just initialized

- virtual T **remove** ()=0

  removes the element this iterator points to.

Detailed Description

template<class T>class tool::collection::Iterator< T >

Basic definition of an iterator over a modifiable collection.

Member Function Documentation

template<class T > virtual bool tool::collection::Iterator< T >::moveNext ( ) [pure virtual] moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip
Returns

true, if there is a next element, otherwise false

Implemented in **tool::collection::LinkedListIterator< T >** (p. 625), and **tool::collection::ArrayListIterator< T >** (p. 303).

template<class T > virtual T& tool::collection::Iterator< T >::current ( ) [pure virtual] returns the current element **moveNext()** (p. 580) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implemented in **tool::collection::LinkedListIterator< T >** (p. 626), and **tool::collection::ArrayListIterator< T >** (p. 303).

template<class T > virtual T tool::collection::Iterator< T >::remove ( ) [pure virtual] removes the element this iterator points to.

@

@plus@

@plus -@

@

@

@skip
Returns

The element removed.

Implemented in **tool::collection::LinkedListIterator< T >** (p. 626), and **tool::collection::ArrayListIterator< T >** (p. 304).

The documentation for this class was generated from the following file:

- Iterator.h

A.4.171 memory::KernelEnvironment Class Reference

Implements an Allocator environment for the kernel mode.

Inherits **memory::allocator::Environment**.

Public Member Functions

- **KernelEnvironment** (cpu::level::IRQLevel irqLevel, **lmm::LinearAddressSpaceManager** &**lasm**)

  Constructor.

- virtual void **enterCriticalSection** ()

  Enters the Allocator's critical section.

- virtual void **leaveCriticalSection** ()

  Leaves the Allocator's critical section.

- virtual char ∗ **allocateBlock** (uint32_t pages)

  Allocates a memory block.

- virtual bool **freeBlock** (void ∗ptr, uint32_t pages)

  Frees a previously allocated memory block.

Private Attributes

- **task::lock::SpinLock mutex**

  The spin lock associated with this heap.

- char **lock** [sizeof(**task::lock::SpinLock::Lock**)]

  The current lock (if any).

- **lmm::LinearAddressSpaceManager** & **lasm**

  The LinearAddressSpaceManager to use.

Detailed Description

Implements an Allocator environment for the kernel mode.

Constructor & Destructor Documentation

memory::KernelEnvironment::KernelEnvironment (cpu::level::IRQLevel *irqLevel,* lmm::LinearAddressSpaceManager & *lasm*) [inline] Constructor.

Parameters

irqLevel	The IRQ level to be used by the Allocator's spin lock.
lasm	The LinearAddressSpaceManager to use for allocating/freeing memory.

Member Function Documentation

virtual char* memory::KernelEnvironment::allocateBlock (uint32_t *pages*) [virtual] Allocates a memory block.

Parameters

pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the allocated block or NULL if allocation failed.

Implements **memory::allocator::Environment** (p. 482).

virtual bool memory::KernelEnvironment::freeBlock (void * *ptr,* uint32_t *pages*) [virtual] Frees a previously allocated memory block.

Parameters

ptr	The pointer to the previously allocated block.

pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the block could be freed, false otherwise.

Implements **memory::allocator::Environment** (p. 483).

The documentation for this class was generated from the following file:

- **KernelEnvironment.h**

A.4.172 fosCli::commands::KernelVersionCliCommand Class Reference

Inherits **fosCli::parser::CliCommand**.

Public Member Functions

- virtual void **execute** ()
- virtual void **addParameter** (char const ∗parameter)

The documentation for this class was generated from the following file:

- KernelVersionCliCommand.h

A.4.173 fosCli::commands::KernelVersionCliCommand-Creator Class Reference

Inherits **fosCli::parser::CliCommandCreator**.

Public Member Functions

- virtual **parser::CliCommand** ∗ **create** ()

The documentation for this class was generated from the following file:

- KernelVersionCliCommandCreator.h

A.4.174 io::driver::keycode::test::KeyboardTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- KeyboardTestCase.h

A.4.175 io::driver::keycode::Keycode Struct Reference

Definition of a **Keycode** (p. 585).

Data Fields

- int **keycode**
- bool **down**

Detailed Description

Definition of a **Keycode** (p. 585).

Field Documentation

int io::driver::keycode::Keycode::keycode

bool io::driver::keycode::Keycode::down The documentation for this struct was generated from the following file:

- KeycodeDevice.h

A.4.176 io::driver::keycode::KeycodeDevice Class Reference

A **Driver** (p. 455) for the translation of scancodes in keycodes.

Inherits **io::driver::Device**.

Public Member Functions

- **KeycodeDevice** (**tool::collection::Queue**< pstwokeyboard::Scancode > &scancodeBuffer)
- **task::pipeandfilter::Pipe** < **Keycode** > & **getKeycodeBuffer** ()

Private Attributes

- **task::pipeandfilter::Pipe** < **Keycode** > **keycodeBuffer**

  Buffer for the Keycodes.

Detailed Description

A **Driver** (p. 455) for the translation of scancodes in keycodes.

Member Function Documentation

task::pipeandfilter::Pipe<Keycode>& io::driver::keycode::KeycodeDevice-::getKeycodeBuffer ( ) `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

the keycode Buffer

References keycodeBuffer.

The documentation for this class was generated from the following file:

- KeycodeDevice.h

A.4.177 io::driver::keycode::KeycodeDriver Class Reference

Driver (p. 455) for the **KeycodeDevice** (p. 586).

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **KeycodeDevice** ∗ **keycodeDevice**

Additional Inherited Members

Detailed Description

Driver (p. 455) for the **KeycodeDevice** (p. 586).

The documentation for this class was generated from the following file:

- KeycodeDriver.h

A.4.178 io::driver::keycode::KeycodeFilter Class Reference

Filter between the Scancode and **Keycode** (p. 585) buffer.

Inherits **task::pipeandfilter::AbstractFilter**< **pstwokeyboard::Scancode, Keycode** >.

Public Member Functions

- virtual void **filter** (pstwokeyboard::Scancode const &scancode, **task::pipeandfilter::Pipe**< **Keycode** > &paramOutPipe)

Private Attributes

- **scancodestates::AbstractScanCodeConverterState** * **scanCodeConverterState**

  state for the state-Pattern to handle escaped scancodes

Detailed Description

Filter between the Scancode and **Keycode** (p. 585) buffer.

The documentation for this class was generated from the following file:

- KeycodeFilter.h

A.4.179 io::driver::keycode::KeycodeTasklet Class Reference

Inherits **task::tasklet::Tasklet**.

Public Member Functions

- **KeycodeTasklet** (**tool::collection::Queue**< io::driver::pstwokeyboard::Scancode > &paramInPipe, **task::pipeandfilter::Pipe**< **Keycode** >

&paramOutPipe)

- virtual task::tasklet::TaskletState **getState** () const

  Returns the tasklet state.

- virtual void **work** ()

  The actual work function per tasklet.

Private Attributes

- **tool::collection::Queue**
  < io::driver::pstwokeyboard::Scancode > & **input**
- **task::pipeandfilter::Pipe**
  < **io::driver::keycode::Keycode** > & **output**
- **KeycodeFilter** ∗ **filter**

Member Function Documentation

virtual task::tasklet::TaskletState io::driver::keycode::KeycodeTasklet-::getState ( ) const [virtual] Returns the tasklet state.

Only if a Tasklet returns HASWORK it will have its **work()** (p. 589) method invoked.

Implements **task::tasklet::Tasklet** (p. 843).

The documentation for this class was generated from the following file:

- KeycodeTasklet.h

A.4.180 io::driver::charlayout::keymap_entry Struct Reference

Data Fields

- char **normal**
- char **shift**
- char **altgr**
- char **ctrl**

Field Documentation

char io::driver::charlayout::keymap_entry::normal

char io::driver::charlayout::keymap_entry::shift

char io::driver::charlayout::keymap_entry::altgr

char io::driver::charlayout::keymap_entry::ctrl The documentation for this struct was generated from the following file:

- CharLayoutFilter.h

A.4.181 tool::collection::KeyValuePair< Key, Value, KeyComp, ValueComp > Class Template Reference

Public Member Functions

- **KeyValuePair** (Key const &key, Value const &value)
- Key const & **getKey** () const
- Value & **getValue** ()
- Value const & **getValue** () const
- bool **operator==** (**KeyValuePair** const &other) const

Private Attributes

- Key const **key**
- Value **value**

The documentation for this class was generated from the following file:

- KeyValuePair.h

A.4.182 memory::lmm::test::LASMTest Class Reference

Tests the **LinearAddressSpaceManager** (p. 597).

Inherits **test::TestCase**.

Public Member Functions

- virtual char const * **getName** ()
- virtual void **run** ()

Private Member Functions

- void **testLIFO** (**LinearAddressSpaceManager** &mgr)
- void **testFIFO** (**LinearAddressSpaceManager** &mgr)
- void **testRandom** (**LinearAddressSpaceManager** &mgr)
- void **testMany1** (**LinearAddressSpaceManager** &mgr)
- void **testMany2** (**LinearAddressSpaceManager** &mgr)

Additional Inherited Members

Detailed Description

Tests the **LinearAddressSpaceManager** (p. 597).

The documentation for this class was generated from the following file:

- **LASMTest.h**

A.4.183 io::driver::block::LBAddress Struct Reference

Structure which represents a single Logic **Block** (p. 350) Address.

Public Member Functions

- **LBAddress** (uint64_t paramAddress)

Data Fields

- uint64_t **address**

Detailed Description

Structure which represents a single Logic **Block** (p. 350) Address.

The documentation for this struct was generated from the following file:

- LBAddress.h

A.4.184 cpu::level::LevelManager Class Reference

Manages the levels and transitions between the levels <<singleton>>

Public Member Functions

- IRQLevel **getLevel** ()

  Returns the current IRQ level.

- IRQLevel **raiseLevel** (IRQLevel newLevel)

  Raises the current IRQ level.

- void **lowerLevel** (IRQLevel newLevel)

  Lowers the current IRQ level.

- void **addTransitionHandler** (**TransitionHandler** ∗handler)

  Adds a transition handler.

- void **removeTransitionHandler** (**TransitionHandler** ∗handler)

  Removes a transition handler.

Static Public Member Functions

- static void **init** ()
- static **LevelManager** & **getInstance** ()

  Returns.

Private Member Functions

- **LevelManager** ()

  Constructor. Initializes the current level to IRQL_PASSIVE.

- void **setLevel** (IRQLevel level)

  Changes the current IRQ level.

- void **addTransitionHandlerForRaise** (**TransitionHandler** ∗handler)

  Adds a transition handler to the list of handlers used when raising the IRQ level.

- void **addTransitionHandlerForLower** (**TransitionHandler** ∗handler)

  Adds a transition handler to the list of handlers used when lowering the IRQ level.

Private Attributes

- IRQLevel **currentLevel**

  The current IRQ level.

- **tool::collection::LinkedList**
  < **TransitionHandler** ∗ > ∗ **handlersForRaise**

  The transition handlers, sorted ascending by lowest IRQ level.

- **tool::collection::LinkedList**
  < **TransitionHandler** ∗ > ∗ **handlersForLower**

  The transition handlers, sorted descending by highest IRQ level.

Static Private Attributes

- static **LevelManager theInstance**

  The singleton instance.

Detailed Description

Manages the levels and transitions between the levels <<singleton>>

Member Function Documentation

void cpu::level::LevelManager::setLevel (IRQLevel *level*) [inline], [private] Changes the current IRQ level.

Parameters

level	The new IRQ level.

References currentLevel.

void cpu::level::LevelManager::addTransitionHandlerForRaise (TransitionHandler * *handler*) [private] Adds a transition handler to the list of handlers used when raising the IRQ level.

Parameters

handler	The handler to add.

Referenced by addTransitionHandler().

void cpu::level::LevelManager::addTransitionHandlerForLower (TransitionHandler * *handler*) [private] Adds a transition handler to the list of handlers used when lowering the IRQ level.

Parameters

handler	The handler to add.

Referenced by addTransitionHandler().

static LevelManager& cpu::level::LevelManager::getInstance ( ) [inline], [static] Returns.

@

@plus@

@plus -@

@

@

@skip
Returns

the instance of this singleton **LevelManager** (p. 592).

References theInstance.

IRQLevel cpu::level::LevelManager::raiseLevel (IRQLevel *newLevel*)
Raises the current IRQ level.

The new level must be greater than or equal to the current one.

Parameters

newLevel	The new IRQ level.

@

@plus@

@plus -@

@

@

@skip
Returns

The old IRQ level.

void cpu::level::LevelManager::lowerLevel (IRQLevel *newLevel*) Lowers the current IRQ level.

The new level must be less than or equal to the current one.

Parameters

newLevel	The new IRQ level.

void cpu::level::LevelManager::addTransitionHandler (TransitionHandler * *handler*) `[inline]` Adds a transition handler.

Parameters

handler	The handler to add.

References addTransitionHandlerForLower(), and addTransitionHandlerForRaise().

void cpu::level::LevelManager::removeTransitionHandler (TransitionHandler * *handler*) `[inline]` Removes a transition handler.

Parameters

handler	The handler to remove.

References handlersForLower, and handlersForRaise.

The documentation for this class was generated from the following file:

- LevelManager.h

A.4.185 memory::lmm::LinearAddressSpaceManager Class Reference

The **LinearAddressSpaceManager** (p. 597) (LASM) manages the linear address space of one process.

Data Structures

- struct **ReservationTable**

  Holds entries for address space management.

Public Member Functions

- **LinearAddressSpaceManager** (**memory::paging::PageDirectory** ∗**pageDir**, uint32_t linearStart, uint32_t linearEnd)

  Constructor.

- ∼**LinearAddressSpaceManager** ()

  Unmaps all reserved address space.

- **memory::paging::PageDirectory** & **getPageDirectory** () const

  Returns the underlying PageDirectory.

- **AddressSpaceRange allocateSpace** (uint32_t sizeInPages)

  Allocates linear address space.

- bool **freeSpace** (**AddressSpaceRange** const &range)

  Frees linear address space previously allocated by acquireReservation().

Static Public Member Functions

- static **LinearAddressSpaceManager** & **getKernelLASM** ()

Private Member Functions

- **LinearAddressSpaceManager** ()

  Default constructor. Never executed, only to make compiler happy.

- **ReservationTable** ∗ **createTable** ()

  *Creates a new **ReservationTable** (p. 785).*

- **AddressSpaceRange** ∗ **getUnusedEntry** ()

  Searches for an INVALID address range.

- **AddressSpaceRange suballocateRange** (**AddressSpaceRange** &range, uint32_t **startAddress**, uint32_t **endAddress**, **AddressSpaceRange::Type type**)

  Allocates a subrange within another range.

- **AddressSpaceRange** ∗ **splitEntry** (**AddressSpaceRange** &range, uint32_t splitPoint)

  Splits an address range.

- void **mergeAdjacentEntries** (**AddressSpaceRange** &entry)

  Merges adjacent entries of the same type.

- **AddressSpaceRange allocateSpace** (uint32_t size, **AddressSpaceRange::Type type**)

  *Allocates space in the linear address space managed by this **LinearAddressSpaceManager** (p. 597).*

- bool **checkFreeEntries** ()

  Checks whether enough entries exist.

Static Private Member Functions

- static void **init** (**memory::paging::PageDirectory** ∗**pageDir**, uint32_t linearStart, uint32_t linearEnd)

Private Attributes

- **ReservationTable firstTable**

  The first reservation table.

- **ReservationTable** ∗ **curTable**

  Points to the current reservation table.

- uint32_t **numFreeEntries**

  Number of free entries.

- **memory::paging::PageDirectory** ∗ **pageDir**

  The underlying page directory.

Static Private Attributes

- static uint32_t const **EntriesPerTable** = (PAGE_SIZE - sizeof(**ReservationTable** ∗)) / sizeof(**AddressSpaceRange**)

  *Number of entries per **ReservationTable*** (p. *785).*

- static uint32_t const **MinFreeEntries** = 3

  Minimum number of free entries: One for splitting an entry in order to allocate the next reservation table, and two for splitting an arbitrary entry (needed if a part of a reserved range is freed).

- static **LinearAddressSpaceManager kernelManager**

  *The global **LinearAddressSpaceManager*** (p. *597) managing the kernel address space.*

Friends

- class **memory::lmm::test::LASMTest**
- class **boot::BootManager**

Detailed Description

The **LinearAddressSpaceManager** (p. 597) (LASM) manages the linear address space of one process.

Even the Allocator has to acquire linear space for his mappings as well as other memory management units who wants to the linear address of a process. As

most reservations can be done by the Allocator, this class shall only be used for units which are not willing to map memory by the Allocator.

Constructor & Destructor Documentation

memory::lmm::LinearAddressSpaceManager::LinearAddressSpaceManager (memory::paging::PageDirectory * *pageDir,* uint32_t *linearStart,* uint32_t *linearEnd*) Constructor.

Parameters

pageDir	The page directory to manage.
linearStart	The start address (inclusive) of the range of address space to manage.
linearEnd	The end address (exclusive) of the range of address space to manage.

Member Function Documentation

AddressSpaceRange memory::lmm::LinearAddressSpaceManager::allocateSpace (uint32_t *sizeInPages*) `[inline]` Allocates linear address space.

Note that the address range returned is not mapped to physical memory – you have to do it yourself if you need a mapped range.

Parameters

size	The size of the range in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

An **AddressSpaceRange** (p. 269) object describing the memory region. If no linear address space of desired size is available at present, the returned **AddressSpaceRange** (p. 269) is of type INVALID:

References memory::lmm::AddressSpaceRange::ALLOCATED.

bool memory::lmm::LinearAddressSpaceManager::freeSpace (AddressSpaceRange const & *range*) Frees linear address space previously allocated by acquireReservation().

The range to be released need not constitute the whole allocated range, so you can e.g. allocate ten pages at once and later free one or more pages from this range multiple times. Note, however, that due to implementation issues, freeing parts of a memory region could fail in low memory situations.

Note that, unlike **allocateSpace()** (p. 600), the memory region freed will be unmapped if mapped.

Parameters

range	The range to be freed. Its type is ignored.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the range could be freed, else false.

static LinearAddressSpaceManager& memory::lmm::LinearAddressSpaceManager::getKernelLASM ( ) `[inline]`, `[static]` @

@plus@

@plus -@

@

@

@skip

Returns

A reference to the **LinearAddressSpaceManager** (p. 597) for the kernel address space.

References kernelManager.

static void memory::lmm::LinearAddressSpaceManager::init (memory::paging::PageDirectory * *pageDir,* uint32_t *linearStart,* uint32_t *linearEnd*) [inline], [static], [private]

Parameters

pageDir	The page directory to manage.
linearStart	The start address (inclusive) of the range of address space to manage.
linearEnd	The end address (exclusive) of the range of address space to manage.

References kernelManager, and LinearAddressSpaceManager().

ReservationTable∗ memory::lmm::LinearAddressSpaceManager::create-Table ( ) [private] Creates a new **ReservationTable** (p. 785).

@

@plus@

@plus -@

@

@

@skip
Returns

The new reservation table.

AddressSpaceRange∗ memory::lmm::LinearAddressSpaceManager::get-UnusedEntry ( ) [private] Searches for an INVALID address range.

Always succeeds because **checkFreeEntries()** (p. 605) allocates new entries in advance if necessary. @

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the corresponding entry.

AddressSpaceRange memory::lmm::LinearAddressSpaceManager::suballocateRange (AddressSpaceRange & *range*, uint32_t *startAddress*, uint32_t *endAddress*, AddressSpaceRange::Type *type*) [private] Allocates a subrange within another range.

If necessary, the range is split into more ranges. After allocation, adjacent entries of the same type are merged.

Parameters

range	The parent range.
startAddress	The start address of the address subrange.
endAddress	The end address of the address subrange.
type	The type of the subrange.

@

@plus@

@plus -@

@

@

@skip

Returns

The subrange. Note that this range need not exist in the reservation tables "as is" because of merging.

AddressSpaceRange* memory::lmm::LinearAddressSpaceManager::splitEntry (AddressSpaceRange & *range*, uint32_t *splitPoint*) [private] Splits an address range.

Parameters

range	The range to split.
splitPoint	The split point. It is the new end address of the range passed and the start address of the new range returned.

@

@plus@

@plus -@

@

@

@skip
Returns

The second of the two ranges after the split.

void memory::lmm::LinearAddressSpaceManager::mergeAdjacentEntries (AddressSpaceRange & *entry*) [private] Merges adjacent entries of the same type.

Parameters

entry	The entry which should be merged with adjacent entries if possible.

AddressSpaceRange memory::lmm::LinearAddressSpaceManager::allocateSpace (uint32_t *size,* AddressSpaceRange::Type *type*) [private] Allocates space in the linear address space managed by this **LinearAddressSpaceManager** (p. 597).

Parameters

size	The size of the requested address range in bytes.
type	The type of the new range. May not be INVALID.

@

@plus@

@plus -@

@

@

@skip
Returns

The resulting range. If a range with desired size could not be allocated, a range of type INVALID is returned.

bool memory::lmm::LinearAddressSpaceManager::checkFreeEntries ( ) [private] Checks whether enough entries exist.

If not, a new **ReservationTable** (p. 785) is created and linked. @

@plus@

@plus -@

@

@

@skip
Returns

True if enough entries are available, false if there are not enough entries and new entries could not be allocated.

The documentation for this class was generated from the following file:

- LinearAddressSpaceManager.h

A.4.186 tool::collection::LinearMap< Key, Value, KeyComp, ValueComp > Class Template Reference

LinearMap (p. 606) is an implementation of the interface **Map** (p. 642).

Inherits **tool::collection::Map**< **Key, Value, KeyComp, ValueComp** >, and **tool::collection::ArrayList**< **KeyValuePair**< **Key, Value, KeyComp, ValueComp** > >.

Public Types

- typedef **Map**< Key, Value, KeyComp, ValueComp >::**Element Element**
- typedef **ArrayList**< **Element** > ::**Iterator Iterator**

  The underlying iterator types.
- typedef **ArrayList**< **Element** > ::**ConstIterator ConstIterator**

Public Member Functions

- **LinearMap** (**memory::allocator::Allocator** &**allocator**=memory::getAllocator())

- **LinearMap** (**LinearMap**< Key, Value, KeyComp, ValueComp > const &other)
- virtual void **add** (Key const &k, Value const &v)

  Adds a value to the map.
- virtual Value * **get** (Key const &k)

  Returns the value for the specified key.
- virtual Value const * **get** (Key const &k) const
- virtual void **remove** (Key const &k)

  Removes the key and its value from the map.
- virtual bool **hasKey** (Key const &k) const

  Checks if the specified key is already in the map.
- virtual bool **isEmpty** () const

  Returns true if there is no element in this map.
- virtual uint32_t **getSize** () const

  Returns the count of elements in this map.
- virtual **Iterator** * **getIterator** ()

  Returns an iterator over the map's entries.
- virtual **ConstIterator** * **getIterator** () const
- virtual **Element** * **getKeyValuePair** (Key const &k)

  *Returns the **KeyValuePair** (p. 590) of the given k.*
- virtual **Element** const * **getKeyValuePair** (Key const &k) const

Private Attributes

- **memory::allocator::Allocator** & **allocator**

  The allocator to use.

Additional Inherited Members

Detailed Description

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>>class tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >

LinearMap (p. 606) is an implementation of the interface **Map** (p. 642).

Uses an ArrayList<MapEntry> and iterates over this list to find an Entry. As this is not likely for a **Map** (p. 642) and results in a slow search for entries, a real implementation of a map should be considered to use.

Member Function Documentation

template<typename Key, typename Value, typename KeyComp , typename ValueComp > void tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::add (Key const & *k*, Value const & *v*) [virtual]
Adds a value to the map.

If the key already exists, the value is overwritten!

Parameters

key	the key in the map
key	the value for the specified key

Implements **tool::collection::Map< Key, Value, KeyComp, ValueComp >** (p. 643).

Referenced by io::driver::DeviceManager::add(), and ipc::Registry::set().

template<typename Key, typename Value , typename KeyComp , typename ValueComp > Value * tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::get (Key const & *k*) [virtual] Returns the value for the specified key.

Parameters

k	the key to search for

@

@plus@

@plus -@

@

@

@skip
Returns

the value of the specified key, if the key couldn't be found, NULL is returned

Implements **tool::collection::Map< Key, Value, KeyComp, ValueComp >** (p. 644).

Referenced by ipc::Registry::get(), and io::driver::DeviceManager::get().

template<typename Key, typename Value , typename KeyComp , typename ValueComp > void tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::remove (Key const & *k*) `[virtual]` Removes the key and its value from the map.

Parameters

key	key
value	value

Implements **tool::collection::Map< Key, Value, KeyComp, ValueComp >** (p. 644).

Referenced by io::driver::DeviceManager::remove(), and ipc::Registry::unset().

template<typename Key, typename Value , typename KeyComp , typename ValueComp > bool tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::hasKey (Key const & *k*) const `[virtual]` Checks if the specified key is already in the map.

Parameters

k	the key to search

@

@plus@

@plus -@

@

@

@skip
Returns

true if the key is already in the map, otherwise false

Implements **tool::collection::Map**< **Key, Value, KeyComp, ValueComp** > (p. 645).

template<**typename Key, typename Value, typename KeyComp = Comparator**<**Key**>**, typename ValueComp = Comparator**<**Value**>> **virtual bool tool::collection::LinearMap**< **Key, Value, KeyComp, ValueComp** >**::isEmpty ( ) const** [inline], [virtual] Returns true if there is no element in this map.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the map is empty, otherwise false.

Implements **tool::collection::Map**< **Key, Value, KeyComp, ValueComp** > (p. 645).

template<**typename Key, typename Value, typename KeyComp = Comparator**<**Key**>**, typename ValueComp = Comparator**<**Value**>> **virtual uint32_t**

tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::getSize ( ) const [inline], [virtual] Returns the count of elements in this map.

@

@plus@

@plus -@

@

@

@skip
Returns

The size of the map.

Implements **tool::collection::Map< Key, Value, KeyComp, ValueComp >** (p. 646).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual Iterator* tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::getIterator ( ) [inline], [virtual] Returns an iterator over the map's entries.

@

@plus@

@plus -@

@

@

@skip
Returns

an iterator

Implements **tool::collection::Map< Key, Value, KeyComp, ValueComp >** (p. 646).

template<typename Key, typename Value , typename KeyComp , typename ValueComp > LinearMap< Key, Value, KeyComp, ValueComp

>::Element * tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::getKeyValuePair (Key const & *k*) [virtual] Returns the **KeyValuePair** (p. 590) of the given k.

Parameters

k	the key to search for

@

@plus@

@plus -@

@

@

@skip

Returns

the corresponding **KeyValuePair** (p. 590) or NULL, if no KVP could be found

References tool::collection::ArrayListIterator< T >::current(), and tool::collection::ArrayListIterator< T >::moveNext().

The documentation for this class was generated from the following file:

- LinearMap.h

A.4.187 tool::collection::test::LinearMapTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char * **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- LinearMapTestCase.h

A.4.188 tool::collection::LinkedList< T > Class Template Reference

Implementation of **List** (p. 628).

Inherits **tool::collection::List**< **T** >.

Public Types

- typedef **LinkedListIterator**< T > **Iterator**

  The underlying iterator types.

- typedef **ConstLinkedListIterator**< T > **ConstIterator**

Public Member Functions

- **LinkedList** (**memory::allocator::Allocator** &**allocator**=memory::getAllocator())

  *Constructor for class **LinkedList** (p. 613).*

- **LinkedList** (**LinkedList** const &other)

  Copy constructor.

- virtual ~**LinkedList** ()

  *Destructor for class **LinkedList** (p. 613).*

- virtual **memory::allocator::Allocator** & **getAllocator** () const

  Returns the underlying allocator.

- virtual void **add** (T const &element)

  Adds an element to the end of the list.

- virtual bool **remove** (T const &element)

  Removes the first element from the list that is equal to the passed one.

- virtual bool **isEmpty** () const

  Returns true if there are no elements in this list.

- virtual uint32_t **getSize** () const

  Returns the number of elements in this list.

- virtual T & **operator[ ]** (const uint32_t index)

  Returns the element at the specified index.

- virtual T const & **operator[ ]** (const uint32_t index) const
- virtual **Iterator** ∗ **getIterator** ()

  Returns an iterator for this list.

- virtual **ConstIterator** ∗ **getIterator** () const
- virtual bool **add** (T const &insert, T const ∗after)

  adds an element

Private Member Functions

- **LinkedList**< T > & **operator=** (**LinkedList**< T > const &other)=delete
- void **unlink** (**LinkedListEntry**< T > ∗entry)

  Unlinks and frees passed entry from the list.

- void **addAfter** (T const &element, **LinkedListEntry**< T > ∗after)

  Adds passed element behind entry 'after'.

- void **addBefore** (T const &element, **LinkedListEntry**< T > ∗before)

  Adds passed element before entry 'before'.

Private Attributes

- **memory::allocator::Allocator** & **allocator**

  The allocator to use.

- **LinkedListEntry**< T > ∗ **firstElement**

  Contains a pointer to the first element of the list.

- **LinkedListEntry**< T > ∗ **lastElement**

  Contains a pointer to the last element of the list.

- uint32_t **currentSize**

  Represents the current size of the list.

Friends

- class **LinkedListIterator**< **T** >
- class **ConstLinkedListIterator**< **T** >

Detailed Description

template<**class T**>**class tool::collection::LinkedList**< **T** >

Implementation of **List** (p. 628).

Uses a chain of elements which contain the real entries. Due to this, it is easy to insert elements at a specific index. In addition no capacity and so no resize of the list is necessary for this implementation.

Constructor & Destructor Documentation

template<**class T** > **tool::collection::LinkedList**< **T** >**::LinkedList (memory-::allocator::Allocator & *allocator* =** `memory::getAllocator()` **)** `[inline]`
Constructor for class **LinkedList** (p. 613).

Creates an empty **LinkedList** (p. 613).

Parameters

allocator	The allocator to use.

template<**class T** > **tool::collection::LinkedList**< **T** >**::LinkedList (Linked-List**< **T** > **const & *other*)** Copy constructor.

Parameters

other	The list to copy.

References tool::collection::LinkedList< T >::add(), tool::collection::LinkedList< T >::firstElement, tool::collection::LinkedListEntry< T >::getEntry(), and tool-::collection::LinkedListEntry< T >::getNextItem().

template<**class T** > **tool::collection::LinkedList**< **T** >**::~LinkedList ( )** `[inline]`**,** `[virtual]` Destructor for class **LinkedList** (p. 613).

Deletes the list BUT NOT the containing objects. If you want all containing object to be deleted, delete them first!

References tool::collection::LinkedListEntry< T >::getNextItem(), and next.

Member Function Documentation

template<class T> void tool::collection::LinkedList< T >::add (T const & *element*) [inline], [virtual] Adds an element to the end of the list.

Parameters

element	the element to add

Implements **tool::collection::List< T >** (p. 629).

References tool::collection::LinkedListEntry< T >::setPreviousItem().

Referenced by io::driver::interrupt::PICInterruptHandler::add(), task::Thread::addSemaphoreUnlocked(), task::tasklet::TaskletManager::addTasklet(), task::Process::attachThread(), cpu::interrupt::InterruptHandlerContainer::insert(), and tool::collection::LinkedList< T >::LinkedList().

template<class T> bool tool::collection::LinkedList< T >::remove (T const & *element*) [virtual] Removes the first element from the list that is equal to the passed one.

This requires operator==(T,T) to be defined.

Parameters

element	the element to remove

@

@plus@

@plus -@

@

@

@skip

Returns

True if the element has been found and removed, else false.

Implements **tool::collection::List< T >** (p. 629).

References tool::collection::LinkedListEntry< T >::getEntry(), and tool::collection::LinkedListEntry< T >::getNextItem().

Referenced by task::Process::detachThread(), io::driver::interrupt::PICInterruptHandler::remove(), and cpu::interrupt::InterruptHandlerContainer::remove().

template<class T > bool tool::collection::LinkedList< T >::isEmpty ( ) const [inline], [virtual] Returns true if there are no elements in this list.

@

@plus@

@plus -@

@

@

@skip
Returns

true if the list is empty, otherwise false

Implements **tool::collection::List< T >** (p. 630).

Referenced by task::Process::detachThread(), task::Thread::hasSemaphoresUnlocked(), task::scheduler::ThreadQueue::isEmpty(), and io::driver::interrupt::PICInterruptHandler::isEmpty().

template<class T > uint32_t tool::collection::LinkedList< T >::getSize ( ) const [inline], [virtual] Returns the number of elements in this list.

@

@plus@

@plus -@

@

@

@skip
Returns

the number of elements in this list

Implements **tool::collection::List< T >** (p. 631).

template<class T > T & tool::collection::LinkedList< T >::operator[ ] (const uint32_t *index*) [inline], [virtual] Returns the element at the specified index.

Parameters

index	index of element to return

Implements **tool::collection::List< T >** (p. 632).

References fatalError(), tool::collection::LinkedListEntry< T >::getEntry(), and tool::collection::LinkedListEntry< T >::getNextItem().

template<class T > LinkedList< T >::Iterator * tool::collection::LinkedList< T >::getIterator ( ) [inline], [virtual] Returns an iterator for this list.

@

@plus@

@plus -@

@

@

@skip
Returns

an iterator

Implements **tool::collection::List< T >** (p. 632).

template<class T> bool tool::collection::LinkedList< T >::add (T const & *insert,* T const * *after*) [virtual] adds an element

Parameters

insert,the	element to add, behind the element
after	to the list. If the element
after	is not found, this method will return

@

@plus@

@plus -@

@

@

@skip
Returns

false otherwise true will be returned. If

Parameters

after	is NULL, the element is put in front of the list.

References tool::collection::LinkedListEntry< T >::getEntry(), and tool::collection::LinkedListEntry< T >::getNextItem().

template<class T> void tool::collection::LinkedList< T >::unlink (LinkedListEntry< T > * *entry*) [private] Unlinks and frees passed entry from the list.

Parameters

entry	The entry to be unlinked and freed.

References tool::collection::LinkedListEntry< T >::getNextItem(), tool::collection::LinkedListEntry< T >::getPreviousItem(), next, and previous.

template<class T> void tool::collection::LinkedList< T >::addAfter (T const & *element,* LinkedListEntry< T > * *after*) [private] Adds passed element behind entry 'after'.

If that entry is NULL, the element is prepended to the list.

Parameters

element	The element to add.
after	A pointer to the element which to add the element behind.

References tool::collection::LinkedListEntry< T >::getNextItem(), next, tool::collection::LinkedListEntry< T >::setNextItem(), and tool::collection::LinkedListEntry< T >::setPreviousItem().

template<class T> void tool::collection::LinkedList< T >::addBefore (T const & *element,* LinkedListEntry< T > * *before*) [private] Adds passed element before entry 'before'.

If that entry is NULL, the element is appended to the list.

Parameters

element	The element to add.
after	A pointer to the element which to add the element before.

References tool::collection::LinkedListEntry< T >::getPreviousItem(), previous, tool::collection::LinkedListEntry< T >::setNextItem(), and tool::collection::LinkedListEntry< T >::setPreviousItem().

Field Documentation

template<class T> uint32_t tool::collection::LinkedList< T >::currentSize [private] Represents the current size of the list.

Will be updated with every add and remove.

The documentation for this class was generated from the following file:

- LinkedList.h

A.4.189 tool::collection::LinkedListEntry< T > Class Template Reference

A **LinkedListEntry** (p. 620) is the elements which encapsulates the original value stored in a link list.

Public Member Functions

- **LinkedListEntry** (T const &value)

  *Constructor for class **LinkedListEntry** (p. 620).*

- ~**LinkedListEntry** ()

  *Destructor for class **LinkedListEntry** (p. 620).*

- void **setPreviousItem** (**LinkedListEntry** *__previous__)

  Sets the previous item.

- void **setNextItem** (**LinkedListEntry** *__next__)

  Sets the next item.

- **LinkedListEntry** * **getPreviousItem** () const

  Returns the previous entry.

- **LinkedListEntry** ∗ **getNextItem** () const

  Returns the next entry.

- T & **getEntry** ()

  Getter for attribute entry.

- T const & **getEntry** () const

Private Attributes

- T **entry**

  *The value stored in the **LinkedList** (p. 613).*

- **LinkedListEntry** ∗ **previousItem**

  *A pointer to the previous **LinkedListEntry** (p. 620).*

- **LinkedListEntry** ∗ **nextItem**

  *A pointer to the next **LinkedListEntry** (p. 620).*

Detailed Description

template<class T>class tool::collection::LinkedListEntry< T >

A **LinkedListEntry** (p. 620) is the elements which encapsulates the original value stored in a link list.

This class provides the capability of chaining various objects of the same type without changing the behavior of this type.

Constructor & Destructor Documentation

template<class T> tool::collection::LinkedListEntry< T >::LinkedListEntry (T const & *value*) Constructor for class **LinkedListEntry** (p. 620).

Parameters

value	is a pointer to the original value, that shall be stored in the **LinkedList** (p. 613).

Member Function Documentation

template<class T> T & tool::collection::LinkedListEntry< T >::getEntry ( ) Getter for attribute entry.

@

@plus@

@plus -@

@

@

@skip
Returns

a pointer to the entry

Referenced by tool::collection::LinkedList< T >::add(), tool::collection::LinkedList< T >::LinkedList(), tool::collection::LinkedList< T >::operator[ ](), and tool::collection::LinkedList< T >::remove().

template<class T> void tool::collection::LinkedListEntry< T >::setPreviousItem (LinkedListEntry< T > * *previous*) Sets the previous item.

Parameters

previous	The previous item in the chain of linked elements

References previous.

Referenced by tool::collection::LinkedList< T >::add(), tool::collection::LinkedList< T >::addAfter(), and tool::collection::LinkedList< T >::addBefore().

template<class T> void tool::collection::LinkedListEntry< T >::setNextItem (LinkedListEntry< T > * *next*) Sets the next item.

Parameters

next	The next item in the chain of linked elements

References next.

Referenced by tool::collection::LinkedList< T >::addAfter(), and tool::collection::LinkedList< T >::addBefore().

template<class T > LinkedListEntry< T > * tool::collection::LinkedListEntry< T >::getPreviousItem ( ) const Returns the previous entry.

@

@plus@

@plus -@

@

@

@skip
Returns

a pointer to the next item in the chain

Referenced by tool::collection::LinkedList< T >::addBefore(), and tool::collection::LinkedList< T >::unlink().

template<class T > LinkedListEntry< T > * tool::collection::LinkedListEntry< T >::getNextItem ( ) const Returns the next entry.

@

@plus@

@plus -@

@

@

@skip
Returns

a pointer to the next item in the chain

Referenced by tool::collection::LinkedList< T >::add(), tool::collection::LinkedList< T >::addAfter(), tool::collection::LinkedList< T >::LinkedList(), tool::collection::LinkedList< T >::operator[ ](), tool::collection::LinkedList< T >::remove(), tool::collection::LinkedList< T >::unlink(), and tool::collection::LinkedList< T >::~LinkedList().

The documentation for this class was generated from the following file:

- LinkedListEntry.h

A.4.190 tool::collection::LinkedListIterator< T > Class Template Reference

Implementation of **tool::collection::Iterator** (p. 579).

Inherits **tool::collection::Iterator**< **T** >.

Public Member Functions

- **LinkedListIterator** (**LinkedList**< T > &**list**)

  Constructor.

- virtual ~**LinkedListIterator** ()

  *Destructor for class **LinkedListIterator** (p. 624).*

- virtual bool **moveNext** ()

  moves the iterator to the next element

- virtual T & **current** ()

  *returns the current element **moveNext()** (p. 625) has to be called before this function is used the first time.*

- virtual T **remove** ()

  removes the element this iterator points to.

- virtual void **reset** ()

  resets the pointer, as if he was just initialized

- bool **movePrevious** ()
- void **insert** (T const &element)

  Inserts an element behind that one this iterator points to.

- void **insertBefore** (T const &element)

  Inserts an element before that one this iterator points to.

Private Attributes

- **LinkedList**< T > & **list**

  Pointer to the underlying list.

- **LinkedListEntry**< T > ∗ **currentPosition**

  Pointer to the current element.

- **LinkedListEntry**< T > ∗& **firstElement**

  *The first element in the **LinkedList** (p. 613).*

- **LinkedListEntry**< T > ∗& **lastElement**

  *The last element in the **LinkedList** (p. 613).*

Detailed Description

template<class T>class tool::collection::LinkedListIterator< T >

Implementation of **tool::collection::Iterator** (p. 579).

Provides the capability to iterate over a **LinkedList** (p. 613) via the **Iterator** (p. 579) interface.

Constructor & Destructor Documentation

template<class T> tool::collection::LinkedListIterator< T >::LinkedList-Iterator (LinkedList< T > & *list*) Constructor.

Parameters

list	The underlying list.

Member Function Documentation

template<class T> virtual bool tool::collection::LinkedListIterator< T >::moveNext ( ) `[virtual]` moves the iterator to the next element

@

@plus@

@plus -@

@

@

@skip

Returns

true, if there is a next element, otherwise false

Implements **tool::collection::Iterator**< **T** > (p. 580).

Referenced by task::scheduler::ThreadQueue::cycle(), task::scheduler::ThreadQueue::enqueueBack(), and io::time::WaitingQueue::~WaitingQueue().

template<**class T**> **virtual T& tool::collection::LinkedListIterator**< **T** >**::current ( )** [virtual] returns the current element **moveNext()** (p. 625) has to be called before this function is used the first time.

@

@plus@

@plus -@

@

@

@skip
Returns

a reference to the current element

Implements **tool::collection::Iterator**< **T** > (p. 581).

template<**class T**> **virtual T tool::collection::LinkedListIterator**< **T** >**::remove ( )** [virtual] removes the element this iterator points to.

@

@plus@

@plus -@

@

@

@skip
Returns

The element removed.

Implements **tool::collection::Iterator**< **T** > (p. 581).

Referenced by task::scheduler::ThreadQueue::dequeue(), and io::time::WaitingQueue::~WaitingQueue().

template<class T> void tool::collection::LinkedListIterator< T >::insert (T const & *element*) Inserts an element behind that one this iterator points to.

This means that the next **moveNext()** (p. 625) call will let the iterator point to the newly created element.

Parameters

element	The element to add.

template<class T> void tool::collection::LinkedListIterator< T >::insertBefore (T const & *element*) Inserts an element before that one this iterator points to.

Parameters

element	The element to add.

Referenced by task::scheduler::ThreadQueue::enqueueBack(), and task::scheduler::ThreadQueue::enqueueFront().

The documentation for this class was generated from the following file:

- LinkedListIterator.h

A.4.191 tool::collection::test::LinkedListTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char * **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- LinkedListTestCase.h

A.4.192 tool::collection::List< T > Class Template Reference

Basic definition of a list.

Inherited by **tool::collection::ArrayList< DriverDescriptor >**, **tool::collection::ArrayList< int >**, **tool::collection::ArrayList< KeyValuePair< int, Device *, Comparator< int >, Comparator< Device * > > >**, **tool::collection::ArrayList< KeyValuePair< Key, Value, KeyComp, ValueComp > >**, **tool::collection::ArrayList< KeyValuePair< uint32_t, task::Process *, Comparator< uint32_t >, Comparator< task::Process * > > >**, **tool::collection::ArrayList< KeyValuePair< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > >**, **tool::collection::ArrayList< memory::paging::PageTable * >**, **tool::collection::LinkedList< cpu::level::TransitionHandler * >**[virtual], **tool::collection::LinkedList< fosCli::parser::CliCommandCreator * >**[virtual], **tool::collection::LinkedList< InterruptHandler * >**[virtual], **tool::collection::LinkedList< io::time::WaitingQueueEntry * >**[virtual], **tool::collection::LinkedList< IRQHandler * >**[virtual], **tool::collection::LinkedList< lock::Semaphore * >**[virtual], **tool::collection::LinkedList< Resource >**[virtual], **tool::collection::LinkedList< task::Thread * >**[virtual], **tool::collection::LinkedList< Tasklet * >**[virtual], **tool::collection::LinkedList< Thread * >**[virtual], **tool::collection::ArrayList< T >**, and **tool::collection::LinkedList< T >**[virtual].

Public Member Functions

- virtual **memory::allocator::Allocator** & **getAllocator** () const =0

  Returns the underlying allocator.

- virtual void **add** (T const &element)=0

  Adds an element to the end of the list.

- virtual bool **remove** (T const &element)=0

  Removes the first element from the list that is equal to the passed one.

- virtual bool **isEmpty** () const =0

  Returns true if there are no elements in this list.

- virtual uint32_t **getSize** () const =0

  Returns the number of elements in this list.

- virtual T & **operator[ ]** (const uint32_t index)=0

  Returns the element at the specified index.

- virtual T const & **operator[ ]** (const uint32_t index) const =0
- virtual **Iterator**< T > ∗ **getIterator** ()=0

  Returns an iterator for this list.

- virtual **ConstIterator**< T > ∗ **getIterator** () const =0

Detailed Description

template<class T>class tool::collection::List< T >

Basic definition of a list.

Member Function Documentation

template<class T> virtual void tool::collection::List< T >::add (T const & *element*) [pure virtual] Adds an element to the end of the list.

Parameters

element	the element to add

Implemented in **tool::collection::LinkedList< T >** (p. 616), and **tool::collection::ArrayList< T >** (p. 297).

template<class T> virtual bool tool::collection::List< T >::remove (T const & *element*) [pure virtual] Removes the first element from the list that is equal to the passed one.

This requires operator==(T,T) to be defined.

Parameters

element	the element to remove

@

@plus@

@plus -@

@

@

@skip
Returns

True if the element has been found and removed, else false.

Implemented in **tool::collection::LinkedList< T >** (p. 616), and **tool::collection::ArrayList< T >** (p. 297).

template<class T> virtual bool tool::collection::List< T >::isEmpty ( ) const [pure virtual] Returns true if there are no elements in this list.

@

@plus@

@plus -@

@

@

@skip
Returns

true if the list is empty, otherwise false

Implemented in **tool::collection::LinkedList< T >** (p. 617), **tool::collection::LinkedList< IRQHandler * >** (p. 617), **tool::collection::LinkedList< Thread * >** (p. 617), **tool::collection::LinkedList< cpu::level::TransitionHandler * >** (p. 617), **tool::collection::LinkedList< InterruptHandler * >** (p. 617), **tool::collection::LinkedList< task::Thread * >** (p. 617), **tool::collection::LinkedList< fosCli::parser::CliCommandCreator * >** (p. 617), **tool::collection::LinkedList< Tasklet * >** (p. 617), **tool::collection::LinkedList< io::time::WaitingQueueEntry * >** (p. 617), **tool::collection::LinkedList< lock::Semaphore * >** (p. 617), **tool::collection::LinkedList< Resource >** (p. 617), **tool::collection::ArrayList< T >** (p. 298), **tool::collection::ArrayList< KeyValuePair< uint32_t, task::Process *, Comparator< uint32_t >, Comparator< task::Process * > > >** (p. 298), **tool::collection::ArrayList< KeyValuePair< int, Device *, Comparator< int >, Comparator< Device * > > >** (p. 298), **tool::collection::ArrayList< DriverDescriptor >** (p. 298), **tool::collection::ArrayList< KeyValuePair< Key, Value, KeyComp, ValueComp > >** (p. 298), **tool::collection::ArrayList< memory::paging::PageTable * >** (p. 298), **tool::collection::ArrayList< int >** (p. 298), **tool::collection::ArrayList< KeyValuePair< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > >** (p. 298), **tool::collection::LinearMap< Key,**

Value, KeyComp, ValueComp > (p. 610), **tool::collection::LinearMap**< **int, Device** ∗ > (p. 610), **tool::collection::LinearMap**< **uint32_t, task::-Process** ∗ > (p. 610), and **tool::collection::LinearMap**< **uint32_t, void** ∗ > (p. 610).

template<**class T**> **virtual uint32_t tool::collection::List**< **T** >**::getSize ( ) const** [pure virtual] Returns the number of elements in this list.

@

@plus@

@plus -@

@

@

@skip
Returns

the number of elements in this list

Implemented in **tool::collection::LinkedList**< **T** > (p. 617), **tool::collection-::LinkedList**< **IRQHandler** ∗ > (p. 617), **tool::collection::LinkedList**< **Thread** ∗ > (p. 617), **tool::collection::LinkedList**< **cpu::level::TransitionHandler** ∗ > (p. 617), **tool::collection::LinkedList**< **InterruptHandler** ∗ > (p. 617), **tool::collection::LinkedList**< **task::Thread** ∗ > (p. 617), **tool::collection::-LinkedList**< **fosCli::parser::CliCommandCreator** ∗ > (p. 617), **tool::collection-::LinkedList**< **Tasklet** ∗ > (p. 617), **tool::collection::LinkedList**< **io::time-::WaitingQueueEntry** ∗ > (p. 617), **tool::collection::LinkedList**< **lock::-Semaphore** ∗ > (p. 617), **tool::collection::LinkedList**< **Resource** > (p. 617), **tool::collection::ArrayList**< **T** > (p. 299), **tool::collection::ArrayList**< **KeyValuePair**< **uint32_t, task::Process** ∗**, Comparator**< **uint32_t** >**, Comparator**< **task::Process** ∗ > > > (p. 299), **tool::collection::ArrayList**< **KeyValue-Pair**< **int, Device** ∗**, Comparator**< **int** >**, Comparator**< **Device** ∗ > > > (p. 299), **tool::collection::ArrayList**< **DriverDescriptor** > (p. 299), **tool-::collection::ArrayList**< **KeyValuePair**< **Key, Value, KeyComp, ValueComp** > > (p. 299), **tool::collection::ArrayList**< **memory::paging::PageTable** ∗ > (p. 299), **tool::collection::ArrayList**< **int** > (p. 299), **tool::collection::-ArrayList**< **KeyValuePair**< **uint32_t, void** ∗**, Comparator**< **uint32_t** >**, Comparator**< **void** ∗ > > > (p. 299), **tool::collection::LinearMap**< **Key, Value, KeyComp, ValueComp** > (p. 610), **tool::collection::LinearMap**< **int, Device** ∗ > (p. 610), **tool::collection::LinearMap**< **uint32_t, task::-**

Process $*>$ (p. 610), and **tool::collection::LinearMap**$<$ **uint32_t, void** $*>$ (p. 610).

template$<$**class T**$>$ **virtual T& tool::collection::List**$<$ **T** $>$**::operator[ ] (const uint32_t *index*)** [pure virtual] Returns the element at the specified index.

Parameters

index	index of element to return

Implemented in **tool::collection::LinkedList**$<$ **T** $>$ (p. 618), **tool::collection::LinkedList**$<$ **IRQHandler** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **Thread** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **cpu::level::TransitionHandler** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **InterruptHandler** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **task::Thread** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **fosCli::parser::CliCommandCreator** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **Tasklet** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **io::time::WaitingQueueEntry** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **lock::Semaphore** $*>$ (p. 618), **tool::collection::LinkedList**$<$ **Resource** $>$ (p. 618), **tool::collection::ArrayList**$<$ **T** $>$ (p. 300), **tool::collection::ArrayList**$<$ **KeyValuePair**$<$ **uint32_t, task::Process** $*$**, Comparator**$<$ **uint32_t** $>$**, Comparator**$<$ **task::Process** $*>>>$ (p. 300), **tool::collection::ArrayList**$<$ **KeyValuePair**$<$ **int, Device** $*$**, Comparator**$<$ **int** $>$**, Comparator**$<$ **Device** $*>>>$ (p. 300), **tool::collection::ArrayList**$<$ **DriverDescriptor** $>$ (p. 300), **tool::collection::ArrayList**$<$ **KeyValuePair**$<$ **Key, Value, KeyComp, ValueComp** $>>$ (p. 300), **tool::collection::ArrayList**$<$ **memory::paging::PageTable** $*>$ (p. 300), **tool::collection::ArrayList**$<$ **int** $>$ (p. 300), and **tool::collection::ArrayList**$<$ **KeyValuePair**$<$ **uint32_t, void** $*$**, Comparator**$<$ **uint32_t** $>$**, Comparator**$<$ **void** $*>>>$ (p. 300).

template$<$**class T**$>$ **virtual Iterator**$<$**T**$>*$ **tool::collection::List**$<$ **T** $>$**::getIterator ( )** [pure virtual] Returns an iterator for this list.

@

@plus@

@plus -@

@

@

@skip
Returns

an iterator

Implemented in **tool::collection::LinkedList< T >** (p. 618), **tool::collection::LinkedList< IRQHandler * >** (p. 618), **tool::collection::LinkedList< Thread * >** (p. 618), **tool::collection::LinkedList< cpu::level::TransitionHandler * >** (p. 618), **tool::collection::LinkedList< InterruptHandler * >** (p. 618), **tool::collection::LinkedList< task::Thread * >** (p. 618), **tool::collection::LinkedList< fosCli::parser::CliCommandCreator * >** (p. 618), **tool::collection::LinkedList< Tasklet * >** (p. 618), **tool::collection::LinkedList< io::time::WaitingQueueEntry * >** (p. 618), **tool::collection::LinkedList< lock::Semaphore * >** (p. 618), **tool::collection::LinkedList< Resource >** (p. 618), **tool::collection::ArrayList< T >** (p. 299), **tool::collection::ArrayList< KeyValuePair< uint32_t, task::Process *, Comparator< uint32_t >, Comparator< task::Process * > > >** (p. 299), **tool::collection::ArrayList< KeyValuePair< int, Device *, Comparator< int >, Comparator< Device * > > >** (p. 299), **tool::collection::ArrayList< DriverDescriptor >** (p. 299), **tool::collection::ArrayList< KeyValuePair< Key, Value, KeyComp, ValueComp > >** (p. 299), **tool::collection::ArrayList< memory::paging::PageTable * >** (p. 299), **tool::collection::ArrayList< int >** (p. 299), **tool::collection::ArrayList< KeyValuePair< uint32_t, void *, Comparator< uint32_t >, Comparator< void * > > >** (p. 299), **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 611), **tool::collection::LinearMap< int, Device * >** (p. 611), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 611), and **tool::collection::LinearMap< uint32_t, void * >** (p. 611).

The documentation for this class was generated from the following file:

- List.h

A.4.193 api::task::LoadProgramRequest Class Reference

A **LoadProgramRequest** (p. 633) represents the desire to load an application.

Inherits **ipc::Request**.

Public Member Functions

- **LoadProgramRequest** (void *startPointer, uint32_t **sizeInByte**)

Constructor.

- void ∗ **getStartPointer** ()

  Getter for the start address.

- uint32_t **getSizeInByte** ()

  Getter for sizeInByte.

Private Attributes

- uint32_t **sizeInByte**

  Size in byte of the application that shall be loaded.

- void ∗ **ptr**

  Start address of the memory space where the applications code is stored.

Static Private Attributes

- static const uint32_t **id** = 11

  Identifier of this request.

Additional Inherited Members

Detailed Description

A **LoadProgramRequest** (p. 633) represents the desire to load an application.

The startAdress and the size of the application in byte is necessary to perform the load.

Constructor & Destructor Documentation

api::task::LoadProgramRequest::LoadProgramRequest (void ∗ *startPointer*, uint32_t *sizeInByte*) [inline] Constructor.

Parameters

startPointer	is the start address of the memory space where the applications code is stored
sizeInByte	Size in byte of the applications code that shall be loaded

Member Function Documentation

void* api::task::LoadProgramRequest::getStartPointer ( ) [inline]

Getter for the start address.

@

@plus@

@plus -@

@

@

@skip

Returns

the start address of the applications code

References ptr.

uint32_t api::task::LoadProgramRequest::getSizeInByte ( ) [inline]

Getter for sizeInByte.

@

@plus@

@plus -@

@

@

@skip

Returns

the size of the applications code in byte

References sizeInByte.

The documentation for this class was generated from the following file:

- LoadProgramRequest.h

A.4.194 task::lock::Semaphore::Lock Class Reference

Locker class which acquires a semaphore in the constructor and releases it in the destructor.

Public Member Functions

- **Lock** (**Semaphore** &**mutex**)

  Constructor.

- ∼**Lock** ()

  Destructor.

Private Attributes

- **Semaphore** & **mutex**

  The semaphore.

Detailed Description

Locker class which acquires a semaphore in the constructor and releases it in the destructor.

Constructor & Destructor Documentation

task::lock::Semaphore::Lock::Lock (Semaphore & *mutex*) [inline]
Constructor.

Invokes **down()** (p. 818) on the semaphore.

Parameters

mutex	The underlying semaphore.

References task::lock::Semaphore::down().

task::lock::Semaphore::Lock::∼Lock ( ) [inline] Destructor.

Invokes **up()** (p. 818) on the semaphore.

References mutex, and task::lock::Semaphore::up().

The documentation for this class was generated from the following file:

- kernel/task/lock/Semaphore.h

A.4.195 task::lock::SpinLock::Lock Class Reference

Locker class which acquires a spin lock in the constructor and releases it in the destructor.

Public Member Functions

- **Lock** (**SpinLock** &**mutex**)

  Constructor.

- ~**Lock** ()

  Destructor.

- void **yield** ()

  *Invokes **yield()** (p. 637) on the spin lock.*

Private Attributes

- **SpinLock** & **mutex**

  The spin lock.

- cpu::level::IRQLevel **oldLevel**

  The old IRQ level.

Detailed Description

Locker class which acquires a spin lock in the constructor and releases it in the destructor.

Constructor & Destructor Documentation

task::lock::SpinLock::Lock::Lock (SpinLock & *mutex*) `[inline]` Constructor.

Invokes **acquireLock()** (p. 826) on the spin lock.

Parameters

mutex	The underlying spin lock.

References task::lock::SpinLock::acquireLock(), and oldLevel.

task::lock::SpinLock::Lock::∼Lock ( ) [inline] Destructor.

Invokes **releaseLock()** (p. 826) on the spin lock.

References mutex, oldLevel, and task::lock::SpinLock::releaseLock().

The documentation for this class was generated from the following file:

- SpinLock.h

A.4.196 LockTesterThreadDown Class Reference

Inherits **task::Thread**.

Public Member Functions

- **LockTesterThreadDown** (**task::Process** &parent, **task::lock::Semaphore** ∗s)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **task::lock::Semaphore** ∗ **semaphore**

Additional Inherited Members

Member Function Documentation

virtual void LockTesterThreadDown::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- LockTesterThreadDown.h

A.4.197 LockTesterThreadUp Class Reference

Inherits **task::Thread**.

Public Member Functions

- **LockTesterThreadUp** (**task::Process** &parent, **task::lock::Semaphore** *s)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **task::lock::Semaphore** * **semaphore**

Additional Inherited Members

Member Function Documentation

virtual void LockTesterThreadUp::run ( ) `[virtual]` IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- LockTesterThreadUp.h

A.4.198 Logger Class Reference

Logger (p. 640) to log messages of different priorities on the console.

Public Member Functions

- void **setLogLevel** (LogLevel level)
- void **log** (char const ∗s, LogLevel level)
- void **logFormatted** (LogLevel level, char const ∗fmt...)
- void **newLine** (LogLevel level)
- void **logLine** (char const ∗s, LogLevel level)

Private Member Functions

- LogLevel **getLogLevel** ()

Private Attributes

- LogLevel **logLevel**

Detailed Description

Logger (p. 640) to log messages of different priorities on the console.

The documentation for this class was generated from the following file:

- kernel/tool/utils.h

A.4.199 task::priorityinheritance::test::LowPrioThread Class Reference

A low prioritized thread that has access to a resource that a higher prioritized thread wants to have access to.

Inherits **task::Thread**.

Public Member Functions

- **LowPrioThread** (**Process** &**process**, **lock::Semaphore** &**sem**, **lock::-Semaphore** &**start**, **lock::Semaphore** &**exit**)

  Constructor.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **lock::Semaphore** & **sem**

  The semaphore that protects some unspecified resource.

- **lock::Semaphore** & **start**

  The start semaphore released on thread start.

- **lock::Semaphore** & **exit**

  The exit semaphore released on thread exit.

Detailed Description

A low prioritized thread that has access to a resource that a higher prioritized thread wants to have access to.

Constructor & Destructor Documentation

task::priorityinheritance::test::LowPrioThread::LowPrioThread (Process & *process,* lock::Semaphore & *sem,* lock::Semaphore & *start,* lock::Semaphore & *exit*) Constructor.

Parameters

process	The process this thread should attach to.
sem	The semaphore that protects some unspecified resource.
exit	The exit semaphore.

Member Function Documentation

virtual void task::priorityinheritance::test::LowPrioThread::run ( ) [protected], [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **LowPrioThread.h**

A.4.200 tool::collection::Map< Key, Value, KeyComp, ValueComp > Class Template Reference

Provides a mechanism to map a key to a value.

Inherited by **tool::collection::LinearMap**< **Key, Value, KeyComp, ValueComp** >.

Public Types

- typedef **KeyValuePair**< Key, Value, KeyComp, ValueComp > **Element**

Public Member Functions

- virtual void **add** (Key const &k, Value const &v)=0

  Adds a value to the map.

- virtual void **remove** (Key const &k)=0

  Removes the key and its value from the map.

- virtual bool **hasKey** (Key const &k) const =0

  Checks if the specified key is already in the map.

- virtual bool **isEmpty** () const =0

  Returns true if there is no element in this map.

- virtual uint32_t **getSize** () const =0

  Returns the count of elements in this map.

- virtual Value ∗ **get** (Key const &k)=0

  Returns the value for the specified key.

- virtual Value const ∗ **get** (Key const &k) const =0
- virtual **Iterator**< **Element** > ∗ **getIterator** ()=0

  Returns an iterator over the map's entries.

- virtual **ConstIterator**< **Element** > ∗ **getIterator** () const =0

Detailed Description

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>>class tool::collection::Map< Key, Value, KeyComp, ValueComp >

Provides a mechanism to map a key to a value.

Member Function Documentation

template<typename Key, typename Value, typename KeyComp = Comparator<-Key>, typename ValueComp = Comparator<Value>> virtual void tool-::collection::Map< Key, Value, KeyComp, ValueComp >::add (Key const & *k*, Value const & *v*) `[pure virtual]` Adds a value to the map.

If the key already exists, the value is overwritten!

Parameters

key	the key in the map

key	the value for the specified key

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 608), **tool::collection::LinearMap< int, Device * >** (p. 608), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 608), and **tool::collection::LinearMap< uint32_t, void * >** (p. 608).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual Value* tool::collection::Map< Key, Value, KeyComp, ValueComp >::get (Key const & *k*) [pure virtual] Returns the value for the specified key.

Parameters

k	the key to search for

@

@plus@

@plus -@

@

@

@skip

Returns

the value of the specified key, if the key couldn't be found, NULL is returned

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 608), **tool::collection::LinearMap< int, Device * >** (p. 608), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 608), and **tool::collection::LinearMap< uint32_t, void * >** (p. 608).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual void tool::collection::Map< Key, Value, KeyComp, ValueComp >::remove (Key const & *k*) [pure virtual] Removes the key and its value from the map.

Parameters

key	key
value	value

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 609), **tool::collection::LinearMap< int, Device * >** (p. 609), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 609), and **tool::collection::LinearMap< uint32_t, void * >** (p. 609).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual bool tool::collection::Map< Key, Value, KeyComp, ValueComp >::hasKey (Key const & *k*) const [pure virtual] Checks if the specified key is already in the map.

Parameters

k	the key to search

@

@plus@

@plus -@

@

@

@skip

Returns

true if the key is already in the map, otherwise false

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 609), **tool::collection::LinearMap< int, Device * >** (p. 609), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 609), and **tool::collection::LinearMap< uint32_t, void * >** (p. 609).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual bool tool::collection::Map< Key, Value, KeyComp, ValueComp >::isEmpty ( ) const [pure virtual] Returns true if there is no element in this map.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the map is empty, otherwise false.

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 610), **tool::collection::LinearMap< int, Device * >** (p. 610), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 610), and **tool::collection::LinearMap< uint32_t, void * >** (p. 610).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual uint32_t tool::collection::Map< Key, Value, KeyComp, ValueComp >::getSize ( ) const [pure virtual] Returns the count of elements in this map.

@

@plus@

@plus -@

@

@

@skip
Returns

The size of the map.

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 610), **tool::collection::LinearMap< int, Device * >** (p. 610), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 610), and **tool::collection::LinearMap< uint32_t, void * >** (p. 610).

template<typename Key, typename Value, typename KeyComp = Comparator<Key>, typename ValueComp = Comparator<Value>> virtual Iterator<Element>* tool::collection::Map< Key, Value, KeyComp, ValueComp >::getIterator ( ) [pure virtual] Returns an iterator over the map's entries.

@

@plus@

@plus -@

@

@

@skip
Returns

an iterator

Implemented in **tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >** (p. 611), **tool::collection::LinearMap< int, Device * >** (p. 611), **tool::collection::LinearMap< uint32_t, task::Process * >** (p. 611), and **tool::collection::LinearMap< uint32_t, void * >** (p. 611).

The documentation for this class was generated from the following file:

- Map.h

A.4.201 ipc::MapRequest Class Reference

Encapsulates a request to map some memory region into some target process.

Public Member Functions

- **MapRequest** ()

  Default constructor.

- **MapRequest** (void ****ptr**, uint32_t **size**)

  Constructor.

- void * **getPointer** () const

  Returns the start of the memory region to be mapped.

- void **setPointer** (void *ptr) const

  Changes the start of the memory region.

- uint32_t **getSize** () const

  Returns the size in bytes of the memory region to be mapped.

Private Attributes

- void ** **ptr**

  Points to a pointer denoting the start of the memory region.

- uint32_t **size**

  The size in bytes of the memory region to be mapped.

Detailed Description

Encapsulates a request to map some memory region into some target process.

Constructor & Destructor Documentation

ipc::MapRequest::MapRequest (void ** *ptr,* uint32_t *size*) [inline]
Constructor.

Parameters

ptr	Points to a pointer denoting the start of the memory region.
size	The size in bytes of the memory region to be mapped.

Member Function Documentation

void ipc::MapRequest::setPointer (void * *ptr*) const [inline] Changes the start of the memory region.

Called after region has been mapped.

Parameters

ptr	The start of the memory region in the target process.

References ptr.

The documentation for this class was generated from the following file:

- **Request.h**

A.4.202 io::driver::interrupt::MaskingHandler Class Reference

Inherits **cpu::level::TransitionHandler**.

Public Member Functions

- **MaskingHandler** (**io::driver::interrupt::PICDevice** &**pic**)
- virtual cpu::level::IRQLevel **lowestLevel** () const

  Returns the lowest IRQ level of a transition (inclusive).
- virtual cpu::level::IRQLevel **highestLevel** () const

  Returns the highest IRQ level of a transition (exclusive).
- virtual void **raise** (cpu::level::IRQLevel, cpu::level::IRQLevel toLevel)

  Is called by the LevelManager whenever the IRQ level is raised.
- virtual void **lower** (cpu::level::IRQLevel, cpu::level::IRQLevel toLevel)

  Is called by the LevelManager whenever the IRQ level is lowered.

Private Attributes

- **io::driver::interrupt::PICDevice** & **pic**

  The associated PIC.
- cpu::level::IRQLevel const **lowestDeviceIRQL**

  The lowest IRQ level.
- cpu::level::IRQLevel const **highestDeviceIRQL**

  The highest IRQ level.

Member Function Documentation

virtual cpu::level::IRQLevel io::driver::interrupt::MaskingHandler::lowestLevel ( ) const `[inline]`**,** `[virtual]` Returns the lowest IRQ level of a transition (inclusive).

The handler is only invoked if the new (while raising) or old (while lowering) IRQ level is greater than or equal to this level. @

@plus@

@plus -@

@

@

@skip

Returns

The highest IRQ level.

Implements **cpu::level::TransitionHandler** (p. 886).

References lowestDeviceIRQL.

virtual cpu::level::IRQLevel io::driver::interrupt::MaskingHandler::highestLevel ( ) const [inline], [virtual] Returns the highest IRQ level of a transition (exclusive).

The handler is only invoked if the old (while raising) or new (while lowering) IRQ level is less than this level. @

@plus@

@plus -@

@

@

@skip
Returns

The highest IRQ level.

Implements **cpu::level::TransitionHandler** (p. 887).

References highestDeviceIRQL.

virtual void io::driver::interrupt::MaskingHandler::raise (cpu::level::IRQLevel *fromLevel*, cpu::level::IRQLevel *toLevel*) [inline], [virtual] Is called by the LevelManager whenever the IRQ level is raised.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implements **cpu::level::TransitionHandler** (p. 887).

References io::driver::interrupt::PICDevice::maskIRQs(), and pic.

virtual void io::driver::interrupt::MaskingHandler::lower (cpu::level::IRQLevel *fromLevel*, cpu::level::IRQLevel *toLevel*) [inline], [virtual] Is called by the LevelManager whenever the IRQ level is lowered.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implements **cpu::level::TransitionHandler** (p. 888).

References io::driver::interrupt::PICDevice::maskIRQs(), and pic.

The documentation for this class was generated from the following file:

- MaskingHandler.h

A.4.203 io::driver::interrupt::MasterPICController Class Reference

Inherits **io::driver::interrupt::PICController**.

Public Member Functions

- **MasterPICController** (uint16_t basePort)
- virtual void **maskIRQs** (cpu::level::IRQLevel untilIRQL)

  Masks all IRQs up to a given IRQ level (including).
- virtual void **init** (uint8_t **vectorBase**, cpu::level::IRQLevel **baseIRQL**)
- virtual cpu::level::IRQLevel **getLevel** (uint8_t pin)

  Returns the IRQ level associated with a PIC input.

Protected Member Functions

- virtual void **sendEOI** ()

  Sends an EOI to this PIC and its master PIC, if any.

Private Member Functions

- virtual void **writeICW3** ()

  Writes the ICW3.
- void **attachSlave** (**SlavePICController** &slave, uint8_t pin)

  Attaches a slave PIC.

- void **detachSlave** (uint8_t pin)

  Detaches a slave PIC.

Private Attributes

- cpu::level::IRQLevel **baseIRQL**

  The base IRQ level initialized by init().

- cpu::level::IRQLevel **topIRQL**

  The highest IRQ level plus one used by this (or a slave) PIC.

- int **numSlaves**

  Number of attached slave PICs.

- **SlavePICController** ∗ **slaves** [**NumInputs**]

  The slave PICs.

- **SlavePICController** ∗ **slavesByInput** [**NumInputs**]

  The associated slave PICs sorted by pin.

- uint8_t **masks** [**NumInputs** ∗**NumInputs**]

  The IRQ masks per IRQL.

Friends

- class **SlavePICController**

Additional Inherited Members

Member Function Documentation

virtual void io::driver::interrupt::MasterPICController::maskIRQs (cpu::level::IRQLevel *untilIRQL*) `[virtual]` Masks all IRQs up to a given IRQ level (including).

All higher IRQLs are unmasked.

Parameters

untilIRQL	The IRQ level up to which IRQs are to be masked.

Implements **io::driver::interrupt::PICDevice** (p. 710).

virtual cpu::level::IRQLevel io::driver::interrupt::MasterPICController::getLevel (uint8_t *pin*) [virtual] Returns the IRQ level associated with a PIC input.

Parameters

pin	The PIC input.

@

@plus@

@plus -@

@

@

@skip

Returns

The associated IRQ level.

Implements **io::driver::interrupt::PICDevice** (p. 709).

void io::driver::interrupt::MasterPICController::attachSlave (SlavePICController & *slave*, uint8_t *pin*) [private] Attaches a slave PIC.

Parameters

slave	The slave.
pin	The master PIC's input the slave is attached to.

void io::driver::interrupt::MasterPICController::detachSlave (uint8_t *pin*) [inline], [private] Detaches a slave PIC.

Parameters

pin	The master PIC's input the slave is attached to.

References slavesByInput.

The documentation for this class was generated from the following file:

- MasterPICController.h

A.4.204 tool::MD5 Class Reference

Public Types

- typedef unsigned int **size_type**

Public Member Functions

- **MD5** (const char ∗text)
- void **update** (const unsigned char ∗buf, size_type length)
- void **update** (const char ∗buf, size_type length)
- **MD5** & **finalize** ()
- **uint128_t getDigest** () const

Private Types

- enum { **blocksize** = 64 }
- typedef unsigned char **uint1**
- typedef unsigned int **uint4**

Private Member Functions

- void **init** ()
- void **transform** (const uint1 block[blocksize])

Static Private Member Functions

- static void **decode** (uint4 output[ ], const uint1 input[ ], size_type len)
- static void **encode** (uint1 output[ ], const uint4 input[ ], size_type len)
- static uint4 **F** (uint4 x, uint4 y, uint4 z)
- static uint4 **G** (uint4 x, uint4 y, uint4 z)
- static uint4 **H** (uint4 x, uint4 y, uint4 z)
- static uint4 **I** (uint4 x, uint4 y, uint4 z)
- static uint4 **rotate_left** (uint4 x, int n)
- static void **FF** (uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)

- static void **GG** (uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)
- static void **HH** (uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)
- static void **II** (uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)

Private Attributes

- bool **finalized**
- uint1 **buffer** [blocksize]
- uint4 **count** [2]
- uint4 **state** [4]
- uint1 **digest** [16]

The documentation for this class was generated from the following file:

- MD5.h

A.4.205 api::task::lock::ModifySemaphoreRequest Class Reference

Modifies a Semaphore.

Inherits **ipc::Request**.

Public Member Functions

- uint32_t **getOperation** () const

  Returns the operation.
- uint32_t **getSemaphore** () const

  Returns the semaphore to modify.
- **ModifySemaphoreRequest** (uint32_t **operation**, uint32_t **semaphore**)

  Constructor.

Static Public Attributes

- static const uint32_t **Id** = 6

  The identifier of this Request.

Private Attributes

- uint32_t **operation**

  Operation which is performed by this request according to class description.

- uint32_t **semaphore**

  Identifier for semaphore object within os kernel.

Additional Inherited Members

Detailed Description

Modifies a Semaphore.

0 = up 1 = down

Constructor & Destructor Documentation

api::task::lock::ModifySemaphoreRequest::ModifySemaphoreRequest (uint32_t *operation*, uint32_t *semaphore*) [inline] Constructor.

Parameters

operation	is the operation which shall be performed: 0 -> up(), 1 -> down()
semaphore	is the identifier for the semaphore within the os kernel

The documentation for this class was generated from the following file:

- **ModifySemaphoreRequest.h**

A.4.206 io::vfs::Mount Class Reference

Represents an actual mount.

Inherits **io::vfs::AbstractMount**.

Inherited by **io::vfs::fat::FatMount**, and **io::vfs::test::DummyMount**.

Public Member Functions

- **Mount** (uint8_t **mountID**)
- virtual **object::Ref**< **Directory** > **getRoot** ()=0

- virtual void **addOpenNode** (**FileSystemNode** ∗n)

  Adds an open node to the list of open nodes.

- virtual void **removeOpenNode** (**FileSystemNode** ∗n)

  Removes an open node from the list of open nodes.

- virtual UmountError::enum_t **requestUmount** ()

  Tries to umount this mount.

- virtual bool **isUmounted** () const

  Returns the state of this mount.

- **tool::collection::List**
  < **FileSystemNode** ∗ > ∗ **getOpenNodes** () const

Protected Member Functions

- virtual void **doUmount** ()=0

  This method does the actual umounting.

Private Attributes

- bool **umounted**

  internal state

- **tool::collection::List**
  < **FileSystemNode** ∗ > ∗ **openNodes**

  List of nodes.

Detailed Description

Represents an actual mount.

The mount can be mounted in the vfs.

Member Function Documentation

virtual object::Ref<Directory> io::vfs::Mount::getRoot ( ) `[pure virtual]`

@

@plus@

@plus -@

@

@

@skip

See Also

AbstractMount::getRoot() (p. 265)

Implements **io::vfs::AbstractMount** (p. 265).

Implemented in **io::vfs::test::DummyMount** (p. 475).

virtual void io::vfs::Mount::addOpenNode (FileSystemNode * *n*) [virtual]

Adds an open node to the list of open nodes.

This method is only called internally.

Parameters

n	The node, that should be added

virtual void io::vfs::Mount::removeOpenNode (FileSystemNode * *n*) [virtual] Removes an open node from the list of open nodes.

This method is only called internally.

Parameters

n	The node, that should be removed

virtual UmountError::enum_t io::vfs::Mount::requestUmount ( ) [virtual]

Tries to umount this mount.

@

@plus@

@plus -@

@

@

@skip

Returns

SUCCESS if the volume is umounted INUSE if the volume is still in use - no umount is performed

virtual bool io::vfs::Mount::isUmounted ( ) const [inline], [virtual]

Returns the state of this mount.

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the mount is umounted false, if the mount is mounted

References umounted.

The documentation for this class was generated from the following file:

- Mount.h

A.4.207 tool::collection::Queue< T >::Node Struct Reference

Public Member Functions

- **Node** (T const &data)

Data Fields

- T **data**
- **Node** * **next**

The documentation for this struct was generated from the following file:

- Queue.h

A.4.208 task::pipeandfilter::Pipe< T >::Node Struct Reference

Public Member Functions

- **Node** (T const &data)

Data Fields

- T **data**
- **Node** ∗ **next**

The documentation for this struct was generated from the following file:

- Pipe.h

A.4.209 task::priorityinheritance::test::NormalThread Class Reference

A normal thread doing some computations, running at PRIO_NORMAL.

Inherits **task::Thread**.

Public Member Functions

- **NormalThread** (**Process** &**process**, **lock::Semaphore** &**start**, **lock::-Semaphore** &**exit**)

  Constructor.

- void **requestTermination** ()

  Requests thread termination.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Member Functions

- bool **isTerminationRequested** () const

  Returns true if thread termination has been requested.

Private Attributes

- **lock::SpinLock mutex**

  Protects access to the thread's data.

- bool **shouldExit**

  True if thread should exit.

- **lock::Semaphore** & **start**

  The start semaphore released on thread start.

- **lock::Semaphore** & **exit**

  The exit semaphore released on thread exit.

Detailed Description

A normal thread doing some computations, running at PRIO_NORMAL.

Constructor & Destructor Documentation

task::priorityinheritance::test::NormalThread::NormalThread (Process & *process,* lock::Semaphore & *start,* lock::Semaphore & *exit*) Constructor.

Parameters

process	The process this thread should attach to.
exit	The exit semaphore.

Member Function Documentation

virtual void task::priorityinheritance::test::NormalThread::run ( ) `[protected]`**,** `[virtual]` IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **NormalThread.h**

A.4.210 api::io::console::OutputStringRequest Class Reference

Prints a string on the console.

Inherits **ipc::Request**.

Public Member Functions

- char const ∗ **getString** () const

  Returns the string to be printed on the console.

- std::va_list **getAp** () const
- **OutputStringRequest** (char const ∗**s**,...)

  Constructor.

Static Public Attributes

- static const uint32_t **Id** = 1

  The identifier of this Request.

Private Attributes

- char const ∗const **s**

  The string to print on the console (input).

- std::va_list **ap**

Additional Inherited Members

Detailed Description

Prints a string on the console.

Constructor & Destructor Documentation

api::io::console::OutputStringRequest::OutputStringRequest (char const * *s, ...*) [inline] Constructor.

Parameters

s	The string to print.

The documentation for this class was generated from the following file:

- **OutputStringRequest.h**

A.4.211 memory::allocator::Page Class Reference

A **Page** (p. 663) applies sub-allocation techniques in order to divide a page into smaller blocks.

Data Structures

- class **PartDescriptor**

  *If valid, a **PartDescriptor** (p. 697) describes a contiguous range of parts that are either allocated or free.*

Public Member Functions

- **Page** ()

  *Constructs an empty, invalid **Page** (p. 663).*

- **PageTable** * **getPageTable** () const

  *Returns the managing **PageTable** (p. 690). Use only on valid pages.*

- bool **initialize** (**Allocator** &allocator)

  *Initializes the **Page** (p. 663) by allocating a free page.*

- void **finalize** (**Allocator** &allocator)

*Makes this **Page** (p. 663) invalid, freeing the underlying page.*

- char ∗ **allocate** (uint32_t size)

  *Tries to allocate a contiguous memory block within this **Page** (p. 663).*

- bool **free** (char ∗**address**)

  Frees a previously allocated memory block.

- bool **isEmpty** () const

  *Returns true if this **Page** (p. 663) is empty.*

- bool **isValid** () const

  *Returns true if this **Page** (p. 663) points to a valid (allocated) page.*

Static Public Member Functions

- static uint32_t **getMaxBlockSize** ()

  Returns the maximum size of a block that can be allocated.

Static Public Attributes

- static uint32_t const **PartSize** = PAGE_SIZE >> **PartCountBits**

  Number of bytes per part.

Static Private Member Functions

- static uint32_t **computePartCount** (uint32_t bytes)

  Returns the number of parts needed when allocating a number of bytes.

Private Attributes

- char ∗ **address**

  The address of the page controlled by us.

- uint32_t **allocatedParts**

  The number of allocated parts.

- **PartDescriptor partDescriptors** [**NumParts**]

  The part descriptors.

Static Private Attributes

- static uint32_t const **PartCountBits** = **TypeInfo**<uint8_t>::NumBits - 1

  Number of bits used for the counter within a part descriptor.

- static uint32_t const **TypeBit** = 1 << **PartCountBits**

  Type bit.

- static uint32_t const **PartCountMask** = **TypeBit** - 1

  Part count mask.

- static uint32_t const **NumParts** = PAGE_SIZE / **PartSize**

  Number of parts managed by this object.

Detailed Description

A **Page** (p. 663) applies sub-allocation techniques in order to divide a page into smaller blocks.

In order to achieve this, the page is divided into a fixed number of parts of PartSize bytes. Allocation requests are rounded up to a multiple of PartSize. Contrary to a buddy allocation, the memory blocks can start at every part, not only at powers of two. This provides lower internal fragmentation at the cost of more organizational overhead.

Member Function Documentation

bool memory::allocator::Page::initialize (Allocator & *allocator*) Initializes the **Page** (p. 663) by allocating a free page.

Parameters

allocator	The calling allocator.

@

@plus@

@plus -@

@

@

@skip

Returns

True if a page could be allocated, false otherwise.

void memory::allocator::Page::finalize (Allocator & *allocator*) Makes this **Page** (p. 663) invalid, freeing the underlying page.

Parameters

allocator	The calling allocator.

char* memory::allocator::Page::allocate (uint32_t *size*) Tries to allocate a contiguous memory block within this **Page** (p. 663).

Parameters

size	The size in bytes of the memory block to be allocated.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the allocated memory block. If no suitable free block could be found, NULL is returned.

bool memory::allocator::Page::free (char * *address*) Frees a previously allocated memory block.

Parameters

address	The address of the memory block to be freed.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the block could be freed, false otherwise.

bool memory::allocator::Page::isEmpty ( ) const [inline] Returns true if this **Page** (p. 663) is empty.

An empty **Page** (p. 663) contains at most data used for internal purposes. @

@plus@

@plus -@

@

@

@skip
Returns

True if the **Page** (p. 663) is empty and can be freed, otherwise false

References allocatedParts.

bool memory::allocator::Page::isValid ( ) const [inline] Returns true if this **Page** (p. 663) points to a valid (allocated) page.

Returns false if the **Page** (p. 663) is not initialized. @

@plus@

@plus -@

@

@

@skip
Returns

True if the **Page** (p. 663) is valid, otherwise false.

References address.

static uint32_t memory::allocator::Page::computePartCount (uint32_t *bytes*) [inline], [static], [private] Returns the number of parts needed when allocating a number of bytes.

Parameters

bytes	The number of bytes to allocate.

@

@plus@

@plus -@

@

@

@skip

Returns

The number of needed parts.

References PartSize.

Field Documentation

uint32_t const memory::allocator::Page::PartSize = PAGE_SIZE >> PartCountBits [static] Number of bytes per part.

A part constitutes the smallest allocation unit. All allocation requests are rounded up to a multiple of the size of a part.

Referenced by computePartCount(), and getMaxBlockSize().

The documentation for this class was generated from the following file:

- **Page.h**

A.4.212 memory::paging::PageContext Class Reference

Public Member Functions

- uint32_t **getCR3** () const
- bool **isActive** () const

Static Public Member Functions

- static **PageContext getCurrent** ()

Private Member Functions

- **PageContext** (uint32_t **cr3Context**)

  *Creates a **PageContext** (p. 668) object.*

- void **enableContext** ()

  *Makes this **PageContext** (p. 668) active.*

Static Private Member Functions

- static **PageDirectory** ∗ **init** (unsigned kernelSize, char ∗&firstFreeLinAddr, char ∗&firstFreePhysAddr)

  Initializes the paging subsystem.

- static **PageDirectory** ∗ **createPagingTables** (unsigned int kernelBlocks, **PageEntry** ∗entries, char ∗&firstFreeLinAddr, char ∗&firstFreePhysAddr)

  Creates the initial paging directory by copying the page tables and the page directory created by the OS loader, then switches to this new page directory.

Private Attributes

- uint32_t **cr3Context**

  The encapsulated physical address of the page directory.

Friends

- class **boot::BootManager**
- class **PageDirectory**

Constructor & Destructor Documentation

memory::paging::PageContext::PageContext (uint32_t *cr3Context*) `[inline]`**,** `[private]` Creates a **PageContext** (p. 668) object.

Parameters

---

cr3Context	The physical address of the page directory associated with the page context.

Referenced by getCurrent().

Member Function Documentation

uint32_t memory::paging::PageContext::getCR3 ( ) const [inline]

@

@plus@

@plus -@

@

@

@skip

Returns

The physical address of the page directory this **PageContext** (p. 668) references.

References cr3Context.

Referenced by isActive().

static PageContext memory::paging::PageContext::getCurrent ( ) [inline], [static] @

@plus@

@plus -@

@

@

@skip

Returns

The active **PageContext** (p. 668).

References PageContext().

Referenced by isActive().

bool memory::paging::PageContext::isActive ( ) const [inline] @

@plus@

@plus -@

@

@

@skip
Returns

True if this **PageContext** (p. 668) is active, else false.

References getCR3(), and getCurrent().

static PageDirectory* memory::paging::PageContext::init (unsigned *kernelSize,* char *& *firstFreeLinAddr,* char *& *firstFreePhysAddr*) [static], [private] Initializes the paging subsystem.

Parameters

kernelSize	The kernel size in bytes.
firstFreeLinAddr	Receives the first available linear address behind the kernel and paging structures.
firstFreePhysAddr	Receives the first available physical address behind the kernel and paging structures.

@

@plus@

@plus -@

@

@

@skip
Returns

The linear address of the page directory to be used for the system process.

static PageDirectory* memory::paging::PageContext::createPagingTables (unsigned int *kernelBlocks,* PageEntry * *entries,* char *& *firstFreeLinAddr,* char *& *firstFreePhysAddr*) [static], [private] Creates the initial paging directory by copying the page tables and the page directory created by the OS loader, then switches to this new page directory.

Parameters

kernelBlocks	The size of the kernel in bytes.
entries	Points to the first page entry of the page directory created by the OS loader.
firstFreeLinAddr	Receives the first linear address behind the kernel and the paging structures.
firstFreePhysAddr	Receives the first physical address behind the kernel and the paging structures.

@

@plus@

@plus -@

@

@

@skip

Returns

The new page directory. This page directory is later to be used when creating the system process.

void memory::paging::PageContext::enableContext ( ) [private] Makes this **PageContext** (p. 668) active.

Used by **createPagingTables()** (p. 671) in order to switch to the new **PageDirectory** (p. 672).

The documentation for this class was generated from the following file:

- PageContext.h

A.4.213 memory::paging::PageDirectory Class Reference

Encapsulates a page directory.

Public Member Functions

- **PageContext toContext** ()

*Transforms this **PageDirectory** (p. 672) into a **PageContext** (p. 668).*

- bool **isInitial** () const

  Returns true if this is the initial page directory.

- **PageEntry** ∗ **getEntryForAddress** (void ∗linAddr)

  *Returns the **PageEntry** (p. 682) pointing to the **PageTable** (p. 689) which is responsible for mapping passed address.*

- **PageEntry** ∗ **getEntryWithOffset** (uint32_t index)

  *Returns the **PageEntry** (p. 682) pointing to the **PageTable** (p. 689) with a specified index.*

- **PageTable** ∗ **getTable** (void ∗linAddr)

  *Returns the **PageTable** (p. 689) responsible for mapping passed address.*

- **PageTable** ∗ **getTable** (uint32_t index)

  *Returns the **PageTable** (p. 689) with a specified index.*

- bool **map** (void ∗linAddr, **PageFlags** const &flags=**PageFlags**())

  Maps the page at some linear address to physical memory.

- void **unmap** (void ∗linAddr)

  Unmaps the page at some linear address from physical memory.

- void **map** (void ∗startLinAddr, void ∗endLinAddr, void ∗startPhysAddr, **PageFlags** const &flags=**PageFlags**())

  Maps a linear address range to physical memory starting at a specified physical address.

- void **map** (uint32_t startLinAddr, uint32_t endLinAddr, uint32_t startPhys-Addr, **PageFlags** const &flags=**PageFlags**())

- void **unmap** (void ∗startLinAddr, void ∗endLinAddr)

  Unmaps a linear address range from physical memory.

- void **unmap** (uint32_t startLinAddr, uint32_t endLinAddr)

Private Member Functions

- **PageDirectory** ()

  Creates the initial page directory.

- **PageDirectory** (::**task::Process** ∗**process**)

  Creates a page directory associated with a new process.

- **PageTable** ∗ **getOrAllocPageTable** (void ∗linAddr)

*Retrieves the **PageTable** (p. 689) for an address.*

- **PageTable** ∗ **getOrAllocPageTable** (uint32_t index)

  *Retrieves the index-th **PageTable** (p. 689).*

- void **freePageTable** (uint32_t index, bool ifEmpty)

  Frees a user space page table.

- bool **checkIfUnmappable** (void ∗linAddr)

  Returns true if passed linear address may not be mapped (because it is already in use by the system).

- void **setProcess** (::**task::Process** ∗**process**)

  *Sets the process associated with this **PageDirectory** (p. 672).*

Static Private Member Functions

- static uint32_t **mapLinearAddressToPageIndex** (void ∗linAddr)

  Maps a linear address to a page entry index.

- static **PageDirectory** ∗ **getCurrent** ()

  *Returns the currently active **PageDirectory** (p. 672).*

Private Attributes

- **PageEntry entries** [ENTRIES_IN_TABLE_OR_DIR]

  *The **PageEntry** (p. 682) array. Flush TLB whenever you modify these!*

- ::**task::Process** ∗ **process**

  The associated process.

Friends

- class **PageContext**
- class **task::Process**
- class **boot::BootManager**

Detailed Description

Encapsulates a page directory.

Constructor & Destructor Documentation

memory::paging::PageDirectory::PageDirectory (::task::Process * *process*) [private] Creates a page directory associated with a new process.

Parameters

process	The associated process.

Member Function Documentation

bool memory::paging::PageDirectory::isInitial ( ) const [inline] Returns true if this is the initial page directory.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

PageEntry* memory::paging::PageDirectory::getEntryForAddress (void * *linAddr*) [inline] Returns the **PageEntry** (p. 682) pointing to the **PageTable** (p. 689) which is responsible for mapping passed address.

Parameters

address	The linear address to map.

@

@plus@

@plus -@

@

@

@skip
Returns

The entry for the **PageTable** (p. 689) associated with passed address.

References getEntryWithOffset(), and mapLinearAddressToPageIndex().

PageEntry* memory::paging::PageDirectory::getEntryWithOffset (uint32_t *index*) [inline] Returns the **PageEntry** (p. 682) pointing to the **PageTable** (p. 689) with a specified index.

Parameters

index	The index of the **PageTable** (p. 689).

@

@plus@

@plus -@

@

@

@skip

Returns

The entry for the **PageTable** (p. 689) associated with passed index.

References entries.

Referenced by getEntryForAddress().

PageTable∗ memory::paging::PageDirectory::getTable (void ∗ *linAddr*) [inline] Returns the **PageTable** (p. 689) responsible for mapping passed address.

Parameters

address	The linear address to map.

@

@plus@

@plus -@

@

@

@skip

Returns

The **PageTable** (p. 689) associated with passed address.

References mapLinearAddressToPageIndex().

PageTable∗ memory::paging::PageDirectory::getTable (uint32_t *index*) Returns the **PageTable** (p. 689) with a specified index.

Parameters

index	The index of the **PageTable** (p. 689).

@

@plus@

@plus -@

@

@

@skip

Returns

The **PageTable** (p. 689) associated with passed index.

bool memory::paging::PageDirectory::map (void ∗ *linAddr,* PageFlags const & *flags* = PageFlags()) Maps the page at some linear address to physical memory.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

linAddr	The linear address to be mapped.
flags	The page flags to use.

@

@plus@

@plus -@

@

@

@skip

Returns

True if an available page frame could be found, else false.

Referenced by map().

void memory::paging::PageDirectory::unmap (void ∗ *linAddr*) Unmaps the page at some linear address from physical memory.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

linAddr	The linear address to be unmapped.

Referenced by unmap().

void memory::paging::PageDirectory::map (void * *startLinAddr,* void * *endLinAddr,* void * *startPhysAddr,* PageFlags const & *flags* = PageFlags()) [inline] Maps a linear address range to physical memory starting at a specified physical address.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

startLinAddr	The start address of the linear address range (inclusive).
endLinAddr	The end address of the linear address range (exclusive).
startPhysAddr	The start address of the physical address range to use for the mapping.
flags	The page flags to use.

References map().

void memory::paging::PageDirectory::unmap (void * *startLinAddr,* void * *endLinAddr*) [inline] Unmaps a linear address range from physical memory.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

startLinAddr	The start address of the linear address range (inclusive).
endLinAddr	The end address of the linear address range (exclusive).

References unmap().

static uint32_t memory::paging::PageDirectory::mapLinearAddressToPageIndex (void * *linAddr*) [inline], [static], [private] Maps a linear address to a page entry index.

Parameters

linAddr	The address to map.

@

@plus@

@plus -@

@

@

@skip

Returns

The index of the **PageEntry** (p. 682).

Referenced by getEntryForAddress(), getOrAllocPageTable(), and getTable().

PageTable∗ memory::paging::PageDirectory::getOrAllocPageTable (void ∗ *linAddr*) `[inline]`**,** `[private]` Retrieves the **PageTable** (p. 689) for an address.

If no such page table exists (which can only happen for addresses in the user space), a new one is created. WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

linAddr	The linear address for which the **PageTable** (p. 689) is to be retrieved.

@

@plus@

@plus -@

@

@

@skip

Returns

The **PageTable** (p. 689) responsible for the address.

References mapLinearAddressToPageIndex().

PageTable* memory::paging::PageDirectory::getOrAllocPageTable (uint32_t *index*) [private] Retrieves the index-th **PageTable** (p. 689).

If no such page table exists (which can only happen for addresses in the user space), a new one is created. WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

index	The index of the **PageTable** (p. 689).

@

@plus@

@plus -@

@

@

@skip

Returns

The **PageTable** (p. 689) associated with passed index.

void memory::paging::PageDirectory::freePageTable (uint32_t *index*, bool *ifEmpty*) [private] Frees a user space page table.

WARNING: Cannot be called through **getCurrent()** (p. 681)!

Parameters

index	The index of the **PageTable** (p. 689) to be freed.
ifEmpty	Free only if page table is empty.

bool memory::paging::PageDirectory::checkIfUnmappable (void * *linAddr*) [private] Returns true if passed linear address may not be mapped (because it is already in use by the system).

This checks against some predefined address ranges created at boot time. If this operation returns false, this does not necessarily mean that the page at passed address is not in use!

Parameters

linAddr	The linear address to check.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the linear address falls into a reserved range, else false.

static PageDirectory* memory::paging::PageDirectory::getCurrent ( ) [inline], [static], [private] Returns the currently active **PageDirectory** (p. 672).

As the "process" member cannot be read through the pointer returned, only a small subset of operations may be called on the resulting object, namely getEntry..() and getTable...(), the latter only for the kernel address space.

void memory::paging::PageDirectory::setProcess (::task::Process * *process*) [inline], [private] Sets the process associated with this **PageDirectory** (p. 672).

Used by the BootManager to associate the initial **PageDirectory** (p. 672) with the system process.

Parameters

process	The process.

References process.

Field Documentation

::task::Process* memory::paging::PageDirectory::process [private] The associated process.

Note that you cannot access this field when going through **getCurrent()** (p. 681) – you need to know the proper linear address of the **PageDirectory** (p. 672) to access this field!

Referenced by setProcess().

The documentation for this class was generated from the following file:

- PageDirectory.h

A.4.214 memory::paging::PageEntry Class Reference

Public Member Functions

- bool **enableEntry** (**PageFlags** const &flags)

  Den aktuellen Eintrag aktivieren und für die zugehörige Seite speicher reservieren.

- void **enableEntry** (void ∗physAddr, **PageFlags** const &flags)

  Den aktuellen Eintrag aktivieren.

- void **disableEntry** ()

  Eintrag deaktivieren und den zugehörigen Speicher freigeben.

- void ∗ **getPhysicalAddress** () const

  Setzen oder abfragen der physikalischen Adresse auf den dieser Eintrag verweist.

- void **setPhysicalAddress** (void ∗)
- bool **isEnabled** () const

  Zeigt an ob der Eintrag aktiv ist.

- void **setPresent** ()

  Setzen des Present Flags.

- void **clearPresent** ()

  Löschen des Present Flags.

- void **setRW** ()

  Methoden zum Setzen, Löschen und Auslesen des R/W Flags.

- void **clearRW** ()
- bool **isRW** () const
- void **setUS** ()

Methoden zum Setzen, Löschen und Auslesen des U/S Flags.

- void **clearUS** ()
- bool **isUS** () const
- **PageFlags getFlags** () const

  *Returns a **PageFlags** (p. 685) object reflecting this entry's flags.*

Private Member Functions

- void **doEnableEntry** (void ∗physAddr, **PageFlags** const &flags)

  Enables this entry.

Private Attributes

- uint32_t **value**

Member Function Documentation

bool memory::paging::PageEntry::enableEntry (PageFlags const & *flags*) Den aktuellen Eintrag aktivieren und für die zugehörige Seite speicher reservieren.

@

@plus@

@plus -@

@

@

@skip
Returns

true, falls Speicher reserviert werden konnte

void memory::paging::PageEntry::enableEntry (void ∗ *physAddr,* PageFlags const & *flags*) Den aktuellen Eintrag aktivieren.

@

@plus@

@plus -@

@

@

@skip
Returns

true, falls Speicher reserviert werden konnte

bool memory::paging::PageEntry::isEnabled ( ) const [inline] Zeigt an ob der Eintrag aktiv ist.

@

@plus@

@plus -@

@

@

@skip
Returns

true falls aktiv.

void memory::paging::PageEntry::doEnableEntry (void * *physAddr*, PageFlags const & *flags*) [private] Enables this entry.

Parameters

physAddr	The physical address to use.
flags	The page flags to use.

The documentation for this class was generated from the following file:

- PageEntry.h

A.4.215 memory::paging::PageFaultExceptionHandler Class Reference

Handles page fault exceptions.

Inherits **cpu::interrupt::InterruptHandler**.

Public Member Functions

- virtual bool **handle** (uint32_t)

  Handles the interrupt.

Detailed Description

Handles page fault exceptions.

Member Function Documentation

virtual bool memory::paging::PageFaultExceptionHandler::handle (uint32_t *errorCode*) `[virtual]` Handles the interrupt.

Has to be implemented by the user.

Parameters

errorCode	An optional error code passed to some exceptions.

@

@plus@

@plus -@

@

@

@skip

Returns

boolean that indicates whether the handling of the interrupt was successful

Implements **cpu::interrupt::InterruptHandler** (p. 567).

The documentation for this class was generated from the following file:

- **PageFaultExceptionHandler.h**

A.4.216 memory::paging::PageFlags Class Reference

Encapsulates flags for a page.

Public Member Functions

- **PageFlags** ()

  Constructor.
- bool **isWritable** () const
- bool **isUserAccessible** () const
- **PageFlags** & **makePageWritable** (bool **writable**)

  Makes this page writable or read-only.
- **PageFlags** & **makePageUserAccessible** (bool **userAccessible**)

  Makes this page accessible by user or supervisor tasks.

Private Attributes

- bool **writable**

  True if page can be written.
- bool **userAccessible**

  True if page can be accessed by user-mode tasks.

Detailed Description

Encapsulates flags for a page.

Constructor & Destructor Documentation

memory::paging::PageFlags::PageFlags ( ) `[inline]` Constructor.

The flags are set to describe a writable page accessible only by supervisor tasks.

Member Function Documentation

bool memory::paging::PageFlags::isWritable ( ) const `[inline]` @

@plus@

@plus -@

@

@

@skip
Returns

True if this page is writable, else false.

References writable.

bool memory::paging::PageFlags::isUserAccessible ( ) const [inline]
@

@plus@

@plus -@

@

@

@skip
Returns

True if this page is accessible by user-mode tasks, else false.

References userAccessible.

PageFlags& memory::paging::PageFlags::makePageWritable (bool *writable*) [inline] Makes this page writable or read-only.

Parameters

writable	If true, the page will be writable. If false, it will be read-only.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References writable.

Referenced by memory::paging::PageEntry::getFlags().

PageFlags& memory::paging::PageFlags::makePageUserAccessible (bool *userAccessible*) [inline] Makes this page accessible by user or supervisor tasks.

Parameters

user-Accessible	If true, the page will be accessible by user-mode tasks running in ring 3. If false, it will be only accessible by supervisor tasks running in rings 0-2.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References userAccessible.

Referenced by memory::paging::PageEntry::getFlags().

The documentation for this class was generated from the following file:

- PageFlags.h

A.4.217 memory::allocator::PageHeader Struct Reference

Put at the beginning of a managed page.

Public Member Functions

- **PageHeader** (**Allocator** &**allocator**, **Page** &**page**)

  Constructor.

Data Fields

- **Allocator** * **allocator**

  The allocator used.

- **Page** ∗ **page**

  *The **Page** (p. 663) managing this page.*

Detailed Description

Put at the beginning of a managed page.

Its size must be less than or equal to **Page::PartSize** (p. 668)!

Constructor & Destructor Documentation

memory::allocator::PageHeader::PageHeader (Allocator & *allocator*, Page & *page*) [inline] Constructor.

Parameters

allocator	The allocator responsible for the underlying page.

The documentation for this struct was generated from the following file:

- **Page.h**

A.4.218 memory::paging::PageTable Class Reference

Public Member Functions

- bool **isEmpty** () const

  Returns true if the page table is empty.

- **PageEntry** ∗ **getEntryForAddress** (void ∗lineareAddress)

  Liefert den PageTableEntry für die angegebene Adresse zurück.

- **PageEntry** ∗ **getEntryWithOffset** (unsigned int offset)

  Liefert den (.

Private Attributes

- **PageEntry entries** [ENTRIES_IN_TABLE_OR_DIR]

Member Function Documentation

PageEntry∗ memory::paging::PageTable::getEntryForAddress (void ∗ *lineareAddress*) Liefert den PageTableEntry für die angegebene Adresse zurück.

Parameters

lineare-Address	die linaere Adresse für die der Eintrag gesucht werden soll

@

@plus@

@plus -@

@

@

@skip

Returns

Der gewünschte Eintrag in der **PageTable** (p. 689)

PageEntry∗ memory::paging::PageTable::getEntryWithOffset (unsigned int *offset*) Liefert den (.

Parameters

offset	)x'ten PageTableEntry aus dieser **PageTable** (p. 689)

The documentation for this class was generated from the following file:

- kernel/memory/paging/PageTable.h

A.4.219 memory::allocator::PageTable Class Reference

PageTables constitute the first-order organization structure of an **Allocator** (p. 286).

Public Member Functions

- bool **isEmpty** () const

  Returns true if this table is empty, i.e.

- bool **isFull** () const

  Returns true if this table is full, i.e.

- char ∗ **allocate** (**Allocator** &allocator, uint32_t size, bool &noPhysMemory)

  Allocates a block of memory.

- bool **free** (**Allocator** &allocator, char ∗address, **Page** ∗page)

  Frees a previously allocated block of memory.

- void **clear** (**Allocator** &allocator)

  Invalidates all managed Pages.

- **PageTable** ∗ **getNext** () const

  *Returns the following **PageTable** (p. 690).*

- void **setNext** (**PageTable** ∗**next**)

  *Sets the following **PageTable** (p. 690).*

Static Public Member Functions

- static **PageTable** ∗ **create** (**Allocator** &allocator, **PageTable** ∗**next**)

  *Creates a **PageTable** (p. 690).*

- static void **destroy** (**Allocator** &allocator, **PageTable** ∗table)

  *Destroys a **PageTable** (p. 690).*

Private Member Functions

- **PageTable** (**PageTable** ∗**next**)

  Constructor.

Private Attributes

- **PageTable** ∗ **next**

  *Points to the next **PageTable** (p. 690).*

- uint32_t **numValid**

  The number of valid Pages.

- **Page pages** [**NumEntries**]

  *The Pages this **PageTable** (p. 690) manages.*

Static Private Attributes

- static uint32_t const **NumEntries**

  *Number of Pages per **PageTable** (p. 690) such that a **PageTable** (p. 690) fits into one page.*

Detailed Description

PageTables constitute the first-order organization structure of an **Allocator** (p. 286).

PageTables contain a number of Pages and are part of a singly-linked list. PageTables are created and destroyed as needed.

Constructor & Destructor Documentation

memory::allocator::PageTable::PageTable (PageTable ∗ *next*) [inline], [private] Constructor.

Parameters

next	The next **PageTable** (p. 690) in the chain.

Member Function Documentation

static PageTable∗ memory::allocator::PageTable::create (Allocator & *allocator*, PageTable ∗ *next*) [static] Creates a **PageTable** (p. 690).

Parameters

allocator	The underlying allocator.
next	The next **PageTable** (p. 690) in the chain.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the new **PageTable** (p. 690) or NULL if not enough memory is available.

static void memory::allocator::PageTable::destroy (Allocator & *allocator,* PageTable * *table*) [static] Destroys a **PageTable** (p. 690).

The table is not cleared beforehand, so you have to ensure that it is empty.

Parameters

allocator	The underlying allocator.
table	The table to destroy.

bool memory::allocator::PageTable::isEmpty ( ) const [inline] Returns true if this table is empty, i.e.

if it does not contain any valid **Page** (p. 663).

References numValid.

bool memory::allocator::PageTable::isFull ( ) const [inline] Returns true if this table is full, i.e.

if it does not contain any invalid **Page** (p. 663).

References NumEntries, and numValid.

char* memory::allocator::PageTable::allocate (Allocator & *allocator,* uint32_t *size,* bool & *noPhysMemory*) Allocates a block of memory.

Pages and/or more PageTables are created if necessary.

Parameters

allocator	The underlying allocator.
size	The number of bytes to allocate. This must not exceed **Page::getMaxBlockSize()** (p. 664).

noPhysMemory	If set after returning to the caller, the physical memory has run out.

@

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the allocated memory or NULL if not enough memory is available.

bool memory::allocator::PageTable::free (Allocator & *allocator,* char * *address,* Page * *page*) Frees a previously allocated block of memory.

Parameters

allocator	The underlying allocator.
address	A pointer to the memory block to be freed.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the memory block could be freed, false otherwise (e.g. because the pointer passed was invalid).

void memory::allocator::PageTable::clear (Allocator & *allocator*) Invalidates all managed Pages.

After this operation, **isEmpty()** (p. 693) returns true.

Parameters

allocator	The underlying allocator.

void memory::allocator::PageTable::setNext (PageTable * *next*) [inline]

Sets the following **PageTable** (p. 690).

Parameters

next	The following **PageTable** (p. 690).

References next.

Field Documentation

uint32_t const memory::allocator::PageTable::NumEntries [static], [private]

Initial value:

=

```
(PAGE_SIZE - (sizeof(PageTable *) + sizeof(uint32_t)))
        / sizeof(Page)
```

Number of Pages per **PageTable** (p. 690) such that a **PageTable** (p. 690) fits into one page.

Referenced by isFull().

The documentation for this class was generated from the following file:

- **library/memory/allocator/PageTable.h**

A.4.220 memory::paging::test::PagingTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char * **getName** ()

Private Member Functions

- void **testPageEntrySetAddress** ()
- void **testPageEntrySetPresent** ()
- void **testTablePageGetEntries** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- PagingTestCase.h

A.4.221 fosCli::parser::Parser Class Reference

Inherits **fosCli::scanner::elements::FosCliElementVisitor**.

Public Member Functions

- **Parser** (**tool::collection::List**< **scanner::elements::FosCliElement** * > *elements)
- void **parse** ()
- virtual void **handle** (**fosCli::scanner::elements::FosCliStringElement** *element)

Private Attributes

- **tool::collection::List** < **scanner::elements::FosCliElement** * > * **input**
- bool **first**
- **CliCommand** * **command**

Additional Inherited Members

The documentation for this class was generated from the following file:

- Parser.h

A.4.222 memory::allocator::Page::PartDescriptor Class Reference

If valid, a **PartDescriptor** (p. 697) describes a contiguous range of parts that are either allocated or free.

Public Types

- enum **Type** { **Unused** = 0, **Allocated** = 1, **Free** = 2 }

  The type of a part descriptor.

Public Member Functions

- **Type getType** () const

  Returns the type of this part descriptor.
- void **setType** (**Type type**)

  Sets the type of this part descriptor.
- uint32_t **getCount** () const

  Returns the count of this part descriptor.
- void **setCount** (uint32_t count)

  Sets the count of this part descriptor.

Private Attributes

- uint8_t **value**

Detailed Description

If valid, a **PartDescriptor** (p. 697) describes a contiguous range of parts that are either allocated or free.

To reduce the need for moving PartDescriptors too often, a **PartDescriptor** (p. 697) can be marked as unused.

Member Enumeration Documentation

enum memory::allocator::Page::PartDescriptor::Type The type of a part descriptor.

Enumerator

Unused Entry is unused. Don't read/write the count in this case.

Allocated Entry describes an allocated memory block.

Free Entry describes a free memory block.

Member Function Documentation

void memory::allocator::Page::PartDescriptor::setType (Type *type*) [inline] Sets the type of this part descriptor.

This operation does not change the count unless you set the type to Unused.

Parameters

type	The type to set.

References Allocated, Free, memory::allocator::Page::TypeBit, and Unused.

void memory::allocator::Page::PartDescriptor::setCount (uint32_t *count*) [inline] Sets the count of this part descriptor.

Only allowed if the type is not Unused. This operation does not change the type.

Parameters

count	The count to set.

References memory::allocator::Page::NumParts, memory::allocator::Page::PartCountMask, and memory::allocator::Page::TypeBit.

The documentation for this class was generated from the following file:

- **Page.h**

A.4.223 ipc::Participant Class Reference

Represents an IPC participant.

Public Member Functions

- **Participant** ()

  Default constructor.

- **Participant** (uint32_t processId)

  Initializes the object from a process identifier.

- uint32_t **getProcessId** () const

  Returns the participant's process identifier.

Private Attributes

- uint32_t **processId**

Detailed Description

Represents an IPC participant.

Constructor & Destructor Documentation

ipc::Participant::Participant (uint32_t *processId*) `[inline]`, `[explicit]`
Initializes the object from a process identifier.

Parameters

processId	The process identifier.

The documentation for this class was generated from the following file:

- **Participant.h**

A.4.224 memory::pmm::PhysicalMemoryManager Class Reference

Manages the physical memory.

Public Member Functions

- void **includeRegion** (physical_addr, uint32_t)

includes a physical memory region for use

- bool **allocBlock** (void ∗&result)

  Reserves a free block and returns its physical address.

- void **useBlock** (void ∗physAddr)

  Increments the usage counter for an already allocated block.

- void **freeBlock** (void ∗physAddr)

  Decrements the usage counter for an already allocated block.

- bool **allocBlocks** (uint32_t numPages, void ∗&result)

  allocates blocks of memory

- void **freeBlocks** (void ∗, uint32_t)

  free blocks of memory

- uint32_t **getMemorySize** () const

  returns amount of physical memory the manager is set to use

- uint32_t **getUsedBlockCount** () const

  returns number of blocks currently in use

- uint32_t **getFreeBlockCount** () const

  returns number of blocks not in use

- uint32_t **getBlockCount** () const

  returns number of memory blocks

- void ∗ **getStartAddressAfterPmm** () const

  returns startAdressAfterPmm

Static Public Member Functions

- static **PhysicalMemoryManager** & **get** ()

  Returns a reference to the physical memory manager.

Private Member Functions

- **PhysicalMemoryManager** (**paging::PageDirectory** ∗initDir, **E820_entry** const ∗mmEntry, char ∗linStartPMM, char ∗physStartPMM)

  Constructor.

- void **allocateMemory** (uint32_t pos)

  mark the frame at pos as allocated.

- bool **freeMemory** (uint32_t pos)

  mark the frame at pos as free.

- bool **isFree** (uint32_t pos) const

  test any pmm_table_entry/frame within pmm_table

- bool **findNextFree** (uint32_t &frame)

  finds next free frame in the pmm_table_entry/frame and returns its index

- bool **findFirstFree** (uint32_t &frame)

  finds first free frame in the pmm_table_entry/frame and returns its index

- bool **findFirstFreeArea** (uint32_t size, uint32_t &startFrame)

  finds first free "size" number of frames and returns its index

- uint32_t **getFreeStartSearch** ()

  returns the start of the search

- void **setFreeStartSearch** (uint32_t value)

  sets the start for the search

- void **freeUnusedMemory** (**paging::PageDirectory** &pageDir, **E820_entry** const ∗mmEntry)

  Unmaps memory that is not used by the kernel but has been mapped by the OS loader.

Static Private Member Functions

- static void **init** (**paging::PageDirectory** ∗initDir, **E820_entry** const ∗mmEntry, char ∗linStartPMM, char ∗physStartPMM)

  Initializes the physical memory manager.

- static physical_addr **getMemorySize** (const **E820_entry** ∗mmEntry)

  return the memory size of the system

Private Attributes

- uint32_t **memorySize**

  size of physical memory

- void * **firstAddressAfterStructures**

  first address after structure

- uint32_t **usedBlocks**

  number of blocks currently in use

- uint32_t **maxBlocks**

  maximum number of available memory blocks

- uint32_t **freeStartSearch**

  position from which the next free block is searched. gets reseted to a freed page, if the free page is lower.

- **PmmTableEntry * tableEntries**

  Structure containing the information about the accessed pages 4GB = 262144 entries.

- **Logger logger**

  enable printing of pmm infos.

Static Private Attributes

- static char **instance** [ ]

Friends

- class **boot::BootManager**

Detailed Description

Manages the physical memory.

Constructor & Destructor Documentation

memory::pmm::PhysicalMemoryManager::PhysicalMemoryManager (paging::PageDirectory * *initDir,* E820_entry const * *mmEntry,* char * *linStartPMM,* char * *physStartPMM*) [private] Constructor.

Parameters

initDir	The initial page directory.
mmEntry	Points to the memory information chain provided by the BIOS.
linStartPMM	The linear address where to put the PMM table.
physStartPMM	The physical address where to put the PMM table.

Referenced by init().

Member Function Documentation

static void memory::pmm::PhysicalMemoryManager::init (paging::PageDirectory * *initDir*, E820_entry const * *mmEntry*, char * *linStartPMM*, char * *physStartPMM*) `[inline]`, `[static]`, `[private]` Initializes the physical memory manager.

Parameters

initDir	The initial page directory.
mmEntry	Points to the memory information chain provided by the BIOS.
linStartPMM	The linear address where to put the PMM table.
physStartPMM	The physical address where to put the PMM table.

References PhysicalMemoryManager().

void memory::pmm::PhysicalMemoryManager::useBlock (void * *physAddr*) Increments the usage counter for an already allocated block.

Parameters

physAddr	The physical address of the memory block.

void memory::pmm::PhysicalMemoryManager::freeBlock (void * *physAddr*) Decrements the usage counter for an already allocated block.

If it reaches zero, the block is freed.

Parameters

physAddr	The physical address of the memory block.

bool memory::pmm::PhysicalMemoryManager::freeMemory (uint32_t *pos*) [private] mark the frame at pos as free.

Returns true if the usage counter has reached zero.

void memory::pmm::PhysicalMemoryManager::freeUnusedMemory (paging::PageDirectory & *pageDir*, E820_entry const * *mmEntry*) [private]
Unmaps memory that is not used by the kernel but has been mapped by the OS loader.

Parameters

pageDir	The PageDirectory to use.
mmEntry	The BIOS memory chain.

The documentation for this class was generated from the following file:

- PhysicalMemoryManager.h

A.4.225 io::driver::interrupt::PICController Class Reference

Inherits **io::driver::interrupt::PICDevice**.

Inherited by **io::driver::interrupt::MasterPICController**, and **io::driver::interrupt::SlavePICController**.

Public Member Functions

- **PICController** (int deviceId, uint16_t basePort)
- virtual void **init** (uint8_t **vectorBase**, cpu::level::IRQLevel baseIRQL)=0
- virtual uint32_t **getId** () const

  Returns the PIC's unique identifier.
- virtual uint8_t **getNumberOfInputs** ()

  Returns the number of IRQ inputs.
- virtual bool **add** (uint8_t pin, **IRQHandler** *i)

Adds an IRQ handler to the beginning of the list of IRQ handlers.

- virtual bool **remove** (uint8_t pin, **IRQHandler** ∗i)

  Removes an IRQ handler from the list of IRQ handlers.

Protected Member Functions

- **IOPort** & **getControlPort** ()
- **IOPort** & **getDataPort** ()
- void **initHardware** (uint8_t **vectorBase**)

  Initializes the PIC.
- void **createIRQHandler** (uint8_t pin, cpu::level::IRQLevel irql)
- virtual void **sendEOI** ()=0

  Sends an EOI to this PIC and its master PIC, if any.
- void **doEOI** ()

  Sends an EOI command to this PIC.
- uint8_t **getIRQBaseMask** () const

  Returns the base IRQ mask.
- void **enablePin** (uint8_t pin)

  Enables an IRQ pin.
- void **disablePin** (uint8_t pin)

  Disables an IRQ pin.

Static Protected Attributes

- static int const **NumInputs** = 8

  Number of inputs in a PIC.

Private Member Functions

- virtual void **writeICW3** ()=0

  Writes the ICW3.
- void **deleteIRQHandlers** ()

  Deletes the IRQ handlers.

Private Attributes

- **IOPort controlPort**

  The PIC's control port.

- **IOPort dataPort**

  The PIC's data port.

- uint8_t **vectorBase**

  The vector base.

- **PICInterruptHandler** ∗ **handlers** [**NumInputs**]

  The IRQ handlers.

- uint8_t **irqBaseMask**

  The IRQ base mask. A bit set means that no handler has been installed.

Static Private Attributes

- static uint8_t const **PIC_CMD_EOI** = 0x20

  The PIC's End-of-Interrupt command.

Friends

- class **PICInterruptHandler**

Member Function Documentation

virtual bool io::driver::interrupt::PICController::add (uint8_t *pin,* IRQHandler ∗ *i*) `[virtual]` Adds an IRQ handler to the beginning of the list of IRQ handlers.

Parameters

i	IRQ handler to be added to the list of IRQ handlers.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the handler has been installed successfully, else false. For example, installing a handler fails if the pin is out of range or if it is associated with a slave PIC.

Implements **io::driver::interrupt::PICDevice** (p. 710).

virtual bool io::driver::interrupt::PICController::remove (uint8_t *pin,* IRQHandler * *i*) [virtual] Removes an IRQ handler from the list of IRQ handlers.

Parameters

i	IRQ handler to be removed from the list of IRQ handlers.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the handler has been uninstalled successfully, else false. For example, uninstalling a handler fails if the pin is out of range, if it is associated with a slave PIC, or if the handler has not been installed using **add()** (p. 706) before.

Implements **io::driver::interrupt::PICDevice** (p. 710).

void io::driver::interrupt::PICController::initHardware (uint8_t *vectorBase*) [protected] Initializes the PIC.

Parameters

vectorBase	The number of the interrupt vector IRQ 0 of this PIC is mapped to.

void io::driver::interrupt::PICController::enablePin (uint8_t *pin*) [protected]

Enables an IRQ pin.

This is done if a slave PIC is associated with the pin (to allow the CPU to handle slave PIC interrupts) or if an IRQ handler is associated with the pin's **PICInterruptHandler** (p. 712).

Parameters

pin	The pin to enable.

void io::driver::interrupt::PICController::disablePin (uint8_t *pin*) [protected]

Disables an IRQ pin.

This is done if a slave PIC is disassociated from the pin or if the last IRQ handler is removed from the pin's **PICInterruptHandler** (p. 712).

Parameters

pin	The pin to disable.

The documentation for this class was generated from the following file:

- PICController.h

A.4.226 io::driver::interrupt::PICDevice Class Reference

Top Class for both PIC Devices (Primary and Slave)

Inherits **io::driver::Device**.

Inherited by **io::driver::interrupt::PICController**.

Public Member Functions

- **PICDevice** (int deviceId)
- virtual uint32_t **getId** () const =0

  Returns the PIC's unique identifier.
- virtual uint8_t **getNumberOfInputs** ()=0

Returns the number of IRQ inputs.

- virtual cpu::level::IRQLevel **getLevel** (uint8_t pin)=0

  Returns the IRQ level associated with a PIC input.

- virtual void **maskIRQs** (cpu::level::IRQLevel untilIRQL)=0

  Masks all IRQs up to a given IRQ level (including).

- virtual bool **add** (uint8_t pin, **IRQHandler** ∗i)=0

  Adds an IRQ handler to the beginning of the list of IRQ handlers.

- virtual bool **remove** (uint8_t pin, **IRQHandler** ∗i)=0

  Removes an IRQ handler from the list of IRQ handlers.

Detailed Description

Top Class for both PIC Devices (Primary and Slave)

Member Function Documentation

virtual cpu::level::IRQLevel io::driver::interrupt::PICDevice::getLevel (uint8_t *pin*) `[pure virtual]` Returns the IRQ level associated with a PIC input.

Parameters

pin	The PIC input.

@

@plus@

@plus -@

@

@

@skip

Returns

The associated IRQ level.

Implemented in **io::driver::interrupt::MasterPICController** (p. 653), and **io::driver::interrupt::SlavePICController** (p. 822).

virtual void io::driver::interrupt::PICDevice::maskIRQs (cpu::level::IRQLevel *untilIRQL*) [pure virtual] Masks all IRQs up to a given IRQ level (including).

All higher IRQLs are unmasked.

Parameters

untilIRQL	The IRQ level up to which IRQs are to be masked.

Implemented in **io::driver::interrupt::MasterPICController** (p. 652), and **io::driver::interrupt::SlavePICController** (p. 822).

Referenced by io::driver::interrupt::MaskingHandler::lower(), and io::driver::interrupt::MaskingHandler::raise().

virtual bool io::driver::interrupt::PICDevice::add (uint8_t *pin,* IRQHandler * *i*) [pure virtual] Adds an IRQ handler to the beginning of the list of IRQ handlers.

Parameters

i	IRQ handler to be added to the list of IRQ handlers.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the handler has been installed successfully, else false. For example, installing a handler fails if the pin is out of range or if it is associated with a slave PIC.

Implemented in **io::driver::interrupt::PICController** (p. 706).

virtual bool io::driver::interrupt::PICDevice::remove (uint8_t *pin,* IRQHandler * *i*) [pure virtual] Removes an IRQ handler from the list of IRQ handlers.

Parameters

i	IRQ handler to be removed from the list of IRQ handlers.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the handler has been uninstalled successfully, else false. For example, uninstalling a handler fails if the pin is out of range, if it is associated with a slave PIC, or if the handler has not been installed using **add()** (p. 710) before.

Implemented in **io::driver::interrupt::PICController** (p. 707).

The documentation for this class was generated from the following file:

- **PICDevice.h**

A.4.227 io::driver::interrupt::PICDriver Class Reference

Driver (p. 455) for both PIC Devices.

Inherits **io::driver::Driver**.

Public Member Functions

- void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **MasterPICController** ∗ **masterPIC**
- **SlavePICController** ∗ **slavePIC**

Static Private Attributes

- static uint16_t const **PIC_BASE_MASTER** = 0x20

  Base port address of master PIC.

- static uint16_t const **PIC_BASE_SLAVE** = 0xA0

  Base port address of slave PIC.

- static uint8_t const **PIC_SLAVE_PIN** = 2

  Master pin the slave PIC is attached to.

Additional Inherited Members

Detailed Description

Driver (p. 455) for both PIC Devices.

The documentation for this class was generated from the following file:

- PICDriver.h

A.4.228 io::driver::interrupt::PICInterruptHandler Class Reference

Interrupt handler installed by the PIC devices that automatically send EOI commands to the relevant PICs.

Inherits **cpu::interrupt::InterruptHandler**.

Public Member Functions

- **PICInterruptHandler** (**PICController** &**pic**, uint8_t **irqNum**)
- void **add** (**IRQHandler** ∗i)

  Adds an IRQ handler to the beginning of the list of IRQ handlers.

- bool **remove** (**IRQHandler** ∗i)

  Removes an IRQ handler from the list of IRQ handlers.

- bool **isEmpty** () const

  Returns true if no IRQ handlers have been registered for this interrupt handler.

- virtual bool **handle** (uint32_t errorCode)

Handles the interrupt.

Private Types

- typedef
  tool::collection::LinkedList
  $<$ **IRQHandler** $*$ $>$ **Handlers**

  Type of a list of associated interrupt handlers.

Private Attributes

- **PICController** & **pic**

  The associated PIC.

- uint8_t const **irqNum**

  The IRQ number.

- **Handlers handlers**

  The associated interrupt handlers.

Detailed Description

Interrupt handler installed by the PIC devices that automatically send EOI commands to the relevant PICs.

Member Function Documentation

void io::driver::interrupt::PICInterruptHandler::add (IRQHandler $*$ ***i*** **)** [inline] Adds an IRQ handler to the beginning of the list of IRQ handlers.

Parameters

i	IRQ handler to be added to the list of IRQ handlers.

References tool::collection::LinkedList$<$ T $>$::add(), and handlers.

bool io::driver::interrupt::PICInterruptHandler::remove (IRQHandler $*$ ***i*** **)** [inline] Removes an IRQ handler from the list of IRQ handlers.

Parameters

i	IRQ handler to be removed from the list of IRQ handlers.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the handler has been removed successfully, else false.

References handlers, and tool::collection::LinkedList< T >::remove().

virtual bool io::driver::interrupt::PICInterruptHandler::handle (uint32_t *errorCode*) `[virtual]` Handles the interrupt.

Has to be implemented by the user.

Parameters

errorCode	An optional error code passed to some exceptions.

@

@plus@

@plus -@

@

@

@skip

Returns

boolean that indicates whether the handling of the interrupt was successful

Implements **cpu::interrupt::InterruptHandler** (p. 567).

The documentation for this class was generated from the following file:

- **PICInterruptHandler.h**

A.4.229 task::pipeandfilter::Pipe< T > Class Template Reference

Pipe (p. 715) for pipe and filter pattern.

Inherited by **task::pipeandfilter::test::TesterPipe**< **T** >.

Data Structures

- struct **Node**

Public Member Functions

- **Pipe** ()

  Constructor.
- bool **isEmpty** () const
- void **write** (T data)

  append the data to the end of the pipe
- T **read** ()

  Reads from the pipe.

Private Attributes

- **Node** ∗ **frontPtr**
- **Node** ∗ **backPtr**
- **task::lock::Semaphore semaphore**
- **task::lock::SpinLock mutex**

Detailed Description

template<**typename T**>**class task::pipeandfilter::Pipe**< **T** >

Pipe (p. 715) for pipe and filter pattern.

As reading from an empty pipe always blocks, reading is only allowed at IRQL == PASSIVE, while writing is allowed at IRQL <= DISPATCH.

Member Function Documentation

template<typename T > bool task::pipeandfilter::Pipe< T >::isEmpty ( ) const [inline] @

@plus@

@plus -@

@

@

@skip
Returns

true, if the **Pipe** (p. 715) is empty

template<typename T> void task::pipeandfilter::Pipe< T >::write (T *data*) append the data to the end of the pipe

Parameters

data	

template<typename T > T task::pipeandfilter::Pipe< T >::read ( )
Reads from the pipe.

Use at IRQL == PASSIVE only! @

@plus@

@plus -@

@

@

@skip
Returns

the first data in the pipe

The documentation for this class was generated from the following file:

- Pipe.h

A.4.230 io::driver::timer::PIT8254ControlWord Class Reference

This class produces the control word for the Programmable Interval Timer Intel 82c54.

Public Member Functions

- void **setCounter** (uint8_t counterNumber)

  Sets the counter.

- void **setRead** ()

  set read

- void **setWrite** ()

  set write

- void **setMode** (PIT8254CounterMode mode)

  set mode

- uint8_t **getControlWord** () const

  Returns the created control word.

Private Attributes

- uint8_t **controlword**

  Used for creation of the control word.

Detailed Description

This class produces the control word for the Programmable Interval Timer Intel 82c54.

Control Word: [_][_][_][_][_][_][_][_] |CNr ||R/W |Mode ||B|

CNr = Counter number (00, 01, 10) - 2 bit R/W = Read/Write (00 = Read, 11 = Write) - 2 bit Mode= Mode (000 - 101) - 3 bit B = BCD coding (0 = no bcd, 1 = bcd) - 1 bit

Member Function Documentation

void io::driver::timer::PIT8254ControlWord::setCounter (uint8_t *counterNumber*) Sets the counter.

Parameters

counterNumber	

void io::driver::timer::PIT8254ControlWord::setMode (PIT8254CounterMode *mode*) set mode

Parameters

mode	

uint8_t io::driver::timer::PIT8254ControlWord::getControlWord ( ) const Returns the created control word.

@

@plus@

@plus -@

@

@

@skip
Returns

The documentation for this class was generated from the following file:

- PIT8254ControlWord.h

A.4.231 io::driver::timer::PIT8254Counter Class Reference

Represents one counter of the Intel 82c54 Programmable Interval Timer.

Public Member Functions

- **PIT8254Counter** (**PIT8254Device** &**device**, uint8_t **counterNumber**)

  Constructor.

- virtual ~**PIT8254Counter** ()

  Destructor.

- void **release** ()

  Releases this counter.

- void **set** (PIT8254CounterMode counterMode, uint16_t counterValue)

  Sets this counter.

- uint16_t **get** () const

  Reads the current value of the counter.

Private Member Functions

- bool **acquire** ()

  Tries to reserve this counter for a caller.

Private Attributes

- **PIT8254Device** & **device**

  The underlying device.

- uint8_t const **counterNumber**

  Counter number.

- **task::lock::SpinLock mutex**

  Spinlock protecting access to Counter objects.

- bool **acquired**

  True if this counter has already been acquired.

Friends

- class **PIT8254Device**

Detailed Description

Represents one counter of the Intel 82c54 Programmable Interval Timer.

Constructor & Destructor Documentation

io::driver::timer::PIT8254Counter::PIT8254Counter (PIT8254Device & *device,* uint8_t *counterNumber*) Constructor.

Deactivates this counter.

Parameters

counterNumber	The number of this counter. (0-2)

virtual io::driver::timer::PIT8254Counter::∼PIT8254Counter ( ) [virtual] Destructor.

Deactivates this counter.

Member Function Documentation

void io::driver::timer::PIT8254Counter::release ( ) Releases this counter.

After this, a new caller can acquire this counter.

void io::driver::timer::PIT8254Counter::set (PIT8254CounterMode *counterMode,* uint16_t *counterValue*) [inline] Sets this counter.

Parameters

counterMode	The counter mode. See PIT8243CounterMode.
counterValue	The initial value for the counter.

References io::driver::timer::PIT8254Device::set().

uint16_t io::driver::timer::PIT8254Counter::get ( ) const [inline] Reads the current value of the counter.

@

@plus@

@plus -@

@

@

@skip
Returns

Current value.

References io::driver::timer::PIT8254Device::get().

bool io::driver::timer::PIT8254Counter::acquire ( ) [private] Tries to reserve this counter for a caller.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the counter could be acquired, false if the counter has already been acquired before.

The documentation for this class was generated from the following file:

- PIT8254Counter.h

A.4.232 io::driver::timer::PIT8254Device Class Reference

Implementation of the Intel 82c54 Programmable Interval Timer.

Inherits **io::driver::Device**.

Public Member Functions

- **PIT8254Device** (uint16_t **basePort**)

  Constructor.

- uint8_t **getNumberOfCounters** () const

  Returns the number of counters this PIT possesses.

- **PIT8254Counter** ∗ **acquireCounter** (uint8_t index)

  Acquires a counter.

Private Member Functions

- **PIT8254Counter** ∗ **acquireCounter** (**PIT8254Counter** &counter)

  Assigns a counter to the caller.

- void **set** (**PIT8254Counter** &counter, PIT8254CounterMode counterMode, uint16_t counterValue)

  Sets the mode and initial value of a counter.

- uint16_t **get** (**PIT8254Counter** const &counter) const

  Reads the current value of a counter.

Private Attributes

- **task::lock::SpinLock ioMutex**

  Spinlock preventing interleaved access to the control and data ports.

- uint16_t const **basePort**

  The base port of the PIT.

- **IOPort controlIO**

  The I/O port for the control word of the PIT.

- **PIT8254Counter** ∗ **counters** [**NumCounters**]

  The counters.

Static Private Attributes

- static const uint8_t **NumCounters** = 3

  Number of counters a PIT possesses.

Friends

- class **PIT8254Counter**

Detailed Description

Implementation of the Intel 82c54 Programmable Interval Timer.

Constructor & Destructor Documentation

io::driver::timer::PIT8254Device::PIT8254Device (uint16_t *basePort*)
Constructor.

Parameters

basePort	The base port of the PIT.

Member Function Documentation

PIT8254Counter* io::driver::timer::PIT8254Device::acquireCounter (uint8_t *index*) Acquires a counter.

Parameters

index	The index.

@

@plus@

@plus -@

@

@

@skip
Returns

The counter or NULL if the counter has already been acquired by someone else.

PIT8254Counter* io::driver::timer::PIT8254Device::acquireCounter (PIT8254Counter & *counter*) [private] Assigns a counter to the caller.

Parameters

counter	the counter that should be assigned

@

@plus@

@plus -@

@

@

@skip

Returns

the assigned counter

void io::driver::timer::PIT8254Device::set (PIT8254Counter & *counter,* PIT8254CounterMode *counterMode,* uint16_t *counterValue*) [private]

Sets the mode and initial value of a counter.

Parameters

counter	The counter.
counter-Mode	The counter mode. See PIT8243CounterMode.
counter-Value	The initial value for the counter.

Referenced by io::driver::timer::PIT8254Counter::set().

uint16_t io::driver::timer::PIT8254Device::get (PIT8254Counter const & *counter*) const [private] Reads the current value of a counter.

Parameters

counter	The counter.

@

@plus@

@plus -@

@

@

@skip

Returns

The counter's current value.

Referenced by io::driver::timer::PIT8254Counter::get().

The documentation for this class was generated from the following file:

- PIT8254Device.h

A.4.233 io::driver::timer::PIT8254Driver Class Reference

Driver (p. 455) for the Programmable Interval Timer Intel 82c54.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Static Private Attributes

- static uint16_t const **BasePITPort** = 0x40

  Base input/output port of the first Intel 82c54 in the system.

Additional Inherited Members

Detailed Description

Driver (p. 455) for the Programmable Interval Timer Intel 82c54.

This driver will load the **PIT8254Device** (p. 722).

The documentation for this class was generated from the following file:

- PIT8254Driver.h

A.4.234 memory::pmm::PmmTableEntry Struct Reference

Data Fields

- bool **free**: 1

  True if block is free, false otherwise.

- uint32_t **accessesCount**: 31

  Number of mapped linear addresses to the page.

Field Documentation

bool memory::pmm::PmmTableEntry::free True if block is free, false otherwise.

Referenced by memory::pmm::PhysicalMemoryManager::isFree().

uint32_t memory::pmm::PmmTableEntry::accessesCount Number of mapped linear addresses to the page.

The documentation for this struct was generated from the following file:

- PhysicalMemoryManager.h

A.4.235 io::driver::graphics::Position Class Reference

Represents a position.

Inherits **io::driver::graphics::Delta**.

Public Member Functions

- **Position** (int32_t x, int32_t y, uint32_t numCols)

  *Creates a **Position** (p. 727) object.*

- **Position** (int32_t **offset**)

  *Creates a **Position** (p. 727) object.*

- uint32_t **getColumn** (uint32_t numCols) const

  Returns the column (X coordinate).

- **Position** & **setColumn** (uint32_t x, uint32_t numCols)

Sets the column (X coordinate).

- int32_t **getRow** (uint32_t numCols) const

  Returns the row (Y coordinate).

- **Position** & **setRow** (uint32_t y, uint32_t numCols)

  Sets the row (Y coordinate).

- int32_t **getOffset** () const

  Returns the offset.

- bool **operator==** (**Position** pos) const

  Returns true if this position is equal to another one.

- bool **operator**< (**Position** pos) const

  Returns true if this position is less than another one.

- **Position** & **operator+=** (**Delta** delta)

  Adds a delta to this position.

- **Position** & **operator-=** (**Delta** delta)

  Subtracts a delta from this position.

- **Position operator+** (**Delta** delta) const

  Adds a delta to this position and returns the new position.

- **Position operator-** (**Delta** delta) const

  Subtracts a delta from this position and returns the new position.

- **Position** & **operator++** ()

  Adds a delta of 1 to this position.

- **Position** & **operator--** ()

  Subtracts a delta of 1 from this position.

- **Position operator++** (int)

  Adds a delta of 1 to this position and returns the old position.

- **Position operator--** (int)

  Subtracts a delta of 1 from this position and returns the old position.

- **Delta operator-** (**Position** pos) const

  Subtracts another position from this one.

Additional Inherited Members

Detailed Description

Represents a position.

The origin (0, 0) of the underlying coordinate system is the left top edge of the screen.

Constructor & Destructor Documentation

io::driver::graphics::Position::Position (int32_t *x*, int32_t *y*, uint32_t *numCols*) [inline] Creates a **Position** (p. 727) object.

Parameters

x	The column (X coordinate).
y	The row (Y coordinate).

io::driver::graphics::Position::Position (int32_t *offset*) [inline] Creates a **Position** (p. 727) object.

Parameters

offset	The position's offset from the origin.

Member Function Documentation

uint32_t io::driver::graphics::Position::getColumn (uint32_t *numCols*) const [inline] Returns the column (X coordinate).

Parameters

numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

The column. This column lies in the range 0 <= column < numCols.

References io::driver::graphics::Delta::getColumnDelta().

Referenced by setColumn().

Position& io::driver::graphics::Position::setColumn (uint32_t *x*, uint32_t *numCols*) [inline] Sets the column (X coordinate).

Parameters

x	The column (X coordinate).
numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

References getColumn(), getOffset(), and io::driver::graphics::Delta::setOffset().

int32_t io::driver::graphics::Position::getRow (uint32_t *numCols*) const [inline] Returns the row (Y coordinate).

Parameters

numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

The row delta.

References io::driver::graphics::Delta::getRowDelta().

Referenced by setRow().

Position& io::driver::graphics::Position::setRow (uint32_t *y*, uint32_t *numCols*) [inline] Sets the row (Y coordinate).

Parameters

y	The row (Y coordinate).
numCols	The length of a row.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this

References getOffset(), getRow(), and io::driver::graphics::Delta::setOffset().

bool io::driver::graphics::Position::operator== (Position *pos*) const [inline] Returns true if this position is equal to another one.

Parameters

pos	The position to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this position has the same offset as passed one, else false.

bool io::driver::graphics::Position::operator< (Position *pos*) const [inline] Returns true if this position is less than another one.

Parameters

pos	The position to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this position has a smaller offset as passed one, else false.

Position& io::driver::graphics::Position::operator+= (Delta *delta*) [inline] Adds a delta to this position.

Parameters

delta	The delta to add.

@

@plus@

@plus -@

@

@

@skip

Returns

*this.

References io::driver::graphics::Delta::getOffset(), getOffset(), and io::driver::graphics::Delta::setOffset().

Position& io::driver::graphics::Position::operator-= (Delta *delta*) [inline] Subtracts a delta from this position.

Parameters

delta	The delta to subtract.

@

@plus@

@plus -@

@

@

@skip

Returns

*this.

References io::driver::graphics::Delta::getOffset(), getOffset(), and io::driver::graphics::Delta::setOffset().

Position io::driver::graphics::Position::operator+ (Delta *delta*) const [inline] Adds a delta to this position and returns the new position.

Parameters

delta	The delta to add.

@

@plus@

@plus -@

@

@

@skip

Returns

The new position.

Position io::driver::graphics::Position::operator- (Delta *delta*) const [inline] Subtracts a delta from this position and returns the new position.

Parameters

delta	The delta to subtract.

@

@plus@

@plus -@

@

@

@skip

Returns

The new position.

Position& io::driver::graphics::Position::operator++ ( ) [inline] Adds a delta of 1 to this position.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this.

Position& io::driver::graphics::Position::operator-- ( ) [inline] Subtracts a delta of 1 from this position.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this.

Position io::driver::graphics::Position::operator++ (int) [inline] Adds a delta of 1 to this position and returns the old position.

@

@plus@

@plus -@

@

@

@skip

Returns

The old position.

Position io::driver::graphics::Position::operator-- (int) [inline] Subtracts a delta of 1 from this position and returns the old position.

@

@plus@

@plus -@

@

@

@skip

Returns

The old position.

Delta io::driver::graphics::Position::operator- (Position *pos*) const [inline] Subtracts another position from this one.

Parameters

pos	The position to subtract.

@

@plus@

@plus -@

@

@

@skip
Returns

A delta such that pos.add(delta) == ∗this.
The new position.

References io::driver::graphics::Delta::Delta(), and getOffset().

The documentation for this class was generated from the following file:

- **Position.h**

A.4.236 task::priorityinheritance::test::PriorityInheritanceTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- PriorityInheritanceTestCase.h

A.4.237 task::Process Class Reference

A **Process** (p. 737) represents a collection of Threads.

Data Structures

- class **Request**

  Encapsulates an IPC request and a Semaphore.

Public Member Functions

- ∼**Process** ()

  *Destructor for class **Process** (p. 737).*

- uint32_t **getProcessId** () const

  Returns the id of this process.

- **memory::paging::PageContext getContext** () const

  *Returns the PageContext of this **Process** (p. 737).*

- **Request addRequest** (**ipc::Request** ∗request)

  *Adds a **Request** (p. 777) to this process's request queue.*

- **Request fetchRequest** ()

  *Removes the first **Request** (p. 777) from the request queue and returns it.*

- **memory::lmm::LinearAddressSpaceManager** & **getLASM** ()

  *Returns the LinearAddressSpaceManager that manages this **Process** (p. 737)'s private linear address space.*

Static Public Member Functions

- static **Process** ∗ **createNewProcess** ()

  *Static method for creating a new **Process** (p. 737) object.*

Private Member Functions

- **Process** (**memory::paging::PageDirectory** ∗pageDirectory)

  *Private constructor for class **Process** (p. 737).*

- void **attachThread** (**Thread** ∗thread)

  Attaches a thread to this process.

- void **detachThread** (**Thread** ∗thread)

Detaches a thread from this process.

- void **initPageTableArray** ()

  Initializes the page table array.

- **memory::paging::PageTable** ∗& **getPageTable** (uint32_t index)

  Returns a reference to some PageTable for this process.

Static Private Member Functions

- static **Process** ∗ **createInitialProcess** (**memory::paging::PageDirectory** ∗dir)

  *Static method for creating the initial system **Process** (p. 737) object.*

Private Attributes

- **task::lock::SpinLock mutex**

  *Spinlock protecting the **Process** (p. 737)'s data.*

- uint32_t **identifier**

  Unique identifier which identifies this process.

- **tool::collection::LinkedList** < **Thread** ∗ > **threads**

  Contains all threads belonging to this process.

- **tool::collection::ArrayList** < **memory::paging::PageTable** ∗ > **pageTables**

  The process-specific page tables.

- **memory::lmm::LinearAddressSpaceManager lasm**

  Manages the private linear user address space of this process.

- **tool::collection::Queue**< **Request** > **requestQueue**

  Contains all requests a process has to handle.

- **task::lock::SpinLock requestMutex**

  Spin lock to protect the request queue.

- **task::lock::Semaphore requestSemaphore**

  Semaphore counting the objects in the request queue.

Friends

- class **task::Thread**
- class **memory::paging::PageDirectory**
- class **boot::BootManager**

Detailed Description

A **Process** (p. 737) represents a collection of Threads.

These Threads are arranged in different lists, depending on the state they have. Also a **Process** (p. 737) defines a PageContext in which its Threads live.

Constructor & Destructor Documentation

task::Process::Process (memory::paging::PageDirectory ∗ *pageDirectory*) [private] Private constructor for class **Process** (p. 737).

Please use the static method "createNewProcess". Threads can be placed later.

Parameters

page-Directory	The page directiry for this process.

Referenced by createInitialProcess(), and createNewProcess().

Member Function Documentation

void task::Process::attachThread (Thread ∗ *thread*) [inline], [private] Attaches a thread to this process.

Is called by **Thread** (p. 865)'s constructor.

Parameters

thread	The thread to attach.

References tool::collection::LinkedList< T >::add(), and threads.

void task::Process::detachThread (Thread ∗ *thread*) [inline], [private] Detaches a thread from this process.

Is called by **Thread** (p. 865)'s destructor. If this **Process** (p. 737) does not possess any threads after detaching the passed one, it is destroyed.

Parameters

thread	The thread to detach.

References tool::collection::LinkedList< T >::isEmpty(), tool::collection::LinkedList< T >::remove(), and threads.

static Process* task::Process::createInitialProcess (memory::paging::PageDirectory * *dir*) `[inline]`, `[static]`, `[private]` Static method for creating the initial system **Process** (p. 737) object.

Parameters

dir	The initial page directory.

@

@plus@

@plus -@

@

@

@skip

Returns

The newly created system **Process** (p. 737).

References Process().

memory::paging::PageTable*& task::Process::getPageTable (uint32_t *index*) `[inline]`, `[private]` Returns a reference to some PageTable for this process.

This is used by PageDirectory::getTable() to retrieve page tables that are specific for a process.

Parameters

index	The number of the page table to retrieve.

@

@plus@

@plus -@

@

@

@skip
Returns

The page table. It may be NULL if no page table with this index exists. In this case, the PageDirectory may set it to some valid PageTable.

References pageTables.

static Process∗ task::Process::createNewProcess ( ) [inline], [static]
Static method for creating a new **Process** (p. 737) object.

@

@plus@

@plus -@

@

@

@skip
Returns

The newly created **Process** (p. 737).

References Process().

Request task::Process::addRequest (ipc::Request ∗ *request*) Adds a **Request** (p. 777) to this process's request queue.

Parameters

request	The request to add.

Request task::Process::fetchRequest ( ) Removes the first **Request** (p. 777) from the request queue and returns it.

@

@plus@

@plus -@

@

@

@skip

Returns

The next **Request** (p. 777) to handle.

The documentation for this class was generated from the following file:

- Process.h

A.4.238 api::task::ProcessIdRequest Class Reference

Inherits **ipc::Request**.

Public Member Functions

- void **setProcessId** (uint32_t **id**)

Private Attributes

- uint32_t **ProcessId**

Static Private Attributes

- static const uint32_t **Id** = 9

Additional Inherited Members

The documentation for this class was generated from the following file:

- ProcessIdRequest.h

A.4.239 task::ProcessManager Class Reference

Public Member Functions

- **Process** & **getCurrentProcess** ()

  Returns the current process.

- **Process** ∗ **getProcess** (uint32_t processId)

*Returns a **Process** (p. 737) given a process identifier.*

- **Process** & **getSystemProcess** ()

  Returns the system process managing system threads.

Static Public Member Functions

- static **ProcessManager** & **getInstance** ()

  *Returns a reference to the **ProcessManager** (p. 744).*

Private Member Functions

- **ProcessManager** ()

  Constructor.

- uint32_t **attachProcess** (**Process** &process)

  Attaches a process.

- void **detachProcess** (**Process** &process)

  Detaches a process.

Static Private Member Functions

- static void **init** ()

  *Creates the **ProcessManager** (p. 744) object.*

Private Attributes

- **task::lock::SpinLock mutex**

  *Spinlock protecting the **ProcessManager** (p. 744)'s data.*

- **tool::collection::LinearMap**
  $<$ uint32_t, **Process** $*$ $>$ **processes**

  Maps identifiers to processes.

- uint32_t **nextProcessId**

  The next available process identifier.

Static Private Attributes

- static **ProcessManager** ∗ **theInstance**

  *The singleton instance of the **ProcessManager** (p. 744).*

- static uint32_t const **SystemProcessId** = 1

  The identifier of the system process.

Friends

- class **boot::BootManager**
- class **Process**

Member Function Documentation

Process∗ task::ProcessManager::getProcess (uint32_t *processId*) [inline] Returns a **Process** (p. 737) given a process identifier.

Parameters

processId	The **Process** (p. 737) identifier.

@

@plus@

@plus -@

@

@

@skip

Returns

The **Process** (p. 737) or NULL if not found.

References mutex, and processes.

Referenced by getSystemProcess().

uint32_t task::ProcessManager::attachProcess (Process & *process*) [inline], [private] Attaches a process.

Parameters

process	The process to attach.

References mutex, nextProcessId, and processes.

void task::ProcessManager::detachProcess (Process & *process*) [inline], [private] Detaches a process.

Parameters

process	The process to detach.

References task::Process::getProcessId(), mutex, and processes.

The documentation for this class was generated from the following file:

- **ProcessManager.h**

A.4.240 task::ProgramLoader Class Reference

The singleton class **ProcessManager** (p. 744) is responsible for loading an application.

Public Member Functions

- ∼**ProgramLoader** ()

  Destructor.

- bool **loadProgram** (void ∗ptr, uint32_t sizeInByte)

  Performs the loading of a program as described in the class description.

Static Public Member Functions

- static **ProgramLoader** & **getTheInstance** ()

  Static method to achieve the instance, as usual for a singleton.

Private Member Functions

- **ProgramLoader** ()

  Private constructor due to singleton pattern.

Static Private Attributes

- static bool **instanceFlag**

  Used to determine whether there is already an instance or not.

- static **ProgramLoader** ∗ **instance**

  contains the instance

Detailed Description

The singleton class **ProcessManager** (p. 744) is responsible for loading an application.

Therefore it needs a start pointer and the size of this application. Every application gets its own **Process** (p. 737) and an initial **Thread** (p. 865), that is already added to the Scheduler's list after the loading.

Member Function Documentation

static ProgramLoader& task::ProgramLoader::getTheInstance ( ) [static]
Static method to achieve the instance, as usual for a singleton.

@

@plus@

@plus -@

@

@

@skip
Returns

bool task::ProgramLoader::loadProgram (void ∗ ***ptr,*** **uint32_t** ***sizeInByte*** **)** Performs the loading of a program as described in the class description.

Parameters

ptr	as void∗ is the start address of the program to be loaded
sizeInByte	is the amount of bytes that will be copied to the new environment

@

@plus@

@plus -@

@

@

@skip

Returns

true, if no errors occurred during the load

The documentation for this class was generated from the following file:

- ProgramLoader.h

A.4.241 io::driver::pstwo::PS2Device Class Reference

Device (p. 436) for the PS2 Controller.

Inherits **io::driver::Device**.

Public Member Functions

- uint8_t **sendPS2Command** (uint8_t command)
- uint8_t **readBuffer** ()
- void **disableFirstPS2Port** ()

  Disables the first PS/2 Port.
- void **disableSecondPS2Port** ()

  Disabler the second PS_2 Port.
- void **enableFirstPS2Port** ()

  enable the first PS/2 Port
- void **enableSecondPS2Port** ()

  enable the second PS/2 Port

Private Types

- typedef **tool::BitField**
  < uint8_t, 0, 0, 1 > **KeyboardInterruptEnabledField**
- typedef **tool::BitField**
  < uint8_t, 0, 1, 1 > **AuxUnitInterruptEnabledField**
- typedef **tool::BitField**
  < uint8_t, 0, 5, 1 > **DualChannelField**
- typedef **tool::BitField**
  < uint8_t, 0, 6, 1 > **ScancodeTranslationField**

Private Member Functions

- void **sendKbdCommand** (uint8_t command)

  Send a KBD command to the Command Port.
- uint8_t **sendKbdCommandWithResponse** (uint8_t command)

  Send a KBD command to the Command Port and waits for an answer.
- void **sendKbdCommand** (uint8_t command, uint8_t nextByte)

  Send a KBD command to the Command Port with a second data byte to the KBC Data Port.
- uint8_t **readStatus** ()
- void **cleanOutputBuffer** ()
- bool **isOutputBufferEmpty** ()
- bool **isInputBufferEmpty** ()
- void **disableFirstPS2Interrupt** (uint8_t &controllerByte)

  sets in the
- void **disableSecondPS2Interrupt** (uint8_t &controllerByte)

  sets in the
- void **enableFirstPS2PortTranslation** (uint8_t &controllerByte)

  sets in the
- bool **hasDualChannel** (uint8_t controllerByte)
- uint8_t **getControllerConfigurationByte** ()
- void **sendControllerConfigurationByte** (uint8_t controllerByte)

  sends the

- bool **selfTestPassed** ()
- bool **port1TestPassed** ()
- bool **port2TestPassed** ()
- void **enableFirstPS2Interrupt** (uint8_t &controllerByte)

  sets in the
- void **enableSecondPS2Interrupt** (uint8_t &controllerByte)

  sets in the

Private Attributes

- **IOPort kbcDataIO**

  Access to the KBC Data Port.
- **IOPort kbcStatusAndCommandIO**

  Access to the KBC Status / Command Port.

Detailed Description

Device (p. 436) for the PS2 Controller.

Member Function Documentation

void io::driver::pstwo::PS2Device::sendKbdCommand (uint8_t *command*) [private] Send a KBD command to the Command Port.

Parameters

command	

Referenced by disableFirstPS2Port(), disableSecondPS2Port(), enableFirstPS2Port(), enableSecondPS2Port(), and sendControllerConfigurationByte().

uint8_t io::driver::pstwo::PS2Device::sendKbdCommandWithResponse (uint8_t *command*) [private] Send a KBD command to the Command Port and waits for an answer.

Parameters

command	

@

@plus@

@plus -@

@

@

@skip

Returns

the answer of the command

Referenced by getControllerConfigurationByte(), port1TestPassed(), port2TestPassed(), and selfTestPassed().

void io::driver::pstwo::PS2Device::sendKbdCommand (uint8_t *command*, uint8_t *nextByte*) [private] Send a KBD command to the Command Port with a second data byte to the KBC Data Port.

Parameters

command	
nextByte	

uint8_t io::driver::pstwo::PS2Device::readStatus ( ) [inline], [private]

@

@plus@

@plus -@

@

@

@skip

Returns

the status byte of the KBC Status Port

References kbcStatusAndCommandIO, and io::IOPort::read8().

Referenced by isInputBufferEmpty(), and isOutputBufferEmpty().

bool io::driver::pstwo::PS2Device::isOutputBufferEmpty ( ) [inline], [private] @

@plus@

@plus -@

@

@

@skip
Returns

true, if the output buffer is empty.

References readStatus().

bool io::driver::pstwo::PS2Device::isInputBufferEmpty ( ) [inline], [private] @

@plus@

@plus -@

@

@

@skip
Returns

true, if the input Buffer of the KBC Data Port is empty.

References readStatus().

void io::driver::pstwo::PS2Device::disableFirstPS2Interrupt (uint8_t & *controllerByte*) [inline], [private] sets in the

Parameters

controllerByte,that	the first PS/2 Interrupt is disabled.

References tool::BitField< Base, Index, BitPosition, Length >::set().

void io::driver::pstwo::PS2Device::disableSecondPS2Interrupt (uint8_t & *controllerByte*) [inline], [private] sets in the

Parameters

controllerByte,that	the second PS/2 Interrupt is disabled.

References tool::BitField< Base, Index, BitPosition, Length >::set().

void io::driver::pstwo::PS2Device::enableFirstPS2PortTranslation (uint8_t & *controllerByte*) `[inline]`, `[private]` sets in the

Parameters

controllerByte,that	in the first PS/2 Port the scancode translation is enabled.

References tool::BitField< Base, Index, BitPosition, Length >::set().

bool io::driver::pstwo::PS2Device::hasDualChannel (uint8_t *controllerByte*) `[inline]`, `[private]`

Parameters

controllerByte	

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the KBC Controller has a second PS/2 port

References tool::BitField< Base, Index, BitPosition, Length >::get().

uint8_t io::driver::pstwo::PS2Device::getControllerConfigurationByte ( ) `[inline]`, `[private]` @

@plus@

@plus -@

@

@

@skip

Returns

the controller configuration byte

References sendKbdCommandWithResponse().

void io::driver::pstwo::PS2Device::sendControllerConfigurationByte (uint8_t *controllerByte*) [inline], [private] sends the

Parameters

controllerByte	to the KBC Controller

References sendKbdCommand().

bool io::driver::pstwo::PS2Device::selfTestPassed ( ) [inline], [private]

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the PS2 controller passes the self test

References sendKbdCommandWithResponse().

bool io::driver::pstwo::PS2Device::port1TestPassed ( ) [inline], [private]

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the first PS/2 port passes the self test

References sendKbdCommandWithResponse().

bool io::driver::pstwo::PS2Device::port2TestPassed ( ) [inline], [private]

@

@plus@

@plus -@

@

@

@skip

Returns

true, if the second PS/2 port passes the self test

References sendKbdCommandWithResponse().

void io::driver::pstwo::PS2Device::enableFirstPS2Interrupt (uint8_t & *controllerByte*) [inline], [private] sets in the

Parameters

controller-Byte2,that	the first PS/2 Interrupt is enabled

References tool::BitField< Base, Index, BitPosition, Length >::set().

void io::driver::pstwo::PS2Device::enableSecondPS2Interrupt (uint8-_t & *controllerByte*) [inline], [private] sets in the

Parameters

controller-Byte2,that	the second PS/2 Interrupt is enabled

References tool::BitField< Base, Index, BitPosition, Length >::set().

The documentation for this class was generated from the following file:

- PS2Device.h

A.4.242 io::driver::pstwo::PS2Driver Class Reference

Driver (p. 455) for the PS/2 **Device** (p. 436).

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **PS2Device** ∗ **ps2device**

Additional Inherited Members

Detailed Description

Driver (p. 455) for the PS/2 **Device** (p. 436).

The documentation for this class was generated from the following file:

- PS2Driver.h

A.4.243 io::driver::pstwokeyboard::PS2KeyboardDevice Class Reference

Device (p. 436) for the PS/2 Keyboard.

Inherits **io::driver::Device**.

Public Member Functions

- **PS2KeyboardDevice** (**pstwo::PS2Device** ∗paramPs2Device)
- **tool::collection::Queue**
  < Scancode > & **getScancodeBuffer** ()
- void **setRepetitionRate** (uint8_t command)

sets the repetition rate from the

- void **setLed** (bool ScrollLockOn, bool NumLockOn, bool CapsLockOn)

  sets the LED of the keyboard

- void **activate** ()

  activate the keyboard

Private Member Functions

- void **sendKbdCommand** (uint8_t command)

  send an keyboard

- int **readScancode** ()

Private Attributes

- **tool::collection::Queue**< Scancode > **scancodeBuffer**

  Buffer the incoming scancodes.

- **pstwo::PS2Device** ∗ **ps2Device**

  Access to the PS/2 ***Device*** *(*p. *436).*

Friends

- class **PS2KeyboardHandler**

Detailed Description

Device (p. 436) for the PS/2 Keyboard.

Member Function Documentation

void io::driver::pstwokeyboard::PS2KeyboardDevice::sendKbdCommand (uint8_t *command*) [private] send an keyboard

Parameters

command	to the ps/2 keyboard

int io::driver::pstwokeyboard::PS2KeyboardDevice::readScancode ( ) [inline], [private] @

@plus@

@plus -@

@

@

@skip
Returns

the read scancode

References ps2Device.

tool::collection::Queue<Scancode>& io::driver::pstwokeyboard::PS2KeyboardDevice::getScancodeBuffer ( ) [inline] @

@plus@

@plus -@

@

@

@skip
Returns

the scancode buffer

References scancodeBuffer.

void io::driver::pstwokeyboard::PS2KeyboardDevice::setRepetitionRate (uint8_t *command*) sets the repetition rate from the

Parameters

command	

void io::driver::pstwokeyboard::PS2KeyboardDevice::setLed (bool *ScrollLockOn,* bool *NumLockOn,* bool *CapsLockOn*) sets the LED of the keyboard

Parameters

ScrollLockOn	
NumLockOn	
CapsLockOn	

The documentation for this class was generated from the following file:

- PS2KeyboardDevice.h

A.4.244 io::driver::pstwokeyboard::PS2KeyboardDriver Class Reference

Driver (p. 455) for the PS/2 Keyboard **Device** (p. 436).

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **PS2KeyboardDevice** ∗ **ps2keyboard**

Additional Inherited Members

Detailed Description

Driver (p. 455) for the PS/2 Keyboard **Device** (p. 436).

The documentation for this class was generated from the following file:

- PS2KeyboardDriver.h

A.4.245 io::driver::pstwokeyboard::PS2KeyboardHandler Class Reference

The **PS2KeyboardHandler** (p. 761) handles the interrupts which are fired when a key on the PS/2 keyboard is pressed or released.

Inherits **io::driver::interrupt::IRQHandler**.

Public Member Functions

- **PS2KeyboardHandler** (**PS2KeyboardDevice** ∗parKeyboardDevice)
- virtual bool **handle** ()

  Handles the interrupt.

Private Attributes

- **PS2KeyboardDevice** ∗ **keyboard**

Detailed Description

The **PS2KeyboardHandler** (p. 761) handles the interrupts which are fired when a key on the PS/2 keyboard is pressed or released.

Member Function Documentation

virtual bool io::driver::pstwokeyboard::PS2KeyboardHandler::handle ( ) [virtual] Handles the interrupt.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the IRQHandler handled the IRQ, else false. In the latter case, the PICInterruptHandler will call the next IRQHandler in the chain (if available).

Implements **io::driver::interrupt::IRQHandler** (p. 579).

The documentation for this class was generated from the following file:

- PS2KeyboardHandler.h

A.4.246 tool::collection::Queue< T > Class Template Reference

Queue (p. 762) class which does not use any locking.

Data Structures

- struct **Node**

Public Member Functions

- **Queue** (**memory::allocator::Allocator** &allocator=memory::getAllocator())
- bool **isEmpty** () const
- void **enqueue** (T data)

  append the data to the end of the queue
- T **dequeue** ()

Private Attributes

- **memory::allocator::Allocator** & **allocator**
- **Node** ∗ **frontPtr**

- **Node ∗ backPtr**

Detailed Description

template<typename T>class tool::collection::Queue< T >

Queue (p. 762) class which does not use any locking.

You have to ensure synchronized access by using a spin lock.

Member Function Documentation

template<typename T > bool tool::collection::Queue< T >::isEmpty ( ) const [inline] @

@plus@

@plus -@

@

@

@skip
Returns

true, if the queue is empty

Referenced by tool::collection::Queue< T >::enqueue().

template<typename T> void tool::collection::Queue< T >::enqueue (T *data*) append the data to the end of the queue

Parameters

data	

References tool::collection::Queue< T >::isEmpty().

template<typename T > T tool::collection::Queue< T >::dequeue ( ) [inline] @

@plus@

@plus -@

@

@

@skip

Returns

the first data in the queue REQUIRED: this->**isEmpty()** (p. 763) == false

The documentation for this class was generated from the following file:

- Queue.h

A.4.247 api::io::console::ReadKeyRequest Class Reference

Request for reading one character from the console.

Inherits **ipc::Request**.

Public Member Functions

- **ReadKeyRequest** ()

  Default constructor.

- char **getReadKey** () const
- void **setReadKey** (char **readKey**)

  sets the character which was read from the console to the value

Static Public Attributes

- static const uint32_t **Id** = 4

  The identifier of this Request.

Private Attributes

- char **readKey**

  The character which was read from the console.

Additional Inherited Members

Detailed Description

Request for reading one character from the console.

Member Function Documentation

char api::io::console::ReadKeyRequest::getReadKey ( ) const [inline]

@

@plus@

@plus -@

@

@

@skip

Returns

Returns the character which was read from the console

References readKey.

void api::io::console::ReadKeyRequest::setReadKey (char *readKey*) [inline] sets the character which was read from the console to the value

Parameters

readKey	

References readKey.

The documentation for this class was generated from the following file:

- ReadKeyRequest.h

A.4.248 object::Ref< T > Class Template Reference

Represents a reference to a reference-counted object.

Public Member Functions

- **Ref** (T ∗**object**=NULL)

  Constructor.

- **Ref** (**Ref**< T > const &ref)

  Copy constructor.

- template<typename U >
  Ref (**Ref**< U > const &ref)

  Constructor.

- **Ref**< T > & **operator=** (**Ref**< T > const &ref)

  Copy assignment operator.

- template<typename U >
  Ref< T > & **operator=** (**Ref**< U > const &ref)

  Copy assignment operator.

- ∼**Ref** ()

  Destructor. Decrements the reference count of referenced object.

- T ∗ **operator->** () const

  Returns a pointer to the referenced object.

- T & **operator**∗ () const

  Returns a reference to the referenced object.

- **operator bool** () const

  Returns true if the pointer is valid.

- bool **operator!** () const

  Returns false if the pointer is valid.

Private Member Functions

- void **incRef** ()

  Increments the reference count of referenced object (if any).

- void **decRef** ()

  Decrements the reference count of referenced object (if any).

Private Attributes

- T * **object**

  The referenced object.

Friends

- template<typename U >
  class **Ref**

Detailed Description

template<typename T>class object::Ref< T >

Represents a reference to a reference-counted object.

Constructor & Destructor Documentation

template<typename T> object::Ref< T >::Ref (T * *object* = NULL) [inline] Constructor.

Increments the reference count of referenced object.

Parameters

object	The referenced object. May be NULL to create an empty reference.

template<typename T> object::Ref< T >::Ref (Ref< T > const & *ref*) [inline] Copy constructor.

Increments the reference count of referenced object.

Parameters

ref	The reference to copy.

template<typename T> template<typename U > object::Ref< T >::Ref (Ref< U > const & *ref*) [inline] Constructor.

Increments the reference count of referenced object.

Parameters

ref	The reference to copy.

Member Function Documentation

template<typename T> Ref<T>& object::Ref< T >::operator= (Ref< T > const & *ref*) [inline] Copy assignment operator.

Decrements the reference count of the object being currently referenced and increments the reference count of passed object.

Parameters

ref	The reference to assign.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

template<typename T> template<typename U > Ref<T>& object::Ref< T >::operator= (Ref< U > const & *ref*) [inline] Copy assignment operator.

Decrements the reference count of the object being currently referenced and increments the reference count of passed object.

Parameters

ref	The reference to assign.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

The documentation for this class was generated from the following file:

- **Ref.h**

A.4.249 object::RefCountedObject Class Reference

Represents a reference-counted object.

Inherited by **io::vfs::AbstractMount**, and **io::vfs::FileSystemNode**.

Protected Member Functions

- virtual ~**RefCountedObject** ()

  Destructor.

Private Member Functions

- void **incRef** ()

  Increments the reference count of this object.

- void **decRef** ()

  Decrements the reference count of this object.

Private Attributes

- uint32_t **refCount**

  Number of references.

Friends

- template<typename T >
  void **Ref** ()
- template<typename T >
  void **Ref** ()

Detailed Description

Represents a reference-counted object.

If the last reference is gone, the object will destroy itself.

Member Function Documentation

void object::RefCountedObject::decRef ( ) [inline], [private] Decrements the reference count of this object.

If it reaches zero, the object is deleted.

References refCount.

The documentation for this class was generated from the following file:

- **RefCountedObject.h**

A.4.250 object::test::RefTestCase Class Reference

Tests reference-counted objects.

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual char const * **getName** ()

Additional Inherited Members

Detailed Description

Tests reference-counted objects.

The documentation for this class was generated from the following file:

- RefTestCase.h

A.4.251 ipc::Registry Class Reference

Manages remotely accessible objects.

Public Member Functions

- void **set** (void ∗object)

  Sets the interface for the current Process.

- void **unset** ()

  Unsets the interface for the current Process.

- bool **get** (**Participant** receiver, **RemoteObject** &result)

  Looks up an interface.

Static Public Member Functions

- static **Registry** & **getInstance** ()

  Returns the singleton instance of this class.

Private Types

- typedef
  tool::collection::LinearMap
  < uint32_t, void ∗ > **Entries**

  Map from Process identifiers to interface objects.

Private Member Functions

- **Registry** ()

  Constructor.

Static Private Member Functions

- static void **init** ()

  *Initializes the root of the **Registry*** (p. 771).

Private Attributes

- **task::lock::Semaphore mutex**

  *Protects the data of the **Registry** (p. 771).*

- **Entries entries**

  Maps process identifiers to interface objects.

Static Private Attributes

- static **Registry theInstance**

  The singleton instance of this class.

Friends

- class **boot::BootManager**

Detailed Description

Manages remotely accessible objects.

Each process can register one object that constitutes the interface of the Process and that can be accessed by other Processes through an IPC proxy.

Member Function Documentation

void ipc::Registry::set (void ∗ *object*) [inline] Sets the interface for the current Process.

Parameters

process	The process.
object	A pointer to the interface object.

References tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >-::add(), entries, getInstance(), and mutex.

bool ipc::Registry::get (Participant *receiver,* RemoteObject & *result*) [inline] Looks up an interface.

Parameters

receiver	The **Participant** (p. 698) the interface of which should be retrieved.
result	If successful, set to a valid **RemoteObject** (p. 773) encapsulating the **Participant** (p. 698)'s interface.

@

@plus@

@plus -@

@

@

@skip

Returns

True if a **RemoteObject** (p. 773) was found for passed **Participant** (p. 698), else false.

References entries, tool::collection::LinearMap< Key, Value, KeyComp, ValueComp >::get(), and mutex.

The documentation for this class was generated from the following file:

- **Registry.h**

A.4.252 ipc::RemoteObject Class Reference

Public Member Functions

- **RemoteObject** ()

  Default constructor.

- **RemoteObject** (**Participant targetProcess**, void ∗**handle**)

  Constructor.

- **Participant getTargetProcess** () const

  Returns the process this object lives in.

- void ∗ **getHandle** () const

  Returns the handle to the target object.

- bool **operator==** (**RemoteObject** const &other) const

*Compares two **RemoteObject** (p. 773) objects.*

Private Attributes

- **Participant targetProcess**

  The process this object lives in.

- void * **handle**

  A handle to the target object.

Constructor & Destructor Documentation

ipc::RemoteObject::RemoteObject (Participant *targetProcess,* void * *handle*) `[inline]` Constructor.

Parameters

target-Process	The target participant.
handle	A handle to the target object handling requests.

Member Function Documentation

bool ipc::RemoteObject::operator== (RemoteObject const & *other*) const `[inline]` Compares two **RemoteObject** (p. 773) objects.

Parameters

other	The other **RemoteObject** (p. 773) to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if both RemoteObjects have the same target process identifier and the same handle, else false.

References ipc::Participant::getProcessId(), handle, and targetProcess.

The documentation for this class was generated from the following file:

- **RemoteObject.h**

A.4.253 ipc::Request Class Reference

Represents a request.

Inherited by **api::io::console::ClearScreenRequest**, **api::io::console::OutputStringRequest**, **api::io::console::ReadKeyRequest**, **api::io::time::CurrentTimeRequest**, **api::kernel::GetVersionRequest**, **api::memory::lmm::AllocBlockRequest**, **api::memory::lmm::FreeBlockRequest**, **api::task::LoadProgramRequest**, **api::task::lock::CreateSemaphoreRequest**, **api::task::lock::ModifySemaphoreRequest**, **api::task::ProcessIdRequest**, **api::task::scheduler::ExitThreadRequest**, and **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- uint32_t **getId** () const

  Returns the request identifier.

- void **addMapRequest** (**MapRequest** const &mapRequest)

  *Adds an additional **MapRequest** (p. 647).*

- uint32_t **getNumberOfMapRequests** () const

  Returns the number of additional map requests.

- **MapRequest** & **getMapRequest** (uint32_t index)

  *Returns the n-th additional **MapRequest** (p. 647).*

Protected Member Functions

- **Request** (uint32_t **id**)

  Constructor.

Private Attributes

- uint32_t const **id**

The request identifier.

- uint32_t **numMapRequests**

  The number of additional MapRequests.

- **MapRequest mapRequests** [**NumMaxMapRequests**]

  Additional MapRequests.

Static Private Attributes

- static const uint32_t **NumMaxMapRequests** = 16

  The maximum number of additional MapRequests.

Detailed Description

Represents a request.

This is an abstract class. Subclasses represent concrete requests.

Constructor & Destructor Documentation

ipc::Request::Request (uint32_t *id*) [inline], [protected] Constructor.

Parameters

id	The request identifier.

Member Function Documentation

void ipc::Request::addMapRequest (MapRequest const & *mapRequest*) [inline] Adds an additional **MapRequest** (p. 647).

This can be used for sharing more memory regions between source and destination process, e.g. if data is to be exchanged via external buffers.

Parameters

mapRequest	The **MapRequest** (p. 647).

References mapRequests, and numMapRequests.

Referenced by ipc::Attribute< Array< T > >::map(), and ipc::Attribute< Array< T const > >::map().

MapRequest& ipc::Request::getMapRequest (uint32_t *index*) [inline]

Returns the n-th additional **MapRequest** (p. 647).

Parameters

index	The index of the **MapRequest** (p. 647) to retrieve.

@

@plus@

@plus -@

@

@

@skip

Returns

The **MapRequest** (p. 647).

References mapRequests.

Field Documentation

uint32_t const ipc::Request::id [private] The request identifier.

Each different request needs a different identifier.

Referenced by getId().

The documentation for this class was generated from the following file:

- **Request.h**

A.4.254 task::Process::Request Class Reference

Encapsulates an IPC request and a Semaphore.

Public Member Functions

- **Request** (**ipc::Request** ∗**request**)

Constructor.

- **ipc::Request** ∗ **getRequest** () const

  Returns the encapsulated IPC request.

- **task::lock::Semaphore** ∗ **getSemaphore** () const

  Returns the associated Semaphore.

Private Attributes

- **ipc::Request** ∗ **request**

  The encapsulated IPC request.

- **task::lock::Semaphore** ∗ **semaphore**

  The associated Semaphore.

Detailed Description

Encapsulates an IPC request and a Semaphore.

Constructor & Destructor Documentation

task::Process::Request::Request (ipc::Request ∗ ***request*** **)** [inline]
Constructor.

Creates a Semaphore for this request.

Parameters

request	The encapsulated IPC request.

The documentation for this class was generated from the following file:

- Process.h

A.4.255 ipc::test::FantasticObjectProxy::Request< 0 > Class Template Reference

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.256 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_add > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target, P1Type_add p1, P2Type_-add p2)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**< RetType_add > **result**
- ::**ipc::Attribute**< P1Type_add > **p1**
- ::**ipc::Attribute**< P2Type_add > **p2**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_add >

- **Request** (p. **??**) encapsulating the call of a binary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.257 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_getAnswer > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**
  < RetType_getAnswer > **result**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_getAnswer >

- **Request** (p. **??**) encapsulating the call of a nullary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.258 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_increment > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target, P1Type_increment p1)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**
  < RetType_increment > **result**
- ::**ipc::Attribute**
  < P1Type_increment > **p1**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request<
FantasticObjectProxy::Id_increment >

- **Request** (p. **??**) encapsulating the call of a unary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.259 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementElements > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target, **P1Type_incrementElements** p1)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**
  < **P1Type_incrementElements** > **p1**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request<
FantasticObjectProxy::Id_incrementElements >

- **Request** (p. **??**) encapsulating the call of a unary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.260 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementInPlace > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target, P1Type_incrementInPlace p1)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**
  < P1Type_incrementInPlace > **p1**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementInPlace >

- **Request** (p. **??**) encapsulating the call of a unary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.261 ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_print > Class Template Reference

Inherits **ipc::test::FantasticObjectProxy::RequestBase**.

Public Member Functions

- **Request** (::**ipc::RemoteObject** const &target, **P1Type_print** p1)

Static Public Member Functions

- static void **initializeRequest** ()
- static void **finalizeRequest** ()

Data Fields

- ::**ipc::Attribute**< **P1Type_print** > **p1**

Additional Inherited Members

Detailed Description

template<>class ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_print >

- **Request** (p. **??**) encapsulating the call of a unary operation.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.262 ipc::test::FantasticObjectProxy::RequestBase Class Reference

Inherits **ipc::Request**.

Inherited by **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_add >**, **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_getAnswer >**, **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_increment >**, **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementElements >**, **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_incrementInPlace >**, and **ipc::test::FantasticObjectProxy::Request< FantasticObjectProxy::Id_print >**.

Public Member Functions

- **RequestBase** (uint32_t **id**,::**ipc::RemoteObject** target)
- ::**ipc::RemoteObject** const & **getTarget** () const
- void **handle** ()

Static Protected Member Functions

- static void **registerCommand** (uint32_t **id**, **CommandBase** *command)
- static void **unregisterCommand** (uint32_t **id**)

Private Attributes

- ::**ipc::RemoteObject target**

Additional Inherited Members

Detailed Description

- Base class of all generated requests.

Member Function Documentation

::ipc::RemoteObject const& ipc::test::FantasticObjectProxy::RequestBase::getTarget ( ) const [inline]

- Returns the target **RemoteObject** (p. 773).

void ipc::test::FantasticObjectProxy::RequestBase::handle ( ) [inline]

- Handles an incoming request by dispatching it to the correct method.

The documentation for this class was generated from the following file:

- FantasticObject.h

A.4.263 memory::lmm::LinearAddressSpaceManager::ReservationTable Struct Reference

Holds entries for address space management.

Public Member Functions

- **ReservationTable** (**ReservationTable** ∗**next**=NULL)

  Constructor.

Data Fields

- **AddressSpaceRange entries** [**EntriesPerTable**]

  The entries.

- **ReservationTable** ∗ **next**

  Points to the next ***ReservationTable*** *(*p. *785) if available.*

Detailed Description

Holds entries for address space management.

Part of a linked list.

Constructor & Destructor Documentation

memory::lmm::LinearAddressSpaceManager::ReservationTable::ReservationTable (**ReservationTable** ∗ ***next*** **=** `NULL` **)** Constructor.

Initializes all entries to INVALID.

Parameters

next	The next reservation table. May be NULL only for the first table.

The documentation for this struct was generated from the following file:

- LinearAddressSpaceManager.h

A.4.264 task::Thread::Resource Struct Reference

Associates acquired Semaphores with Threads waiting for them.

Public Types

- typedef
  tool::collection::LinkedList
  < **Thread** ∗ > **Waiters**

  Type of a list of Threads waiting for a Semaphore.

Public Member Functions

- **Resource** (**lock::Semaphore** &**semaphore**)

  Constructor.

- bool **operator==** (**Resource** const &other) const

  Returns true if passed object is equal to this one.

Data Fields

- **lock::Semaphore** & **semaphore**

  The Semaphore we have acquired.

- uint32_t **useCount**

  The number of times we acquired the Semaphore.

- **Waiters waiters**

  Another threads waiting for the semaphore.

Detailed Description

Associates acquired Semaphores with Threads waiting for them.

Constructor & Destructor Documentation

task::Thread::Resource::Resource (lock::Semaphore & *semaphore*)
Constructor.

Parameters

semaphore	The semaphore.

Member Function Documentation

bool task::Thread::Resource::operator== (Resource const & *other*) const `[inline]` Returns true if passed object is equal to this one.

Only the Semaphores are compared.

Parameters

other	The other object to compare with.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the other **Resource** (p. 786) object refers to the same Semaphore, else false.

References semaphore.

The documentation for this struct was generated from the following file:

- Thread.h

A.4.265 ipc::Result< T > Class Template Reference

Encapsulates the result of an IPC request.

Public Member Functions

- **Result** (bool **valid**, **Attribute**< T > const &**attr**)

  Constructor.

- bool **isValid** () const

  Returns true if the result is a valid one.

- **Attribute**< T >::ResultType **get** ()

  Returns the result of the IPC request.

Private Attributes

- bool **valid**

  True if the result stored is valid.

- **Attribute**< T > **attr**

  The result of the IPC request.

Detailed Description

template<**typename T**>**class ipc::Result**< **T** >

Encapsulates the result of an IPC request.

Constructor & Destructor Documentation

template<**typename T**> **ipc::Result**< **T** >**::Result (bool *valid,* Attribute**< **T** > **const & *attr*)** [inline] Constructor.

Parameters

valid	If true, the IPC request has been handled. If false, an error occurred while handling the request, e.g. an out-of-memory situation.
attr	The attribute storing the result of the IPC request. Only meaningful if 'valid' is true.

Member Function Documentation

template<**typename T**> **Attribute**<**T**>**::ResultType ipc::Result**< **T** >**::get ( )** [inline] Returns the result of the IPC request.

Only meaningful if **isValid()** (p. 788)==true.

References ipc::Result< T >::attr.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.266 ipc::Result< Array< T > > Class Template Reference

Encapsulates an **Array** (p. 292) result of an IPC request.

Detailed Description

template<typename T>class ipc::Result< Array< T > >

Encapsulates an **Array** (p. 292) result of an IPC request.

As this is currently not supported, the specialization is intentionally left empty.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.267 ipc::Result< void > Class Template Reference

Encapsulates a void result of an IPC request.

Public Member Functions

- **Result** (bool **valid**)

  Constructor.

- bool **isValid** () const

  Returns true if the IPC request has been handled.

Private Attributes

- bool **valid**

  True if the IPC request has been handled.

Detailed Description

template<>class ipc::Result< void >

Encapsulates a void result of an IPC request.

Constructor & Destructor Documentation

ipc::Result< void >::Result (bool *valid*) [inline] Constructor.

Parameters

valid	If true, the IPC request has been handled. If false, an error occurred while handling the request, e.g. an out-of-memory situation.

The documentation for this class was generated from the following file:

- **Proxy.h**

A.4.268 RootDirectory Class Reference

Represents the root directory.

Data Structures

- struct **Entry**

  Describes a directory entry.

Public Member Functions

- **RootDirectory** (**Disk** const ∗**disk**)

  *Creates a **RootDirectory** (p. 791) object.*
- **Entry** const ∗ **getNext** ()

  Returns the next directory entry or null if no further entries exist.
- **Entry** const ∗ **getByName** (char const ∗name)

  Returns the directory entry with passed name or null if no such entry exists.

Private Member Functions

- void **loadNext** ()

  Advances to the next directory entry.

Private Attributes

- **Disk** const ∗ **disk**

  The underlying disk.
- **BPB** const ∗ **bpb**

*The underlying **BPB** (p. 375).*

- unsigned const **numEntries**

  Maximum number of directory entries.

- unsigned const **numEntriesPerSector**

  Maximum number of directory entries per sector.

- void *const **buffer**

  Sector buffer.

- unsigned **currentEntry**

  The index of the current entry.

- **DiskAddress currentSector**

  The address of the current sector loaded into the buffer.

- **Entry * current**

  *The current directory entry returned by the last **getNext()** (p. 792) call.*

Detailed Description

Represents the root directory.

Constructor & Destructor Documentation

RootDirectory::RootDirectory (Disk const * *disk*) Creates a **RootDirectory** (p. 791) object.

Parameters

disk	The disk whose root directory is to be read.

Member Function Documentation

Entry const* RootDirectory::getNext ( ) `[inline]` Returns the next directory entry or null if no further entries exist.

@

@plus@

@plus -@

@

@

@skip
Returns

The next entry or a null pointer otherwise.

References current, and loadNext().

Entry const∗ RootDirectory::getByName (char const ∗ *name*) Returns the directory entry with passed name or null if no such entry exists.

@

@plus@

@plus -@

@

@

@skip
Returns

The desired entry or a null pointer otherwise.

The documentation for this class was generated from the following file:

- **dir.h**

A.4.269 io::driver::rtc::RTCDevice Class Reference

Device (p. 436) for the Real Time Clock (RTC) The RTC is a part of the CMOS RAM.

Inherits **io::driver::Device**.

Public Member Functions

- uint32_t **getTime** ()

  Reads the current date and time from RTC and returns it's timestamp in form of seconds since 1970-01-01 00:00:00.

- void **addHandler** ()

  *adds **RTCHandler** (p. 797) to the Slave PIC on PIN 0*

- void **setIRQ** (bool val)

  activates or deactivates the IRQ (bit 6 of register B)

- void **setBase** (uint8_t base)

  Sets the base for the interrupt.

- void **setRate** (uint8_t rate)

  Sets the rate for the interrupt must be between 0 and 15 otherwise 6 will be set! 0 Deactivates the IRQ.

- void **afterIRQ** ()

  The C register has to be read out, otherwise no further interrupts.

Private Member Functions

- void **writeCtrl** (uint8_t ctrl)

  We'll make all of the read and write operation private.

- void **writeData** (uint8_t data)

  writes <data> into the data port of the RTC

- uint8_t **readCtrl** ()
- uint8_t **readData** ()
- void **write** (uint8_t key, uint8_t val)

  writeCtrl(key); writeData(val);

- uint8_t **read** (uint8_t cmd)

  *writeCtrl(cmd); return **readData()*** (p. 796)*;*

Private Attributes

- **IOPort ctrlIO**

  Access to the RTC Control port.

- **IOPort dataIO**

  Access to the RTC Data port.

Detailed Description

Device (p. 436) for the Real Time Clock (RTC) The RTC is a part of the CMOS RAM.

The bytes 0x00-0x0C on port 0x70 are for RTC

Member Function Documentation

void io::driver::rtc::RTCDevice::writeCtrl (uint8_t *ctrl*) [private] We'll make all of the read and write operation private.

I don't see any use for them from the public. These are used to get/set the clock and activating the IRQ (IRQ8) TODO: create a setDate operation writes <ctrl> into the control port of the RTC

Parameters

ctrl	byte to write into the control port

void io::driver::rtc::RTCDevice::writeData (uint8_t *data*) [private] writes <data> into the data port of the RTC

Parameters

data	byte to write to the data port

uint8_t io::driver::rtc::RTCDevice::readCtrl ( ) [private] @

@plus@

@plus -@

@

@

@skip
Returns

the content of the control port of the RTC allways 0xFF (port is read only)

uint8_t io::driver::rtc::RTCDevice::readData ( ) [private] @

@plus@

@plus -@

@

@

@skip
Returns

the content of the data port of the RTC

void io::driver::rtc::RTCDevice::setIRQ (bool *val*) activates or deactivates the IRQ (bit 6 of register B)

Parameters

val	set or unset the IRQ

void io::driver::rtc::RTCDevice::setBase (uint8_t *base*) Sets the base for the interrupt.

Parameters

base	base to set

void io::driver::rtc::RTCDevice::setRate (uint8_t *rate*) Sets the rate for the interrupt must be between 0 and 15 otherwise 6 will be set! 0 Deactivates the IRQ.

Parameters

rate	rate to set

void io::driver::rtc::RTCDevice::afterIRQ ( ) The C register has to be read out, otherwise no further interrupts.

This is a helper operation for the handler. It read out the Register and does nothing further.

The documentation for this class was generated from the following file:

- RTCDevice.h

A.4.270 io::driver::rtc::RTCDriver Class Reference

Driver (p. 455) for the Real Time Clock (RTC)

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- **RTCDevice** ∗ **rtc**

Additional Inherited Members

Detailed Description

Driver (p. 455) for the Real Time Clock (RTC)

The documentation for this class was generated from the following file:

- RTCDriver.h

A.4.271 io::driver::rtc::RTCHandler Class Reference

The RTC Interrupt handler.

Inherits **io::driver::interrupt::IRQHandler**.

Public Member Functions

- **RTCHandler** (**RTCDevice** ∗**rtc**)
- virtual bool **handle** ()

  Handles the interrupt.

Private Attributes

- **RTCDevice * rtc**

  Buffer for the rtc device.

- **io::time::Time * time**

  Buffer for the Time instance.

- **RTCTasklet * tasklet**

  The queue work is done by a tasklet.

Detailed Description

The RTC Interrupt handler.

Member Function Documentation

virtual bool io::driver::rtc::RTCHandler::handle ( ) [virtual] Handles the interrupt.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the IRQHandler handled the IRQ, else false. In the latter case, the PICInterruptHandler will call the next IRQHandler in the chain (if available).

Implements **io::driver::interrupt::IRQHandler** (p. 579).

Field Documentation

RTCDevice* io::driver::rtc::RTCHandler::rtc [private] Buffer for the rtc device.

Otherwise it would take up to much time to get it from the device manager for each tick

RTCTasklet* io::driver::rtc::RTCHandler::tasklet [private] The queue work is done by a tasklet.

We have to inform it that it has work.

The documentation for this class was generated from the following file:

- RTCHandler.h

A.4.272 io::driver::rtc::RTCTasklet Class Reference

The Tasklet for the RTC IRQ handler.

Inherits **task::tasklet::Tasklet**.

Public Member Functions

- virtual task::tasklet::TaskletState **getState** () const

  Gets the current state of the tasklet.

- virtual void **work** ()

  Sets all wait semaphores up which should be reactivated.

- void **setTaskletHasWork** ()

  Sets the state to HASWORK.

Private Attributes

- task::tasklet::TaskletState **state**

  State variable since we don't want to be activated each run.

Detailed Description

The Tasklet for the RTC IRQ handler.

Member Function Documentation

void io::driver::rtc::RTCTasklet::setTaskletHasWork ( ) Sets the state to HASWORK.

Should be called by the handler.

The documentation for this class was generated from the following file:

- RTCTasklet.h

A.4.273 runtime::test::RTTITestCase Class Reference

Tests RTTI (run-time type identification) in kernel mode.

Inherits **test::TestCase**.

Public Member Functions

- virtual char const * **getName** ()

Protected Member Functions

- virtual void **run** ()

Private Member Functions

- void **testPrimitiveTypes** ()

  Tests RTTI on objects of primitive types.

- void **testClassTypes** ()

  Tests RTTI on objects of class types.

- void **testDynamicCast** ()

  Tests operator dynamic_cast<>.

- void **testNULL** ()

  Tests RTTI on NULL.

Detailed Description

Tests RTTI (run-time type identification) in kernel mode.

The documentation for this class was generated from the following file:

- RTTITestCase.h

A.4.274 task::scheduler::Scheduler Class Reference

The **Scheduler** (p. 801) keeps all Processes of a computer system.

Public Member Functions

- ∼**Scheduler** ()

  Destructor.

- **Thread** ∗ **getCurrentThread** ()

  *Returns a pointer to the **Thread** (p. 865) being currently executed.*

- void **yield** ()

  Requests rescheduling, i.e.

Static Public Member Functions

- static **Scheduler** & **getInstance** ()

  Returns.

Private Types

- typedef
  tool::collection::LinkedList
  < **Thread** ∗ > **Threads**

  Type of a list of Threads.

Private Member Functions

- **Scheduler** ()

  Constructor.

- void **cleanupInitialTask** ()

  Deletes resources associated with the initial boot task.

- void **reschedule** (ThreadState newState, bool putAtEnd)

  Requests rescheduling, i.e.

- void **doReschedule** (ThreadState newState, bool putAtEnd)

Requests rescheduling, i.e.

- void **dispatch** (**Thread** ∗**next**)

  Changes the current thread and performs a task switch.

- void **attachThread** (**Thread** ∗thread)

  *Attaches a **Thread** (p. 865) to the **Scheduler** (p. 801).*

- void **detachTerminatedThreads** ()

  Detaches all terminated threads.

- void **createIdleThread** ()

  Creates the idle thread.

- void **unblockWokenUpThreads** ()

  Unblocks all threads after a semaphore has been released.

- **ThreadQueue** & **getQueue** (task::ThreadPriority prio)

  Returns the queue for a given thread priority.

- **ThreadQueue** & **getActiveQueue** ()

  Returns the queue containing the highest prioritized thread to run.

- void **addThread** (**Thread** ∗thread)

  Adds a thread to its queue.

- void **yield** (ThreadState newState)

  Requests rescheduling, i.e.

- void **block** (cpu::level::IRQLevel origIRQLevel)

  Requests rescheduling because the calling thread is blocking on a Semaphore.

- void **handlePrioInheritanceDown** (**Thread** ∗thread)

  Handles priority inheritance when a thread blocks on a Semaphore.

- void **handlePrioInheritanceUp** (**lock::Semaphore** ∗semaphore)

  Handles priority inheritance when a Semaphore is released, allowing a blocked thread to be unblocked.

- void **setTimeSliceExceeded** ()

  *Called by the **TimerInterruptHandler** (p. 884) to indicate that the current time slice has been exceeded.*

- bool **hasTimeSliceExceeded** ()

  *Called by the **SchedulerHandler** (p. 809) to determine whether the current time slice has been exceeded.*

Static Private Member Functions

- static bool **init** ()

  *Creates the **Scheduler** (p. 801) object.*

- static void **threadEntryPoint** (**Thread** ∗thread)

  Main entry point for new threads.

- static void **threadExit** ()

  Exit point for all threads.

Private Attributes

- **task::lock::SpinLock mutex**

  *Spinlock disabling thread preemption while executing **Scheduler** (p. 801) code.*

- **ThreadQueue** ∗ **runnableThreads** [task::PRIO_NUM]

  List of runnable threads, by current priority.

- task::ThreadPriority **highestPrio**

  Highest priority level currently in use.

- **Thread** ∗ **currentThread**

  The thread currently running.

- **Threads blockedThreads**

  List of blocked threads.

- **Threads terminatedThreads**

  List of terminated threads.

- **io::driver::timer::PIT8254Counter** ∗ **counter**

  The PIT counter producing timer interrupts used for thread preemption.

- **io::driver::interrupt::IRQHandler** ∗ **timerIRQHandler**

  The timer IRQ handler.

- **task::TaskStateSegment** ∗ **initialTss**

  The initial TSS.

- **memory::Selector initialTssSelector**

  The selector for the initial TSS.

- **memory::CodeOrDataSegmentDescriptor** & **tlsDescriptor**

The TLS descriptor.

- **task::lock::SpinLock** ∗ **timeSliceExceededMutex**

  Spinlock protecting the timeSliceUp flag.

- bool **timeSliceExceeded**

  True if the current time slice has been exceeded.

- **cpu::level::TransitionHandler** ∗ **schedulerTransitionHandler**

  *The **Scheduler** (p. 801)'s transition handler.*

- bool **initFlag**

  Set if initialization is successful.

Static Private Attributes

- static **Scheduler** ∗ **instance**

  *Contains the instance of the **Scheduler** (p. 801).*

Friends

- class **SchedulerHandler**
- class **TimerInterruptHandler**
- class **boot::BootManager**
- class **task::Thread**
- class **task::lock::SpinLock**
- class **api::task::scheduler::ExitThreadCommand**

Detailed Description

The **Scheduler** (p. 801) keeps all Processes of a computer system.

He is responsible for order them and their Threads and, if necessary, bring them to execution. For executing them the **Scheduler** (p. 801) uses a class called "Dispatcher".

Threads are ordered by Processes and Processes are ordered by their creation time, while this time is a relative time, not a exact one. So no time of **Process** (p. 737) creation can be provided.

Member Function Documentation

static bool task::scheduler::Scheduler::init ( ) [inline], [static], [private] Creates the **Scheduler** (p. 801) object.

@

@plus@

@plus -@

@

@

@skip
Returns

True if initialization has succeeded, else false.

References initFlag, instance, and Scheduler().

void task::scheduler::Scheduler::cleanupInitialTask ( ) [private] Deletes resources associated with the initial boot task.

Called by BootManager::cleanupAfterBoot().

void task::scheduler::Scheduler::reschedule (ThreadState *newState*, bool *putAtEnd*) [inline], [private] Requests rescheduling, i.e.

choosing the next thread to be run (which need not be different from the current one). Requires IRQL <= DISPATCH.

Parameters

newState	The new state of the current thread after releasing the time slice.
putAtEnd	If true, the currently running thread is put at the end of its queue, otherwise it is put at the beginning of its queue.

References doReschedule(), and mutex.

void task::scheduler::Scheduler::doReschedule (ThreadState *newState*, bool *putAtEnd*) [private] Requests rescheduling, i.e.

choosing the next thread to be run (which need not be different from the current one). Requires the spinlock to be already held by the current thread.

Parameters

newState	The new state of the current thread after releasing the time slice.
putAtEnd	If true, the currently running thread is put at the end of its queue, otherwise it is put at the beginning of its queue.

Referenced by reschedule().

void task::scheduler::Scheduler::dispatch (Thread * *next*) [private]
Changes the current thread and performs a task switch.

Parameters

nextThread	The thread to jump to.

static void task::scheduler::Scheduler::threadEntryPoint (Thread * *thread*) [static], [private] Main entry point for new threads.

Releases the **Scheduler** (p. 801)'s spinlock (thereby resetting the IRQL to PASSIVE) and calls **Thread** (p. 865):run(). After termination the **Thread** (p. 865)'s state is set to TERMINATED.

Parameters

thread	The thread to be run.

void task::scheduler::Scheduler::attachThread (Thread * *thread*) [private] Attaches a **Thread** (p. 865) to the **Scheduler** (p. 801).

This thread must be in state TERMINATED.

Parameters

thread	The **Thread** (p. 865) to attach.

ThreadQueue& task::scheduler::Scheduler::getQueue (task::ThreadPriority *prio*) [inline], [private] Returns the queue for a given thread priority.

Parameters

prio	The priority.

@

@plus@

@plus -@

@

@

@skip

Returns

The associated queue.

References runnableThreads.

Referenced by getActiveQueue().

void task::scheduler::Scheduler::addThread (Thread * *thread*) [private]

Adds a thread to its queue.

Parameters

thread	The thread to add.

void task::scheduler::Scheduler::yield (ThreadState *newState*) [private]

Requests rescheduling, i.e.

choosing the next thread to be run (which need not be different from the current one). Requires IRQL == PASSIVE.

Parameters

newState	The new state of the current thread after releasing the time slice.

void task::scheduler::Scheduler::block (cpu::level::IRQLevel *origIRQLevel*) [private] Requests rescheduling because the calling thread is blocking on a Semaphore.

Parameters

origIRQ-Level	The original IRQ level before acquiring the Semaphore's spin lock. Required to be IRQL_PASSIVE. Otherwise a fatal error occurs. (It is not allowed to switch to another thread while at IRQL_DISPATCH or higher.)

void task::scheduler::Scheduler::handlePrioInheritanceDown (Thread * *thread*) [private] Handles priority inheritance when a thread blocks on a Semaphore.

The priorities of all the threads that have acquired the Semaphore are raised to at least the priority of the blocking thread (recursively).

Parameters

thread	The thread blocking on a Semaphore.

void task::scheduler::Scheduler::handlePrioInheritanceUp (lock::Semaphore * *semaphore*) [private] Handles priority inheritance when a Semaphore is released, allowing a blocked thread to be unblocked.

The priorities of all the threads that have acquired the Semaphore are lowered to the priority they would have if the unblocked thread would never have been blocked.

Parameters

semaphore	The Semaphore that has been released.

bool task::scheduler::Scheduler::hasTimeSliceExceeded ( ) [inline], [private] Called by the **SchedulerHandler** (p. 809) to determine whether the current time slice has been exceeded.

Resets the timeSliceFlag before returning.

References timeSliceExceeded, and timeSliceExceededMutex.

static Scheduler& task::scheduler::Scheduler::getInstance ( ) [inline], [static] Returns.

@

@plus@

@plus -@

@

@

@skip
Returns

the instance of the only **Scheduler** (p. 801), as **Scheduler** (p. 801) is a singleton.

References instance.

Referenced by task::ProcessManager::getCurrentProcess().

void task::scheduler::Scheduler::yield ( ) `[inline]` Requests rescheduling, i.e.

choosing the next thread to be run (which need not be different from the current one). This allows threads to implement cooperative multitasking, e.g. for coroutines. Requires IRQL == PASSIVE.

The documentation for this class was generated from the following file:

- Scheduler.h

A.4.275 task::scheduler::SchedulerHandler Class Reference

Inherits **cpu::level::TransitionHandler**.

Public Member Functions

- **SchedulerHandler** (cpu::level::IRQLevel timerIRQL)
- virtual cpu::level::IRQLevel **lowestLevel** () const

  Returns the lowest IRQ level of a transition (inclusive).
- virtual cpu::level::IRQLevel **highestLevel** () const

  Returns the highest IRQ level of a transition (exclusive).
- virtual void **raise** (cpu::level::IRQLevel, cpu::level::IRQLevel)

Is called by the LevelManager whenever the IRQ level is raised.

- virtual void **lower** (cpu::level::IRQLevel fromLevel, cpu::level::IRQLevel toLevel)

  Is called by the LevelManager whenever the IRQ level is lowered.

Private Attributes

- cpu::level::IRQLevel **timerIRQL**

Member Function Documentation

virtual cpu::level::IRQLevel task::scheduler::SchedulerHandler::lowestLevel ( ) const `[inline]`**,** `[virtual]` Returns the lowest IRQ level of a transition (inclusive).

The handler is only invoked if the new (while raising) or old (while lowering) IRQ level is greater than or equal to this level. @

@plus@

@plus -@

@

@

@skip
Returns

The highest IRQ level.

Implements **cpu::level::TransitionHandler** (p. 886).

virtual cpu::level::IRQLevel task::scheduler::SchedulerHandler::highestLevel ( ) const `[inline]`**,** `[virtual]` Returns the highest IRQ level of a transition (exclusive).

The handler is only invoked if the old (while raising) or new (while lowering) IRQ level is less than this level. @

@plus@

@plus -@

@

@

@skip
Returns

The highest IRQ level.

Implements **cpu::level::TransitionHandler** (p. 887).

virtual void task::scheduler::SchedulerHandler::raise (cpu::level::IRQLevel *fromLevel,* cpu::level::IRQLevel *toLevel*) [inline], [virtual]
Is called by the LevelManager whenever the IRQ level is raised.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implements **cpu::level::TransitionHandler** (p. 887).

virtual void task::scheduler::SchedulerHandler::lower (cpu::level::IRQLevel *fromLevel,* cpu::level::IRQLevel *toLevel*) [virtual] Is called by the LevelManager whenever the IRQ level is lowered.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implements **cpu::level::TransitionHandler** (p. 888).

The documentation for this class was generated from the following file:

- SchedulerHandler.h

A.4.276 memory::Selector Class Reference

Encapsulates a **Selector** (p. 811).

Public Member Functions

- **Selector** ()

  Constructor.

- **Selector** (uint16_t **value**)

Constructor.

- uint16_t **getIndex** () const

  Returns the selector index.

- **Selector** & **setIndex** (uint16_t index)

  Sets the selector index.

- uint16_t **getRPL** ()

  *Returns the RPL (Requested Privilege Level) of this **Selector** (p. 811).*

- **Selector** & **setRPL** (uint8_t rpl)

  Sets the RPL (Requested Privilege Level).

- bool **isLDT** () const

  *If this **Selector** (p. 811) points into the LDT, this operation returns true.*

- **Selector** & **setLDT** (bool ldt)

  Sets the LDT flag.

- uint16_t **getValue** () const

  Returns the selector value.

Private Types

- typedef **tool::BitField**
  < uint16_t, 0, 0, 2 > **RPLField**

  Describes the RPL field.

- typedef **tool::BitField**
  < uint16_t, 0, 2, 1 > **LDTField**

  Describes the LDT flag.

- typedef **tool::BitField**
  < uint16_t, 0, 3, 13 > **IndexField**

  Describes the index field.

Private Attributes

- uint16_t **value**

  The selector value.

Detailed Description

Encapsulates a **Selector** (p. 811).

A **Selector** (p. 811) points into some **Descriptor** (p. 432) table (GDT or LDT) and references a code, data, or system segment.

Constructor & Destructor Documentation

memory::Selector::Selector ( ) [inline] Constructor.

Creates a NULL GDT **Selector** (p. 811), RPL 0.

memory::Selector::Selector (uint16_t *value*) [inline], [explicit] Constructor.

Creates a **Selector** (p. 811) from a 16-bit value.

Parameters

value	The selector value.

Member Function Documentation

Selector& memory::Selector::setIndex (uint16_t *index*) [inline] Sets the selector index.

Parameters

index	The selector index.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

References tool::BitField< Base, Index, BitPosition, Length >::set(), and value.

Selector& memory::Selector::setRPL (uint8_t *rpl*) [inline] Sets the RPL (Requested Privilege Level).

Parameters

rpl	The RPL.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References tool::BitField< Base, Index, BitPosition, Length >::set(), and value.

bool memory::Selector::isLDT ( ) const [inline] If this **Selector** (p. 811) points into the LDT, this operation returns true.

Otherwise this **Selector** (p. 811) points into the GDT and this operation returns false in this case.

References tool::BitField< Base, Index, BitPosition, Length >::get(), and value.

Selector& memory::Selector::setLDT (bool *ldt*) [inline] Sets the LDT flag.

Parameters

ldt	If true, this **Selector** (p. 811) points into the LDT, else into the GDT.

@

@plus@

@plus -@

@

@

@skip
Returns

*this

References tool::BitField< Base, Index, BitPosition, Length >::set(), and value.

The documentation for this class was generated from the following file:

- Selector.h

A.4.277 task::lock::Semaphore Class Reference

Class allowing passive waiting for a resource.

Data Structures

- class **Lock**

  Locker class which acquires a semaphore in the constructor and releases it in the destructor.

Public Member Functions

- **Semaphore** (uint32_t **permits**, bool **symmetric**)

  Creates a new semaphore with the specified number of initial permits.

- ~**Semaphore** ()

  Destructor.

- void **down** ()

  *Acquires a permit from the **Semaphore*** (p. *816).*

- void **up** ()

  Release an acquired permit.

- uint32_t **getRemainingPermits** () const

  Returns the count of remaining permits.

Private Types

- typedef
  tool::collection::LinkedList
  < **Thread** * > **Lockers**

  Type of a list of threads.

Private Member Functions

- **Lockers getLockers** () const

  Returns the lockers.

- void **removeLocker** (**Thread** &thread)

  Removes a locker from the list.

Private Attributes

- **SpinLock mutex**

  *Spinlock protecting the **Semaphore** (p. 816) data.*

- uint32_t **permits**

  Internal counter of remaining permits for this semaphore.

- bool const **symmetric**

  *True if the **Semaphore** (p. 816) is symmetric, else false.*

- **Lockers lockers**

  *The threads that have successfully **down()** (p. 818)ed this **Semaphore** (p. 816).*

Friends

- class **scheduler::Scheduler**
- class **task::Thread**

Detailed Description

Class allowing passive waiting for a resource.

Use only when working at PASSIVE level. If you want to synchronize resources at higher IRQ levels use a **SpinLock** (p. 824).

Constructor & Destructor Documentation

task::lock::Semaphore::Semaphore (uint32_t *permits*, bool *symmetric*) Creates a new semaphore with the specified number of initial permits.

Parameters

permits	Number of initial permits.
symmetric	If true, the usage pattern for this **Semaphore** (p. 816) is symmetric. That means that every **Thread** (p. 865) is calling **down()** (p. 818) and later **up()** (p. 818), i.e. the **Semaphore** (p. 816) is used for protecting access to a set of resources. A typical scenario is a critical section or more generally a guard letting only at most "permits" threads into a method at the same time. Symmetric semaphores are handled specially by the Scheduler, they are especially analyzed for detecting priority inversion situations. If false, the thread that calls **down()** (p. 818) typically differs from the thread calling **up()** (p. 818). This second usage pattern is typically used when a semaphore is used to count available data in a thread-safe collection (e.g. a pipe). Asymmetric Semaphores are not handled specially by the Scheduler, they are not taken into consideration when trying to detect priority inversion and, consequently, they do not cause thread priorities to be changed, which make them especially good candidates for transferring data from a tasklet to a worker thread.

task::lock::Semaphore::~Semaphore ( ) Destructor.

Unregisters from all lockers.

Member Function Documentation

void task::lock::Semaphore::down ( ) Acquires a permit from the **Semaphore** (p. 816).

Requires IRQL == PASSIVE.

If no permit is available, the current thread will be set to state BLOCKED until a permit is available.

Referenced by task::lock::Semaphore::Lock::Lock(), io::driver::block::ata::commands::AtaCommand::wait(), and io::driver::block::ata::commands::AtaCommand::waitForCompletion().

void task::lock::Semaphore::up ( ) Release an acquired permit.

Requires IRQL <= **Semaphore** (p. 816)'s IRQ level.

Referenced by io::driver::block::ata::commands::AtaCommand::notifyClient(), io::driver::block::ata::commands::AtaCommand::notifyWorker(), and task::lock::-Semaphore::Lock::~Lock().

uint32_t task::lock::Semaphore::getRemainingPermits ( ) const [inline]

Returns the count of remaining permits.

(Not thread-safe, only used for testing purposes.)

References permits.

void task::lock::Semaphore::removeLocker (Thread & *thread*) [private]

Removes a locker from the list.

Called by **Thread::~Thread()** (p. 865).

Parameters

thread	The terminating thread.

The documentation for this class was generated from the following file:

- kernel/task/lock/Semaphore.h

A.4.278 lib::Semaphore Class Reference

Represents a **Semaphore** (p. 819) in user space.

Public Member Functions

- **Semaphore** ()

  *Constructor. Creates a new **Semaphore** (p. 819).*

- ~**Semaphore** ()

  *Destructor. Destroys the **Semaphore** (p. 819).*

- void **up** ()

  *Performs a **up()** (p. 819) on the **Semaphore** (p. 819).*

- void **down** ()

  *Performs a **down()** (p. 819) on the **Semaphore** (p. 819).*

Private Attributes

- uint32_t **semaphorePointer**

  *Identifies the real **Semaphore** (p. 819) kernel object.*

Detailed Description

Represents a **Semaphore** (p. 819) in user space.

Locking will be performed in the kernel, so this is only a proxy.

The documentation for this class was generated from the following file:

- apps/library/Semaphore.h

A.4.279 task::semaphore::test::SemaphoreTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()
- virtual const char * **getName** ()

Private Member Functions

- void **testSemaphore_01** ()
- void **testSemaphore_02** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- SemaphoreTestCase.h

A.4.280 io::driver::interrupt::SlavePICController Class Reference

Inherits **io::driver::interrupt::PICController**.

Public Member Functions

- **SlavePICController** (uint16_t basePort, **MasterPICController** &**master**, uint8_t **masterPin**)
- virtual void **maskIRQs** (cpu::level::IRQLevel untilIRQL)

  Masks all IRQs up to a given IRQ level (including).
- virtual void **init** (uint8_t **vectorBase**, cpu::level::IRQLevel **baseIRQL**)
- virtual cpu::level::IRQLevel **getLevel** (uint8_t pin)

  Returns the IRQ level associated with a PIC input.

Protected Member Functions

- virtual void **sendEOI** ()

  Sends an EOI to this PIC and its master PIC, if any.

Private Member Functions

- virtual void **writeICW3** ()

  Writes the ICW3.

Private Attributes

- **MasterPICController** & **master**

  The associated master PIC.
- uint8_t const **masterPin**

  The master PIN this slave is attached to.
- cpu::level::IRQLevel **baseIRQL**

  The base IRQ level initialized by init().

Additional Inherited Members

Member Function Documentation

virtual void io::driver::interrupt::SlavePICController::maskIRQs (cpu::level::IRQLevel *untilIRQL*) [virtual] Masks all IRQs up to a given IRQ level (including).

All higher IRQLs are unmasked.

Parameters

untilIRQL	The IRQ level up to which IRQs are to be masked.

Implements **io::driver::interrupt::PICDevice** (p. 710).

virtual cpu::level::IRQLevel io::driver::interrupt::SlavePICController::getLevel (uint8_t *pin*) [inline], [virtual] Returns the IRQ level associated with a PIC input.

Parameters

pin	The PIC input.

@

@plus@

@plus -@

@

@

@skip
Returns

The associated IRQ level.

Implements **io::driver::interrupt::PICDevice** (p. 709).

References io::driver::interrupt::PICController::NumInputs.

The documentation for this class was generated from the following file:

- SlavePICController.h

A.4.281 io::driver::speaker::SpeakerDevice Class Reference

PC Speaker device.

Inherits **io::driver::classes::AudioClass**.

Public Member Functions

- virtual void **setFrequency** (int newFrequency)

  Set the frequency of the sound.
- virtual void **on** ()

  Turns the device on.
- virtual void **off** ()

  Turns the device off.

Private Attributes

- **timer::PIT8254Counter** ∗ **myCounter**

  My cached pit counter.

Additional Inherited Members

Detailed Description

PC Speaker device.

The documentation for this class was generated from the following file:

- SpeakerDevice.h

A.4.282 io::driver::speaker::SpeakerDriver Class Reference

Driver (p. 455) for the Speaker.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Additional Inherited Members

Detailed Description

Driver (p. 455) for the Speaker.

The documentation for this class was generated from the following file:

- SpeakerDriver.h

A.4.283 task::lock::SpinLock Class Reference

Mutex for a critical area using busy waiting.

Data Structures

- class **Lock**

  Locker class which acquires a spin lock in the constructor and releases it in the destructor.

Public Member Functions

- **SpinLock** (cpu::level::IRQLevel **irqLevel**)

  Constructor.

- cpu::level::IRQLevel **getIRQLevel** () const

  Returns the associated IRQ level.

- void **setIRQLevel** (cpu::level::IRQLevel **irqLevel**)

  Changes the associated IRQ level.

Private Member Functions

- cpu::level::IRQLevel **acquireLock** ()

  Acquires the spin lock using busy waiting.

- void **releaseLock** (cpu::level::IRQLevel **irqLevel**)

  Releases the spin lock and restores the IRQ level.

- void **releaseLock** ()

  Releases the spin lock only.

- void **yield** (cpu::level::IRQLevel **irqLevel**)

  Releases and re-acquires the spin lock, requesting the scheduler to release the time slice in between.

Private Attributes

- volatile uint32_t **lockValue**

  The lock value.

- cpu::level::IRQLevel **irqLevel**

  The IRQ level to enter while holding the spin lock.

Static Private Attributes

- static const int **UNLOCKED** = 0

  Represents the lock state 'Unlocked'.

- static const int **LOCKED** = 1

  Represents the lock state 'Locked'.

Friends

- class **scheduler::Scheduler**

Detailed Description

Mutex for a critical area using busy waiting.

Constructor & Destructor Documentation

task::lock::SpinLock::SpinLock (cpu::level::IRQLevel *irqLevel*) Constructor.

Sets the spin lock to UNLOCKED.

Parameters

irqLevel	The IRQ level to enter while holding the spin lock. Needs to be at least IRQL_DISPATCH.

Member Function Documentation

cpu::level::IRQLevel task::lock::SpinLock::acquireLock ( ) [private] Acquires the spin lock using busy waiting.

@

@plus@

@plus -@

@

@

@skip
Returns

The old IRQ level.

Referenced by task::lock::SpinLock::Lock::Lock().

void task::lock::SpinLock::releaseLock (cpu::level::IRQLevel *irqLevel*) [private] Releases the spin lock and restores the IRQ level.

Requires **acquireLock()** (p. 826) to have been called before!

Parameters

irqLevel	The IRQ level to restore.

Referenced by task::lock::SpinLock::Lock::~Lock().

void task::lock::SpinLock::releaseLock ( ) [inline], [private] Releases the spin lock only.

Called by the Scheduler just after the current **Thread** (p. 865) has been marked as being BLOCKED.

References lockValue, and UNLOCKED.

void task::lock::SpinLock::yield (cpu::level::IRQLevel *irqLevel*) [private]

Releases and re-acquires the spin lock, requesting the scheduler to release the time slice in between.

May be called only if the spin lock has been acquired at PASSIVE level.

Referenced by task::lock::SpinLock::Lock::yield().

The documentation for this class was generated from the following file:

- SpinLock.h

A.4.284 task::spinlock::test::SpinLockTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual char const ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- SpinLockTestCase.h

A.4.285 task::spinlock::test::SpinLockTestThread Class Reference

Inherits **task::Thread**.

Public Member Functions

- **SpinLockTestThread** (**Process** &parent, **lock::Semaphore** &exitSem, uint32_t amountOfRuns, **Counter** ∗theCounter)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!
- virtual const char ∗ **getName** ()

Private Attributes

- **lock::Semaphore** & **exitSem**
- **Counter** ∗ **myCounter**
- uint32_t **runs**

Additional Inherited Members

Member Function Documentation

virtual void task::spinlock::test::SpinLockTestThread::run ( ) [virtual]
IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **SpinLockTestThread.h**

A.4.286 boot::SplashScreen Class Reference

Class for writing a cool logo of the FHDW OS.

Public Member Functions

- **SplashScreen** (**io::driver::graphics::TextDevice** &**device**)

  Constructor.

- void **makeSplashScreen** ()

  Print out the logo to device (VGA)

Private Attributes

- **io::driver::graphics::TextDevice** & **device**

  chached TextDevice

Detailed Description

Class for writing a cool logo of the FHDW OS.

Constructor & Destructor Documentation

boot::SplashScreen::SplashScreen (io::driver::graphics::TextDevice & *device*) Constructor.

Parameters

device	The TextDevice to write the logo to. Will be chached in pritate device attribute.

The documentation for this class was generated from the following file:

- SplashScreen.h

A.4.287 tool::test::StringTestCase Class Reference

Test case for string functions.

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()

- virtual char const * **getName** ()

Private Member Functions

- void **testStrLen** ()

  Tests strlen().

- void **testStrCmp** ()

  Tests strcmp().

- void **testStrCat** ()

  *Tests **strcat()*** (p. *972).*

Additional Inherited Members

Detailed Description

Test case for string functions.

The documentation for this class was generated from the following file:

- StringTestCase.h

A.4.288 ipc::SysCallHandler Class Reference

Public Member Functions

- ~**SysCallHandler** ()

  Destructor. Removes the call gate.

- **memory::Selector getCallGate** () const

  Returns the call gate.

Static Public Member Functions

- static **SysCallHandler** & **getInstance** ()

  Returns the singleton instance of this class.

Private Member Functions

- **SysCallHandler** ()

  Constructor. Creates a call gate to let ring 3 process call the kernel.

- bool **handleSysCall** (**Request** &request, uint32_t size, **Participant** sender, **Participant** receiver)

  *Called by **sysCallHandler()** (p. 832).*

- bool **handleKernelRequest** (**Request** &request, **Participant** sender)

  Handles a kernel request.

- bool **handleIPCRequest** (**Request** &request, uint32_t size, **task::Process** &sender, **task::Process** &receiver)

  Handles an IPC request.

Static Private Member Functions

- static void **init** ()

  Initializes the singleton instance of this class.

- static bool **sysCallHandler** (**Request** &request, uint32_t size, uint32_t sender, uint32_t receiver)

  Called by assembler code to handle a system call.

- static bool **invalidSystemCall** (**Request** &, **Participant**)

  Called when an invalid system call is issued.

Private Attributes

- uint32_t **numSystemCalls**

  The index of the highest supported system call plus one.

- **memory::Selector callGate**

  The call gate ring 3 processes need to use to call the kernel.

Static Private Attributes

- static uint32_t const **NumParameters** = 4

  Number of 32-bit parameters the call gate expects.

- static **SysCallHandler theInstance**

  The singleton instance of this class.

- static RequestHandler **table** [ ]

  The system call table.

Friends

- class **boot::BootManager**

Member Function Documentation

static bool ipc::SysCallHandler::sysCallHandler (Request & *request*, uint32_t *size*, uint32_t *sender*, uint32_t *receiver*) [static], [private]
Called by assembler code to handle a system call.

Parameters

request	The **Request** (p. 775) to handle.
size	The size of the **Request** (p. 775) in bytes.
sender	The sender of the **Request** (p. 775).
receiver	The receiver of the **Request** (p. 775).

@

@plus@

@plus -@

@

@

@skip
Returns

True if the **Request** (p. 775) has been successfully handled, else false.

bool ipc::SysCallHandler::handleSysCall (Request & *request*, uint32_t *size*, Participant *sender*, Participant *receiver*) [private] Called by **sysCallHandler()** (p. 832).

Parameters

request	The **Request** (p. 775) to handle.
size	The size of the **Request** (p. 775) in bytes.
sender	The sender of the **Request** (p. 775).
receiver	The receiver of the **Request** (p. 775).

@

@plus@

@plus -@

@

@

@skip

Returns

True if the **Request** (p. 775) has been successfully handled, else false.

bool ipc::SysCallHandler::handleKernelRequest (Request & *request,* Participant *sender*) [private] Handles a kernel request.

Parameters

request	The **Request** (p. 775) to handle.
sender	The sender of the **Request** (p. 775).

@

@plus@

@plus -@

@

@

@skip

Returns

True if the **Request** (p. 775) has been successfully handled, else false.

bool ipc::SysCallHandler::handleIPCRequest (Request & *request,* uint32_t *size,* task::Process & *sender,* task::Process & *receiver*) [private] Handles an IPC request.

Parameters

request	The **Request** (p. 775) to handle.
size	The size of the **Request** (p. 775) in bytes.
sender	The sender of the **Request** (p. 775).
receiver	The receiver of the **Request** (p. 775).

@

@plus@

@plus -@

@

@

@skip

Returns

True if the **Request** (p. 775) has been successfully handled, else false.

static bool ipc::SysCallHandler::invalidSystemCall (Request & , Participant) [inline], [static], [private] Called when an invalid system call is issued.

@

@plus@

@plus -@

@

@

@skip

Returns

Always false.

The documentation for this class was generated from the following file:

- **SysCallHandler.h**

A.4.289 ipc::test::SysCallTestCase Class Reference

Tests the communication with the kernel via system calls.

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual char const ∗ **getName** ()

Private Member Functions

- void **testGetVersion** ()

Additional Inherited Members

Detailed Description

Tests the communication with the kernel via system calls.

The documentation for this class was generated from the following file:

- SysCallTestCase.h

A.4.290 lib::System Class Reference

The class **System** (p. 835) encapsulates many functions necessary for initializing a user application, managing the communication with the os kernel and managing memory.

Data Structures

- class **SystemCallFailed**

  Thrown if a system call fails.

Public Member Functions

- template<typename T >
  void **syscall** (T &request) throw (SystemCallFailed)

  Executes a kernel system call.
- **ipc::CommandRelay** & **getRelay** ()

Getter for the CommandRealy.

- ~**System** ()

  Desctructor.

- **memory::allocator::Allocator** & **getAllocator** ()

  Getter for the Allocator.

Static Public Member Functions

- static void **init** (**ipc::CommandRelay** &**relay**)

  ATTENTION: CALL THIS METHOD FIRST IN EVERY APPLICATION!

- static **System** & **getInstance** ()

  *Getter for the instance of this **System** (p. 835), due to Singleton pattern.*

Private Member Functions

- **System** (**ipc::CommandRelay** &**relay**)

  private constructor

Private Attributes

- **ipc::CommandRelay** & **relay**

  CommandRelay used to communicate with the os kernel.

- **lib::runtime::UserEnvironment environ**

  Responsible for managing memory requests, is used by the Allocator as well.

- **memory::allocator::Allocator allocator**

  Responsible for creating objects on the heap and manages the heaps memory.

Static Private Attributes

- static char **instance** [ ]

  *Contains the instance of the **System** (p. 835).*

Detailed Description

The class **System** (p. 835) encapsulates many functions necessary for initializing a user application, managing the communication with the os kernel and managing memory.

Member Function Documentation

static void lib::System::init (ipc::CommandRelay & *relay*) [static]
ATTENTION: CALL THIS METHOD FIRST IN EVERY APPLICATION!

Initializes the **System** (p. 835) and everything necessary to use the application.

Parameters

relay	of class CommandRelay is the CommandRelay used for communication with the os kernel

template<typename T > void lib::System::syscall (T & *request*) throw SystemCallFailed) [inline] Executes a kernel system call.

Parameters

request	A refernce to the Request object.
size	The size in bytes of the Request object.

Exceptions

SystemCallFailed *(p. 839)*	if executing the system call fails.

References relay, and ipc::CommandRelay::relay().

ipc::CommandRelay& lib::System::getRelay ( ) [inline] Getter for the CommandRealy.

@

@plus@

@plus -@

@

@

@skip
Returns

the CommandRelay

References relay.

memory::allocator::Allocator& lib::System::getAllocator ( ) [inline]
Getter for the Allocator.

@

@plus@

@plus -@

@

@

@skip
Returns

the allocator

References allocator.

static System& lib::System::getInstance ( ) [static] Getter for the instance of this **System** (p. 835), due to Singleton pattern.

@

@plus@

@plus -@

@

@

@skip
Returns

the instance

The documentation for this class was generated from the following file:

- System.h

A.4.291 lib::System::SystemCallFailed Class Reference

Thrown if a system call fails.

Inherits exception.

Public Member Functions

- **SystemCallFailed** (uint32_t **sysCallId**)

  Constructor.

- **SystemCallFailed** (**SystemCallFailed** const &other)

  Copy constructor.

- **SystemCallFailed** & **operator=** (**SystemCallFailed** const &other)

  Copy assignment operator.

- virtual ∼**SystemCallFailed** () throw ()

  Destructor.

- virtual char const ∗ **what** () const throw ()

Private Attributes

- uint32_t **sysCallId**

  The identifier of the failed system call.

- char ∗ **errorMessage**

  The error message.

Detailed Description

Thrown if a system call fails.

Constructor & Destructor Documentation

lib::System::SystemCallFailed::SystemCallFailed (uint32_t *sysCallId*)
Constructor.

Parameters

sysCallId	The identifier of the failed system call.

lib::System::SystemCallFailed::SystemCallFailed (SystemCallFailed const & *other*) Copy constructor.

Parameters

other	The object to copy.

Member Function Documentation

SystemCallFailed& lib::System::SystemCallFailed::operator= (SystemCallFailed const & *other*) Copy assignment operator.

Parameters

other	The object to assign.

@

@plus@

@plus -@

@

@

@skip

Returns

∗this

The documentation for this class was generated from the following file:

- System.h

A.4.292 memory::SystemSegmentDescriptor Class Reference

Describes a system segment descriptor.

Inherits **memory::Descriptor**.

Inherited by **memory::GateDescriptor**, and **memory::TSSDescriptor**.

Public Types

- enum **SystemSegmentType** {
  SystemSegmentTSSAvailable = 1, **SystemSegmentLDT** = 2, **SystemSegmentTSSBusy** = 3, **SystemSegmentCallGate** = 4, **SystemSegmentTaskGate** = 5, **SystemSegmentInterruptGate** = 6, **SystemSegmentTrapGate** = 7 }

  Describes the type of a system segment.

Public Member Functions

- **SystemSegmentType getSystemSegmentType** () const

  Returns the system segment type.

- **SystemSegmentDescriptor** & **setSystemSegmentType** (**SystemSegmentType type**)

  Sets the system segment type.

Private Types

- typedef **tool::BitField**
  < uint32_t, 1, 8, 3 > **SystemSegmentTypeField**

Additional Inherited Members

Detailed Description

Describes a system segment descriptor.

Member Enumeration Documentation

enum memory::SystemSegmentDescriptor::SystemSegmentType Describes the type of a system segment.

Enumerator

SystemSegmentTSSAvailable available TSS

SystemSegmentLDT LDT (local descriptor table)

SystemSegmentTSSBusy busy TSS

SystemSegmentCallGate call gate

SystemSegmentTaskGate task gate
SystemSegmentInterruptGate interrupt gate
SystemSegmentTrapGate trap gate

Member Function Documentation

SystemSegmentDescriptor& memory::SystemSegmentDescriptor::setSystemSegmentType (SystemSegmentType *type*) [inline] Sets the system segment type.

Parameters

type	The system segment type.

@

@plus@

@plus -@

@

@

@skip
Returns

$*$this

References memory::Descriptor::data, and tool::BitField$<$ Base, Index, BitPosition, Length $>$::set().

The documentation for this class was generated from the following file:

- Descriptor.h

A.4.293 task::tasklet::Tasklet Class Reference

A **Tasklet** (p. 842) is a piece of code that is run at IRQL_DISPATCH before falling down to IRQL_PASSIVE.

Inherited by **boot::CleanupTasklet**, **io::driver::block::ata::AtaTasklet**, **io::driver::keycode::KeycodeTasklet**, **io::driver::rtc::RTCTasklet**, and **task::tasklet::test::TestTasklet**.

Public Member Functions

- virtual ~**Tasklet** ()

  Destructor.

- virtual TaskletState **getState** () const =0

  Returns the tasklet state.

- virtual void **work** ()=0

  The actual work function per tasklet.

Detailed Description

A **Tasklet** (p. 842) is a piece of code that is run at IRQL_DISPATCH before falling down to IRQL_PASSIVE.

It is useful for "bottom halves" of interrupt handlers as tasklets can be interrupted by any hardware interrupt (but not preempted by other threads) and so they are run at a higher priority as "normal" threads (which run at IRQL_PASSIVE and can be preempted at any time). However, this means that each **Tasklet** (p. 842)'s **work()** (p. 843) method must not execute too long and must not block. If you need to block, e.g. because you have to read from a Pipe, use a dedicated worker thread.

Member Function Documentation

virtual TaskletState task::tasklet::Tasklet::getState ( ) const [pure virtual] Returns the tasklet state.

Only if a **Tasklet** (p. 842) returns HASWORK it will have its **work()** (p. 843) method invoked.

Implemented in **io::driver::block::ata::AtaTasklet** (p. 336), **io::driver::keycode::KeycodeTasklet** (p. 589), **io::driver::rtc::RTCTasklet** (p. 799), **boot::CleanupTasklet** (p. 384), and **task::tasklet::test::TestTasklet** (p. 857).

The documentation for this class was generated from the following file:

- Tasklet.h

A.4.294 task::tasklet::TaskletManager Class Reference

The **TaskletManager** (p. 844) manages all Tasklets in the system.

Public Member Functions

- virtual ∼**TaskletManager** ()

  Destructor. Deletes all registered Tasklets.

- void **addTasklet** (**Tasklet** ∗tasklet)

  *Registers a **Tasklet** (p. 842).*

Static Public Member Functions

- static **TaskletManager** & **getInstance** ()

  *Returns the singleton **TaskletManager** (p. 844) instance.*

Private Types

- typedef
  tool::collection::LinkedList
  < **Tasklet** ∗ > **Tasklets**

  Type of a list of Tasklets.

Private Member Functions

- **TaskletManager** ()

  Constructor.

- void **work** ()

  Runs all Tasklets which have something to do once.

Static Private Member Functions

- static void **init** ()

  Initializes the singleton instance of this class.

Private Attributes

- **task::lock::SpinLock mutex**

  *Spinlock protecting the **TaskletManager** (p. 844)'s data.*

- **Tasklets tasklets**

  The registered Tasklets.

Static Private Attributes

- static **TaskletManager instance**

  *The singleton **TaskletManager** (p. 844) instance.*

Friends

- class **boot::BootManager**
- class **task::scheduler::SchedulerHandler**

Detailed Description

The **TaskletManager** (p. 844) manages all Tasklets in the system.

Member Function Documentation

void task::tasklet::TaskletManager::addTasklet (Tasklet * *tasklet*) `[inline]` Registers a **Tasklet** (p. 842).

To deregister it again, let its getState() method return TOKILL.

Parameters

tasklet	The **Tasklet** (p. 842) to register.

References tool::collection::LinkedList< T >::add(), mutex, and tasklets.

The documentation for this class was generated from the following file:

- TaskletManager.h

A.4.295 task::TaskStateSegment Struct Reference

TSS structure described by the intel documentation.

Public Types

- enum { **TraceOff** = 0 << 0, **TraceOn** = 1 << 0 }

Data Fields

- enum task::TaskStateSegment:: { ... } **__attribute__**
- uint16_t **link**
- uint16_t **__link**
- uint32_t **esp0**
- uint16_t **ss0**
- uint16_t **__ss0**
- uint32_t **esp1**
- uint16_t **ss1**
- uint16_t **__ss1**
- uint32_t **esp2**
- uint16_t **ss2**
- uint16_t **__ss2**
- uint32_t **cr3**
- uint32_t **eip**
- uint32_t **eflags**
- uint32_t **eax**
- uint32_t **ecx**
- uint32_t **edx**
- uint32_t **ebx**
- uint32_t **esp**
- uint32_t **ebp**
- uint32_t **esi**
- uint32_t **edi**
- uint16_t **es**

- uint16_t **__es**
- uint16_t **cs**
- uint16_t **__cs**
- uint16_t **ss**
- uint16_t **__ss**
- uint16_t **ds**
- uint16_t **__ds**
- uint16_t **fs**
- uint16_t **__fs**
- uint16_t **gs**
- uint16_t **__gs**
- uint16_t **ldtr**
- uint16_t **__ldtr**
- uint16_t **trace**
- uint16_t **io_map_base_addr**

Detailed Description

TSS structure described by the intel documentation.

reserved fields begin with "__"

The documentation for this struct was generated from the following file:

- TaskStateSegment.h

A.4.296 runtime::TCB Struct Reference

The **TCB** (p. 847) (Thread Control Block).

Public Member Functions

- void **initTLS** ()

  *Initializes the TLS area placed directly below the **TCB** (p. 847).*

Static Public Member Functions

- static uint32_t **getNumberOfTLSSlots** ()

  Returns the number of 32-bit slots in the TLS area.

Data Fields

- **TCB** ∗ **tcb**

  *Self-referential flat pointer to the **TCB** (p. 847).*

- void ∗ **reserved** [5]

  32-bit elements required by gcc/libgcc.

Private Member Functions

- void ∗∗ **getStartOfTLSArea** (uint32_t numTLSSlots)

  Returns a pointer to the start of the TLS area.

Detailed Description

The **TCB** (p. 847) (Thread Control Block).

It is not directly used by the FHDW Operating System, but is necessary for the correct execution of some gcc and libgcc code which requires the **TCB** (p. 847) to exist on the GNU/Linux platform. The **TCB** (p. 847) can be found at offset zero relative to the start of the segment pointed to by the "gs" segment register.

The TLS area is placed directly below the **TCB** (p. 847) and is thus accessed by negative (!) offsets relative to the **TCB** (p. 847). Don't blame me for this cumbersome memory layout; I didn't invent it. It's officially specified for the ELF ABI based on the IA-32 platform which is expected by gcc and its libraries. Read `http://www.akkadia.org/drepper/tls.pdf` for details.

Member Function Documentation

void runtime::TCB::initTLS ( ) Initializes the TLS area placed directly below the **TCB** (p. 847).

The caller must ensure that the TLS area is initialized with zeros before calling this operation. This is because this operation does not override non-zero slots.

This is necessary because the exception-handling TLS slot is initialized at the beginning of the kernel entry point even before we have a chance to call this operation, and we are not allowed to override this slot with zero.

void∗∗ runtime::TCB::getStartOfTLSArea (uint32_t *numTLSSlots*) [inline], [private] Returns a pointer to the start of the TLS area.

Parameters

numTLS-Slots	The number of slots in the TLS area.

Field Documentation

TCB∗ runtime::TCB::tcb Self-referential flat pointer to the **TCB** (p. 847).

This allows the **TCB** (p. 847) to be addressed using the data segment.

void∗ runtime::TCB::reserved[5] 32-bit elements required by gcc/libgcc.

It is 5 because gcc requires the sixth 32-bit word in the **TCB** (p. 847) (i.e. reserved[4]) to store the so-called "stack canary" which is used to detect a stack overflow.

The documentation for this struct was generated from the following file:

- **kernel/runtime/runtime.h**

A.4.297 tool::Terminal Class Reference

Inherits **task::Thread**.

Public Member Functions

- **Terminal** (**io::console::TextConsole** &out, **task::pipeandfilter::Pipe**< char > &in, **task::Process** &myproc)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- **io::console::TextConsole** & **out**
- **task::pipeandfilter::Pipe**< char > & **in**

Additional Inherited Members

Member Function Documentation

virtual void tool::Terminal::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- Terminal.h

A.4.298 test::TestCase Class Reference

Abstract class for the individual test cases.

Inherited by **io::driver::block::ata::test::AtaReadWriteTest**, **io::driver::block::ata::test::AtaReadWriteTest2**, **io::driver::block::ata::test::AtaReadWriteTest3**, **io::driver::charlayout::test::CharLayoutTestCase**, **io::driver::graphics::vga::test::VgaCharTestCase**, **io::driver::keycode::test::KeyboardTestCase**, **io::driver::test::DriverFrameworkTestCase**, **io::vfs::test::VFSTestCase**, **ipc::test::IPCTestCase**, **ipc::test::SysCallTestCase**, **memory::allocator::test::AllocatorOutOfMemoryTestCase**, **memory::allocator::test::AllocatorTestCase**, **memory::lmm::test::LASMTest**, **memory::paging::test::PagingTestCase**, **object::test::RefTestCase**, **runtime::test::ExceptionTestCase**, **runtime::test::RTTITestCase**, **task::priorityinheritance::test::PriorityInheritanceTestCase**, **task::semaphore::test::SemaphoreTestCase**, **task::semaphore::test::ThreadASemaphoreTestCase** [private], **task::spinlock::test::SpinLockTestCase**, **test::memory::allocator::AllocatorTestCase**, **tool::collection::test::ArrayListTestCase**, **tool::collection::test::LinearMapTestCase**, **tool::collection::test::LinkedListTestCase**, and **tool::test::StringTestCase**.

Public Member Functions

- virtual void **run** ()=0
- virtual void **runTestCase** ()
- virtual const char ∗ **getName** ()=0

Protected Member Functions

- void **assert** (**io::driver::keycode::Keycode** expected, **io::driver::keycode-::Keycode** value)
- void **assert** (void ∗expected, void ∗value)
- void **assert** (unsigned int expected, unsigned int value)
- void **assertTrue** (bool value)
- void **assertFalse** (bool value)

Private Member Functions

- void **printFailure** (unsigned int expected, unsigned int value)
- void **printFailure** (bool expected, bool value)

Detailed Description

Abstract class for the individual test cases.

The documentation for this class was generated from the following file:

- TestCase.h

A.4.299 io::driver::test::TestDevice Class Reference

Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason.

Inherits **io::driver::Device**.

Additional Inherited Members

Detailed Description

Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason.

Due to this it provides no functional extension.

The documentation for this class was generated from the following file:

- TestDevice.h

A.4.300 io::driver::test::TestDevice2 Class Reference

Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason.

Inherits **io::driver::Device**.

Additional Inherited Members

Detailed Description

Class testDevice is an implementation of abstract class **Device** (p. 436) for testing reason.

Due to this it provides no functional extension.

The documentation for this class was generated from the following file:

- TestDevice2.h

A.4.301 io::driver::test::TestDriver Class Reference

Class testDriver is an implementation of abstract class **Driver** (p. 455) for testing reason.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

Checks if a new device for this driver is available and initiates a corresponding device-object.

Private Attributes

- ::**io::driver::Device** ∗ **myDev**

Additional Inherited Members

Detailed Description

Class testDriver is an implementation of abstract class **Driver** (p. 455) for testing reason.

Due to this it provides no functional extension.

The documentation for this class was generated from the following file:

- TestDriver.h

A.4.302 io::driver::test::TestDriver2 Class Reference

Class testDriver2 is an implementation of abstract class **Driver** (p. 455) for testing reason.

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Initializes a testDevice object.

Private Attributes

- ::**io::driver::Device** ∗ **myDev**

Additional Inherited Members

Detailed Description

Class testDriver2 is an implementation of abstract class **Driver** (p. 455) for testing reason.

Due to this it provides no functional extension.

The documentation for this class was generated from the following file:

- TestDriver2.h

A.4.303 task::pipeandfilter::test::TesterPipe< T > Class Template Reference

Pipe (p. 715) implementation for testing propose, compares the incoming values with the expected ones.

Inherits **task::pipeandfilter::Pipe**< **T** >.

Public Member Functions

- **TesterPipe** (**Pipe**< T > ∗paramExpectedData)
- virtual bool **isEmpty** ()
- virtual void **write** (T data)
- virtual T **read** ()

Private Attributes

- **Pipe**< T > ∗ **expectedData**

Detailed Description

template<**typename T**>**class task::pipeandfilter::test::TesterPipe**< **T** >

Pipe (p. 715) implementation for testing propose, compares the incoming values with the expected ones.

The documentation for this class was generated from the following file:

- TesterPipe.h

A.4.304 test::TestManager Class Reference

Public Member Functions

- void **runAll** ()

  Executes all TestCases in a separate thread.

- void **runTestCase** (**TestCase** &testCase)

  *Executes a single **TestCase** (p. 850) in the current thread.*

Static Public Member Functions

- static **TestManager** & **getInstance** ()

  Returns.

Private Member Functions

- **TestManager** ()

  Private constructor.

- void **doRunAll** ()

  Executes all TestCases.

- void **doRunTestCase** (**TestCase** &testCase)

  *Executes a single **TestCase** (p. 850).*

Static Private Member Functions

- static void **init** ()

  Initializes the singleton instance of this class.

Static Private Attributes

- static **TestManager theInstance**

  *The singleton instance of the **TestManager** (p. 855).*

- static bool **Testing**

  Set to true if testing is enabled.

Friends

- class **boot::BootManager**

Member Function Documentation

static TestManager& test::TestManager::getInstance ( ) [inline], [static] Returns.

@

@plus@

@plus -@

@

@

@skip
Returns

the instance of this singleton **TestManager** (p. 855).

References theInstance.

void test::TestManager::runTestCase (TestCase & *testCase*) [inline]
Executes a single **TestCase** (p. 850) in the current thread.

Parameters

testCase	A reference to the test case to be executed.

References Testing.

The documentation for this class was generated from the following file:

- TestManager.h

A.4.305 task::tasklet::test::TestTasklet Class Reference

Inherits **task::tasklet::Tasklet**.

Public Member Functions

- **TestTasklet** (TaskletState defs)
- virtual TaskletState **getState** () const

  Returns the tasklet state.
- virtual void **work** ()

  The actual work function per tasklet.

Private Attributes

- TaskletState **defaultState**

Member Function Documentation

virtual TaskletState task::tasklet::test::TestTasklet::getState ( ) const `[virtual]` Returns the tasklet state.

Only if a **Tasklet** (p. 842) returns HASWORK it will have its **work()** (p. 857) method invoked.

Implements **task::tasklet::Tasklet** (p. 843).

The documentation for this class was generated from the following file:

- TestTasklet.h

A.4.306 io::driver::graphics::TextChar Class Reference

Class for the characters to print at the console.

Public Member Functions

- **TextChar** (char **ch**=0, io::console::Color **textColor**=io::console::Color::GREY, io::console::Color **backgroundColor**=io::console::Color::BLACK)

  Constructor.
- char **getChar** () const

  Getter for the character.
- io::console::Color **getTextColor** () const

Getter for the text color.

- io::console::Color **getBackgroundColor** () const

  Getter for the text background color.

Private Attributes

- char const **ch**

  The character.

- io::console::Color const **textColor**

  Text color.

- io::console::Color const **backgroundColor**

  Background color.

Detailed Description

Class for the characters to print at the console.

Constructor & Destructor Documentation

io::driver::graphics::TextChar::TextChar (char *ch* = `0`, io::console::Color *textColor* = `io::console::Color::GREY`**, io::console::Color *backgroundColor* =** `io::console::Color::BLACK` **)** `[inline]` Constructor.

Parameters

ch	character (ASCI)
textColor	textcolor of the character
backgroundColor	background color of the character

Member Function Documentation

char io::driver::graphics::TextChar::getChar ( ) const `[inline]` Getter for the character.

@

@plus@

@plus -@

@

@

@skip
Returns

The current set character

References ch.

io::console::Color io::driver::graphics::TextChar::getTextColor ( ) const [inline] Getter for the text color.

@

@plus@

@plus -@

@

@

@skip
Returns

Current set text color

References textColor.

io::console::Color io::driver::graphics::TextChar::getBackgroundColor ( ) const [inline] Getter for the text background color.

@

@plus@

@plus -@

@

@

@skip
Returns

Current set background color

References backgroundColor.

The documentation for this class was generated from the following file:

- **TextChar.h**

A.4.307 io::console::TextConsole Class Reference

The top class for all text output consoles.

Inherited by **boot::BootConsole**, and **io::console::DefaultConsole**.

Public Member Functions

- virtual void **setTextColor** (Color color)=0

  Sets the text color.

- virtual void **setBackgroundColor** (Color color)=0

  Sets the text background color.

- virtual void **write** (char s)=0

  prints one character on the console

- virtual void **write** (char const ∗s)=0

  prints a primitive string on the console

- virtual void **writeLine** (char const ∗s)=0

  *write(s); **newLine()** (*p. *860);*

- virtual void **newLine** ()=0

  sets a new line on the console

- virtual void **scrollUp** ()=0

  scrolls the screen up useful when writing at the last line of the console

- virtual void **clear** ()=0

  Clears the screen.

- virtual void **activate** ()=0

  Activates the console.

- virtual void **deactivate** ()=0

  Deactivates the console.

Detailed Description

The top class for all text output consoles.

Member Function Documentation

virtual void io::console::TextConsole::setTextColor (Color *color*) [pure virtual] Sets the text color.

Parameters

color	The color number to set

Implemented in **boot::BootConsole** (p. 362), and **io::console::DefaultConsole** (p. 425).

virtual void io::console::TextConsole::setBackgroundColor (Color *color*) [pure virtual] Sets the text background color.

Parameters

color	The color number to set

Implemented in **boot::BootConsole** (p. 362), and **io::console::DefaultConsole** (p. 425).

virtual void io::console::TextConsole::write (char *s*) [pure virtual] prints one character on the console

Parameters

s	the character to print

Implemented in **boot::BootConsole** (p. 362), and **io::console::DefaultConsole** (p. 426).

virtual void io::console::TextConsole::write (char const * *s*) [pure virtual] prints a primitive string on the console

Parameters

s	the startpoint of the string

Implemented in **boot::BootConsole** (p. 363), and **io::console::DefaultConsole** (p. 426).

virtual void io::console::TextConsole::writeLine (char const * *s*) [pure virtual] write(s); **newLine()** (p. 860);

Parameters

s	

Implemented in **boot::BootConsole** (p. 363), and **io::console::DefaultConsole** (p. 426).

virtual void io::console::TextConsole::clear ( ) [pure virtual] Clears the screen.

Well only useful on screens (printer would laugh at you :))

Implemented in **boot::BootConsole** (p. 360), and **io::console::DefaultConsole** (p. 426).

The documentation for this class was generated from the following file:

- TextConsole.h

A.4.308 io::driver::graphics::TextDevice Class Reference

Driver (p. 455) for write signs in a text device and to move the cursor of the text device.

Inherits **io::driver::Device**.

Inherited by **io::driver::graphics::vga::VgaTextDevice**.

Public Member Functions

- **TextDevice** (int ClassId, uint32_t **columns**, uint32_t rows)

  Default constructor.
- uint32_t **getNumCols** () const
- uint32_t **getNumRows** () const

- virtual void **writeAt** (**Position** pos, **TextChar** ch)=0

  Writes the.

- virtual **Cursor** ∗ **getCursor** ()=0

  Returns the associated cursor.

- virtual **TextChar getCharForPosition** (**Position** pos)=0
- virtual void **activate** ()=0

  Activates the device.

- virtual void **deactivate** ()=0

  Deactivates the device.

Private Attributes

- uint32_t const **columns**

  Device *(p. 436) information (columns and rows)*

- uint32_t const **rows**

Detailed Description

Driver (p. 455) for write signs in a text device and to move the cursor of the text device.

Constructor & Destructor Documentation

io::driver::graphics::TextDevice::TextDevice (int *ClassId,* uint32_t *columns,* uint32_t *rows*) [inline] Default constructor.

Parameters

columns	The count of columns which this **TextDevice** (p. 862) provide
rows	The count of rows which this **TextDevice** (p. 862) provide

Member Function Documentation

uint32_t io::driver::graphics::TextDevice::getNumCols ( ) const [inline]

@

@plus@

@plus -@

@

@

@skip
Returns

the count of columns which this **TextDevice** (p. 862) provide

References columns.

uint32_t io::driver::graphics::TextDevice::getNumRows ( ) const [inline]
@

@plus@

@plus -@

@

@

@skip
Returns

the count of rows which this **TextDevice** (p. 862) provide

virtual void io::driver::graphics::TextDevice::writeAt (Position *pos,* TextChar *ch*) [pure virtual] Writes the.

Parameters

ch	to position (
pos).	

Implemented in **io::driver::graphics::vga::VgaTextDevice** (p. 908).

virtual TextChar io::driver::graphics::TextDevice::getCharForPosition (Position *pos*) [pure virtual] @

@plus@

@plus -@

@

@

@skip

Returns

the **TextChar** (p. 857) for the position (

Parameters

pos)	

Implemented in **io::driver::graphics::vga::VgaTextDevice** (p. 908).

The documentation for this class was generated from the following file:

- TextDevice.h

A.4.309 task::Thread Class Reference

Thread (p. 865) is an abstract class.

Inherited by **boot::BootThread**, **boot::UserSpaceInitThread**, **ClockThread**, **io::driver::block::ata::AtaBusWorkerThread**, **ipc::test::IPCReceiverThread**, **ipc::test::IPCSenderThread**, **LockTesterThreadDown**, **LockTesterThreadUp**, **task::pipeandfilter::FilterThread**< **T, V** >, **task::priorityinheritance::test::HighPrioThread**, **task::priorityinheritance::test::LowPrioThread**, **task::priorityinheritance::test::NormalThread**, **task::scheduler::IdleThread**, **task::semaphore::test::ThreadASemaphoreTestCase**, **task::spinlock::test::SpinLockTestThread**, **task::UserModeThread**, and **tool::Terminal**.

Data Structures

- struct **Resource**

  Associates acquired Semaphores with Threads waiting for them.

Public Member Functions

- virtual ∼**Thread** ()

  *Destructor for class **Thread** (p. 865).*

- bool **startRunning** ()

  *Call this method to start a new **Thread** (p. 865).*

- **Process** & **getProcess** () const

  *Returns a reference to the **Process** (p. 737) containing this **Thread** (p. 865).*

Protected Member Functions

- **Thread** (**Process** &parent, ThreadPriority **basePriority**=PRIO_NORMAL, bool **userModeThread**=false)

  *Initializes a **Thread** (p. 865) object by creating a thread stack and a task.*

- virtual void **run** ()=0

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

- **ThreadStack** & **getRing3Stack** ()

  Returns the ring 3 stack.

Private Types

- typedef
  tool::collection::LinkedList
  < **lock::Semaphore** ∗ > **Semaphores**

  Type of a list of Semaphores.

- typedef
  tool::collection::LinkedList
  < **Resource** > **Resources**

  Type of a list of Resources this thread has locked.

Private Member Functions

- ThreadState **getState** () const

  *Returns the state of this **Thread** (p. 865).*

- void **setState** (ThreadState newState)

  *Sets the state of this **Thread** (p. 865).*

- ThreadPriority **getPriority** () const

  *Returns the current priority of this **Thread** (p. 865).*

- void **computeDynamicPriority** () const

  *(Re)computes the dynamic priority of this **Thread** (p. 865) in case there are other Threads blocked upon Semaphores we have acquired.*

- **lock::Semaphore** ∗ **getSemaphoreBlockedUpon** () const

*Returns the Semaphore this **Thread** (p. 865) is currently waiting for or NULL if this **Thread** (p. 865) is not blocked.*

- void **setSemaphoreBlockedUpon** (**lock::Semaphore** *semaphore)

  *Sets the Semaphore this **Thread** (p. 865) is currently waiting for.*

- bool **hasSemaphoresUnlocked** () const

  Returns true if there are Semaphores this thread has unlocked since the last task switch.

- **Semaphores** & **getSemaphoresUnlocked** ()

  Returns all Semaphores this thread has unlocked since the last task switch.

- void **addSemaphoreUnlocked** (**lock::Semaphore** &semaphore)

  *Adds the Semaphore to the list of Semaphores this **Thread** (p. 865) has just unlocked.*

- void **addResource** (**lock::Semaphore** &semaphore)

  *Called by Semaphore::down() on the current thread when this **Thread** (p. 865) acquires a Semaphore.*

- void **removeResource** (**lock::Semaphore** &semaphore, bool destroyed)

  *Called by Semaphore::up() on the current thread when this **Thread** (p. 865) releases a Semaphore.*

- void **addWaiter** (**lock::Semaphore** &semaphore, **Thread** &waiter)

  *Called by the Scheduler when another thread is blocked upon a Semaphore this **Thread** (p. 865) has acquired.*

- void **removeAllWaiters** (**lock::Semaphore** &semaphore)

  *Called by the Scheduler when this **Thread** (p. 865) releases a Semaphore that other threads are waiting for.*

- void **releaseAllResources** ()

  *Unregisters this **Thread** (p. 865) from all Semaphores it has acquired without a corresponding release.*

- **memory::Selector getTssSelector** () const

  Returns the selector of the associated TSS.

- void **initStack** ()

  Initializes the thread stack(s)..

- void **initTSS** ()

  Initializes the TSS. Requires a properly initialized thread stack.

- void * **getPointerToTCB** () const

  Returns a pointer to the thread control block as required by the ELF TLS ABI.

- bool **initTCBAndTLS** ()

  Initializes the TCB and the TLS area for this thread.

Private Attributes

- bool **init**

  *True if **Thread** (p. 865) has been initialized successfully.*

- bool **userModeThread**

  True if this is a user-mode thread.

- ThreadState **state**

  *The state of this **Thread** (p. 865).*

- ThreadPriority **basePriority**

  *The base priority of this **Thread** (p. 865). It is unchangeable.*

- ThreadPriority **currentPriority**

  *The current priority of this **Thread** (p. 865), which may be higher than its base priority due to priority boosts.*

- bool **currentPriorityMustBeRecomputed**

  True if currentPriority is invalid and has to be recomputed.

- **Process** & **process**

  *A reference to the **Process** (p. 737) this **Thread** (p. 865) belongs to.*

- **ThreadStack myStack**

  *The **ThreadStack** (p. 879).*

- **ThreadStack** * **ring0Stack**

  *The ring 0 **ThreadStack** (p. 879) (only necessary user-mode threads).*

- **TaskStateSegment tss**

  *The **TaskStateSegment** (p. 846) of this **Thread** (p. 865).*

- **memory::Selector tssSelector**

  *The segment selector for the TSS (Task State Segment) of this **Thread** (p. 865).*

- **lock::Semaphore** * **semaphoreBlockedUpon**

  The semaphore this thread is waiting for.

- **Semaphores semaphoresUnlocked**

  The semaphores this thread has just unlocked recently.

- **Resources resourcesLocked**

  The resources this thread has locked.

- uint32_t **numWaiters**

  Number of resources other threads are blocked upon.

- **lock::SpinLock resourceMutex**

  Spinlock protecting the resource list.

- **memory::lmm::AddressSpaceRange tcbRange**

  Address space range allocated to hold the TCB.

- void ∗ **tcb**

  Pointer to the TCB (thread control block).

Friends

- class **::task::scheduler::Scheduler**
- class **::task::scheduler::SchedulerHandler**
- class **::task::lock::Semaphore**

Detailed Description

Thread (p. 865) is an abstract class.

This class covers all necessary steps for making a thread manageable. The real code, a **Thread** (p. 865) should run, has to be implemented in a Class' run-method, which extends this class.

Constructor & Destructor Documentation

task::Thread::Thread (Process & *parent*, ThreadPriority *basePriority* = `PRIO_NORMAL`**, bool *userModeThread* =** `false` **)** [protected] Initializes a **Thread** (p. 865) object by creating a thread stack and a task.

Parameters

parent	The process this thread belongs to.
basePriority	The thread's base priority (defaults to PRIO_NORMAL). User-mode threads may not run at PRIO_HIGH or higher (these priority classes are reserved for kernel-mode worker threads) or at PRIO_LOWEST (this priority class is reserved for the idle thread).
userModeThread	If false (the default), true, the thread is a kernel-mode thread running at ring 0. Otherwise it is a user-mode thread running at ring 3.

Member Function Documentation

ThreadState task::Thread::getState ( ) const [inline], [private] Returns the state of this **Thread** (p. 865).

Called by the scheduler.

References state.

void task::Thread::setState (ThreadState *newState*) [inline], [private] Sets the state of this **Thread** (p. 865).

Called by the scheduler when the **Thread** (p. 865) changes its state.

Parameters

newState	The new **Thread** (p. 865) state.

References state.

void task::Thread::computeDynamicPriority ( ) const [private] (Re)computes the dynamic priority of this **Thread** (p. 865) in case there are other Threads blocked upon Semaphores we have acquired.

This is needed to avoid priority inversion problems.

Referenced by getPriority().

void task::Thread::setSemaphoreBlockedUpon (lock::Semaphore * *semaphore*) [inline], [private] Sets the Semaphore this **Thread** (p. 865) is currently waiting for.

Use NULL to indicate that this **Thread** (p. 865) is not blocked. Called by Semaphore-::down() on the current **Thread** (p. 865).

Parameters

semaphore	The Semaphore this **Thread** (p. 865) is currently waiting for.

References semaphoreBlockedUpon.

bool task::Thread::hasSemaphoresUnlocked ( ) const [inline], [private]
Returns true if there are Semaphores this thread has unlocked since the last task switch.

Called by the SchedulerHandler on the current **Thread** (p. 865) to determine if rescheduling is necessary.

References tool::collection::LinkedList< T >::isEmpty(), and semaphoresUnlocked.

Semaphores& task::Thread::getSemaphoresUnlocked ( ) [inline], [private] Returns all Semaphores this thread has unlocked since the last task switch.

Called by the Scheduler from within the Scheduler's critical section.

References semaphoresUnlocked.

void task::Thread::addSemaphoreUnlocked (lock::Semaphore & *semaphore*) [inline], [private] Adds the Semaphore to the list of Semaphores this **Thread** (p. 865) has just unlocked.

Called by Semaphore::up() on the current **Thread** (p. 865) for asymmetric Semaphores. (Symmetric Semaphores call **removeResource()** (p. 873) which then calls **addSemaphoreUnlocked()** (p. 872).)

Parameters

semaphore	The Semaphore this **Thread** (p. 865) has just unlocked.

References tool::collection::LinkedList< T >::add(), and semaphoresUnlocked.

void task::Thread::addResource (lock::Semaphore & *semaphore*) [private] Called by Semaphore::down() on the current thread when this **Thread** (p. 865) acquires a Semaphore.

Parameters

semaphore	The semaphore this **Thread** (p. 865) has acquired.

void task::Thread::removeResource (lock::Semaphore & *semaphore,* bool *destroyed*) [private] Called by Semaphore::up() on the current thread when this **Thread** (p. 865) releases a Semaphore.

If there are active waiters on passed Semaphore, the entry is not removed immediately.

Parameters

semaphore	The semaphore this **Thread** (p. 865) has released.
destroyed	If true, the Semaphore is completely unregistered instead of only decrementing the use count by one. This is used by the Semaphore's destructor.

void task::Thread::addWaiter (lock::Semaphore & *semaphore,* Thread & *waiter*) [private] Called by the Scheduler when another thread is blocked upon a Semaphore this **Thread** (p. 865) has acquired.

Parameters

semaphore	The semaphore another thread is blocked upon.
waiter	The thread blocking.

void task::Thread::removeAllWaiters (lock::Semaphore & *semaphore*) [private] Called by the Scheduler when this **Thread** (p. 865) releases a Semaphore that other threads are waiting for.

Parameters

semaphore	The semaphore this thread has released.

void task::Thread::releaseAllResources ( ) [private] Unregisters this **Thread** (p. 865) from all Semaphores it has acquired without a corresponding release.

This can happen if e.g. the thread reads from a pipe. Called by Scheduler::threadEntryPoint() after the **Thread** (p. 865) has finished its execution.

bool task::Thread::initTCBAndTLS ( ) [private] Initializes the TCB and the TLS area for this thread.

@

@plus@

@plus -@

@

@

@skip
Returns

> True if successful, else false. The operation may fail if e.g. memory for the TCB and TLS area could not be allocated.

virtual void task::Thread::run ( ) [protected], [pure virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implemented in **task::priorityinheritance::test::NormalThread** (p. 661), **io::driver::block::ata::AtaBusWorkerThread** (p. 312), **task::priorityinheritance::test::HighPrioThread** (p. 562), **task::priorityinheritance::test::LowPrioThread** (p. 642), **task::semaphore::test::ThreadASemaphoreTestCase** (p. 876), **boot::UserSpaceInitThread** (p. 897), **task::spinlock::test::SpinLockTestThread** (p. 828), **boot::BootThread** (p. 375), **ipc::test::IPCReceiverThread** (p. 576), **task::pipeandfilter::FilterThread**< **T, V** > (p. 539), **ipc::test::IPCSenderThread** (p. 577), **task::scheduler::IdleThread** (p. 563), **task::UserModeThread** (p. 895), **tool::Terminal** (p. 850), **ClockThread** (p. 388), **LockTesterThreadUp** (p. 639), and **LockTesterThreadDown** (p. 638).

ThreadStack& task::Thread::getRing3Stack ( ) [inline], [protected] Returns the ring 3 stack.

Only meaningful if thread has been created with the userModeThread flag being set.

References myStack.

bool task::Thread::startRunning ( ) Call this method to start a new **Thread** (p. 865).

After implementing the **run()** (p. 874)-method and initializing the **Thread** (p. 865) nothing else must be done except calling this method. After calling this method, the **Thread** (p. 865) will be executed when the Scheduler chooses this **Thread** (p. 865). So in fact calling this method does not result in switching to this **Thread** (p. 865) immediately. @

@plus@

@plus -@

@

@

@skip
Returns

True if thread could be started successfully, else false.

Field Documentation

Process& task::Thread::process [private] A reference to the **Process** (p. 737) this **Thread** (p. 865) belongs to.

Each **Thread** (p. 865) belongs to exactly one **Process** (p. 737).

Referenced by getProcess().

The documentation for this class was generated from the following file:

- Thread.h

A.4.310 task::semaphore::test::ThreadASemaphoreTestCase Class Reference

Inherits **task::Thread**, and **test::TestCase**.

Public Member Functions

- **ThreadASemaphoreTestCase** (**Process** &parent, char out, **lock::Semaphore** &semaphore, **lock::Semaphore** &exitSemaphore)

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

- virtual const char * **getName** ()

Private Attributes

- **lock::Semaphore** & **mySemaphore**
- **lock::Semaphore** & **myExitSemaphore**
- char **output**

Additional Inherited Members

Member Function Documentation

virtual void task::semaphore::test::ThreadASemaphoreTestCase::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- ThreadASemaphoreTestCase.h

A.4.311 task::scheduler::ThreadQueue Class Reference

Represents a queue of threads.

Public Member Functions

- **ThreadQueue** ()

  Creates an empty queue.

- bool **isEmpty** () const

  Returns true if the queue is empty.

- void **enqueueBack** (**Thread** ∗thread)

  Appends a thread to the end of the queue.

- void **enqueueFront** (**Thread** ∗thread)

  Adds a thread in front of the queue and makes it the head of the queue.

- void **remove** (**Thread** ∗thread)

  Removes a thread from the queue.

- **Thread** ∗ **dequeue** ()

  Removes the first thread in the queue and returns it.

Private Types

- typedef
  tool::collection::LinkedList
  < **Thread** ∗ > **Threads**

  Type of a list of threads.

Private Member Functions

- void **cycle** ()

  Advances the internal iterator.

Private Attributes

- **Threads threads**

  The threads in the queue.

- **Threads::Iterator it**

  The iterator pointing to the head of the queue.

Detailed Description

Represents a queue of threads.

The scheduler uses thread queues to manage threads running at different priority levels.

Note that the methods of this class are optimized for speed, so there are NO safety checks. Be careful!

Member Function Documentation

void task::scheduler::ThreadQueue::enqueueBack (Thread * *thread*) [inline] Appends a thread to the end of the queue.

This does not change the head of the queue.

Parameters

thread	The thread to add.

References tool::collection::LinkedListIterator< T >::insertBefore(), isEmpty(), it, and tool::collection::LinkedListIterator< T >::moveNext().

void task::scheduler::ThreadQueue::enqueueFront (Thread * *thread*) [inline] Adds a thread in front of the queue and makes it the head of the queue.

Parameters

thread	The thread to add.

References tool::collection::LinkedListIterator< T >::insertBefore(), and it.

void task::scheduler::ThreadQueue::remove (Thread * *thread*) Removes a thread from the queue.

Afterwards, the head of the queue is unspecified (if the queue is non-empty) or undefined (if the queue is empty).

Beware: The caller MUST ensure that the thread is already in the queue, otherwise this method will loop endlessly!

Parameters

thread	The thread to remove.

Thread* task::scheduler::ThreadQueue::dequeue ( ) [inline] Removes the first thread in the queue and returns it.

Beware: The caller MUST ensure that the queue is not empty!

References cycle(), it, and tool::collection::LinkedListIterator< T >::remove().

void task::scheduler::ThreadQueue::cycle ( ) [inline], [private] Advances the internal iterator.

If it points behind the last element afterwards, it is reset to point to the first one.

References it, and tool::collection::LinkedListIterator< T >::moveNext().

Referenced by dequeue().

The documentation for this class was generated from the following file:

- **ThreadQueue.h**

A.4.312 task::ThreadStack Class Reference

Public Member Functions

- **ThreadStack** (**Thread** &thread, bool userAccessible)

  Constructor.

- ~**ThreadStack** ()

  Destructor.

- void **push** (uint32_t element)

  Adds a new.

- uint32_t **getStackPointer** () const

  Returns the current stack pointer for the target process.

- void **removeAliasMapping** ()

  Requests to remove the kernel mapping for the stack.

Private Member Functions

- void **createAliasMapping** ()

  Creates an alias mapping for the current process.

Private Attributes

- uint32_t * **data**

- uint32_t * **stackPointer**
- uint32_t * **targetStackPointer**

Constructor & Destructor Documentation

task::ThreadStack::ThreadStack (Thread & *thread,* bool *userAccessible*) Constructor.

Parameters

thread	The **Thread** (p. 865) this stack belongs to.
user-Accessible	If true, the pages allocated for the stack are pages accessible by ring 3 code, else they are supervisor pages only accessible by ring 0 code.

Member Function Documentation

void task::ThreadStack::push (uint32_t *element*) [inline] Adds a new.

Parameters

element	onto the Stack.

uint32_t task::ThreadStack::getStackPointer ( ) const [inline] Returns the current stack pointer for the target process.

As such a pointer cannot be used in any other process, the return type is not a pointer to catch invalid access early.

@

@plus@

@plus -@

@

@

@skip
Returns

The target stack pointer or zero if initialization has failed.

void task::ThreadStack::removeAliasMapping ( ) [inline] Requests to remove the kernel mapping for the stack.

After that, the stack is only accessible by the mapping created in the target process.

Referenced by ~ThreadStack().

The documentation for this class was generated from the following file:

- ThreadStack.h

A.4.313 io::time::Time Class Reference

The <<singleton>> Class for all the time operations Saves the ticks that are collected from RTC.

Public Member Functions

- uint64_t **getTime** ()
- uint32_t **getTimeInSeconds** ()
- uint64_t **getUptime** ()

  Getter for the current uptime.
- void **init** (uint32_t **baseTime**)

  *Initializes the **Time()*** (p. *881)*
- void **tick** ()

  Handle a tick from the RTC.

Static Public Member Functions

- static **Time** * **getInstance** ()

  Singleton pattern instance returner.

Private Member Functions

- **Time** ()

  Hiden constructor for singleton pattern.

Private Attributes

- uint32_t **baseTime**

  basetime is in seconds

- uint64_t **ticks**

  ticks since start

Static Private Attributes

- static **Time** ∗ **theInstance**

  The static attribute for the singleton pattern.

Detailed Description

The <<singleton>> Class for all the time operations Saves the ticks that are collected from RTC.

Member Function Documentation

uint64_t io::time::Time::getTime ( ) @

@plus@

@plus -@

@

@

@skip
Returns

Current time in milliseconds

uint32_t io::time::Time::getTimeInSeconds ( ) @

@plus@

@plus -@

@

@

@skip
Returns

Current time in seconds

uint64_t io::time::Time::getUptime ( ) Getter for the current uptime.

@

@plus@

@plus -@

@

@

@skip
Returns

Current uptime in milliseconds

static Time* io::time::Time::getInstance ( ) [static] Singleton pattern instance returner.

@

@plus@

@plus -@

@

@

@skip
Returns

The static instance of **Time** (p. 881)

void io::time::Time::init (uint32_t *baseTime*) Initializes the **Time()** (p. 881)

Parameters

baseTime	Seconds from 1970-01-01 00:00:00

The documentation for this class was generated from the following file:

- Time.h

A.4.314 fosCli::commands::TimeCliCommand Class Reference

Inherits **fosCli::parser::CliCommand**.

Public Member Functions

- virtual void **execute** ()
- virtual void **addParameter** (char const ∗parameter)

The documentation for this class was generated from the following file:

- TimeCliCommand.h

A.4.315 fosCli::commands::TimeCliCommandCreator Class Reference

Inherits **fosCli::parser::CliCommandCreator**.

Public Member Functions

- virtual **parser::CliCommand** ∗ **create** ()

The documentation for this class was generated from the following file:

- TimeCliCommandCreator.h

A.4.316 task::scheduler::TimerInterruptHandler Class Reference

Inherits **io::driver::interrupt::IRQHandler**.

Public Member Functions

- **TimerInterruptHandler** (**Scheduler** &**scheduler**)

  Constructor.

- virtual bool **handle** ()

  Handles the interrupt.

Private Attributes

- **Scheduler** & **scheduler**

  The scheduler.

Constructor & Destructor Documentation

task::scheduler::TimerInterruptHandler::TimerInterruptHandler (Scheduler & *scheduler*) `[inline]` Constructor.

Parameters

scheduler	A reference to the scheduler.

Member Function Documentation

virtual bool task::scheduler::TimerInterruptHandler::handle ( ) `[inline]`, `[virtual]` Handles the interrupt.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the IRQHandler handled the IRQ, else false. In the latter case, the PICInterruptHandler will call the next IRQHandler in the chain (if available).

Implements **io::driver::interrupt::IRQHandler** (p. 579).

References scheduler, and task::scheduler::Scheduler::setTimeSliceExceeded().

The documentation for this class was generated from the following file:

- TimerInterruptHandler.h

A.4.317 cpu::level::TransitionHandler Class Reference

Handles the the handler for the different levels.

Inherited by **io::driver::interrupt::MaskingHandler**, and **task::scheduler::SchedulerHandler**.

Public Member Functions

- virtual ~**TransitionHandler** ()

  Destructor.

- virtual IRQLevel **lowestLevel** () const =0

  Returns the lowest IRQ level of a transition (inclusive).

- virtual IRQLevel **highestLevel** () const =0

  Returns the highest IRQ level of a transition (exclusive).

- virtual void **raise** (IRQLevel fromLevel, IRQLevel toLevel)=0

  *Is called by the **LevelManager** (p. 592) whenever the IRQ level is raised.*

- virtual void **lower** (IRQLevel fromLevel, IRQLevel toLevel)=0

  *Is called by the **LevelManager** (p. 592) whenever the IRQ level is lowered.*

Detailed Description

Handles the the handler for the different levels.

Member Function Documentation

virtual IRQLevel cpu::level::TransitionHandler::lowestLevel ( ) const `[pure virtual]` Returns the lowest IRQ level of a transition (inclusive).

The handler is only invoked if the new (while raising) or old (while lowering) IRQ level is greater than or equal to this level. @

@plus@

@plus -@

@

@

@skip
Returns

The highest IRQ level.

Implemented in **io::driver::interrupt::MaskingHandler** (p. 649), and **task::scheduler::SchedulerHandler** (p. 810).

virtual IRQLevel cpu::level::TransitionHandler::highestLevel ( ) const [pure virtual] Returns the highest IRQ level of a transition (exclusive).

The handler is only invoked if the old (while raising) or new (while lowering) IRQ level is less than this level. @

@plus@

@plus -@

@

@

@skip
Returns

The highest IRQ level.

Implemented in **io::driver::interrupt::MaskingHandler** (p. 650), and **task::scheduler::SchedulerHandler** (p. 810).

virtual void cpu::level::TransitionHandler::raise (IRQLevel *fromLevel,* IRQLevel *toLevel*) [pure virtual] Is called by the **LevelManager** (p. 592) whenever the IRQ level is raised.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implemented in **io::driver::interrupt::MaskingHandler** (p. 650), and **task::scheduler::SchedulerHandler** (p. 811).

virtual void cpu::level::TransitionHandler::lower (IRQLevel *fromLevel,* IRQLevel *toLevel*) `[pure virtual]` Is called by the **LevelManager** (p. 592) whenever the IRQ level is lowered.

Parameters

fromLevel	The current IRQ level.
toLevel	The new IRQ level.

Implemented in **io::driver::interrupt::MaskingHandler** (p. 650), and **task::scheduler::SchedulerHandler** (p. 811).

The documentation for this class was generated from the following file:

- TransitionHandler.h

A.4.318 memory::TSSDescriptor Class Reference

Describes a TSS descriptor.

Inherits **memory::SystemSegmentDescriptor**.

Public Member Functions

- uint32_t **getBase** () const

  Returns the segment's base address.

- **TSSDescriptor** & **setBase** (uint32_t base)

  Sets the segment's base address.

- uint32_t **getLimit** () const

  Returns the segment's limit.

- **TSSDescriptor** & **setLimit** (uint32_t limit)

  Sets the segment's limit.

- bool **isLimit4G** () const

  Returns the 4GiB Limit (G) flag.

- **TSSDescriptor** & **setLimit4G** (bool limit4g)

  Sets the 4GiB Limit (G) flag.

Additional Inherited Members

Detailed Description

Describes a TSS descriptor.

Member Function Documentation

TSSDescriptor& memory::TSSDescriptor::setBase (uint32_t *base*) [inline] Sets the segment's base address.

Parameters

base	The new base address.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

uint32_t memory::TSSDescriptor::getLimit ( ) const [inline] Returns the segment's limit.

The value returned is always a full 32-bit value, taking the value of the 4GiB Limit flag (G) into account.

TSSDescriptor& memory::TSSDescriptor::setLimit (uint32_t *limit*) [inline] Sets the segment's limit.

If the limit is greater than or equal to 2^20, the 4GiB Limit flag (G) is set.

Parameters

limit	The new limit.

@

@plus@

@plus -@

@

@

@skip

Returns

*this

TSSDescriptor& memory::TSSDescriptor::setLimit4G (bool *limit4g*) [inline] Sets the 4GiB Limit (G) flag.

Don't call this explicitly, let **setLimit()** (p. 889) determine the correct value for this flag.

Parameters

limit4g	True if the limit is shifted 12 bit to the left, false if no shifting should be done.

@

@plus@

@plus -@

@

@

@skip

Returns

The documentation for this class was generated from the following file:

- Descriptor.h

A.4.319 TypeInfo< int16_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = false
- static const unsigned **NumBits** = 16
- static const unsigned **IndexBits** = 4

The documentation for this struct was generated from the following file:

- **types.h**

A.4.320 TypeInfo< int32_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = false
- static const unsigned **NumBits** = 32
- static const unsigned **IndexBits** = 5

The documentation for this struct was generated from the following file:

- **types.h**

A.4.321 TypeInfo< int64_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = false
- static const unsigned **NumBits** = 64
- static const unsigned **IndexBits** = 6

The documentation for this struct was generated from the following file:

- **types.h**

A.4.322 TypeInfo< int8_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = false

- static const unsigned **NumBits** = 8
- static const unsigned **IndexBits** = 3

The documentation for this struct was generated from the following file:

- **types.h**

A.4.323 TypeInfo< uint16_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = true
- static const unsigned **NumBits** = 16
- static const unsigned **IndexBits** = 4

The documentation for this struct was generated from the following file:

- **types.h**

A.4.324 TypeInfo< uint32_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = true
- static const unsigned **NumBits** = 32
- static const unsigned **IndexBits** = 5

The documentation for this struct was generated from the following file:

- **types.h**

A.4.325 TypeInfo< uint64_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = true
- static const unsigned **NumBits** = 64
- static const unsigned **IndexBits** = 6

The documentation for this struct was generated from the following file:

- **types.h**

A.4.326 TypeInfo< uint8_t > Struct Template Reference

Static Public Attributes

- static const bool **IsUnsigned** = true
- static const unsigned **NumBits** = 8
- static const unsigned **IndexBits** = 3

The documentation for this struct was generated from the following file:

- **types.h**

A.4.327 lib::runtime::UserEnvironment Class Reference

Class **UserEnvironment** (p. 893) is the implementation of **memory::allocator::Environment** (p. 481) for the user space.

Inherits **memory::allocator::Environment**.

Public Member Functions

- virtual void **enterCriticalSection** ()

  Enters the Allocator's critical section.

- virtual void **leaveCriticalSection** ()

  Leaves the Allocator's critical section.

- virtual char * **allocateBlock** (uint32_t pages)

  Allocates a memory block.

- virtual bool **freeBlock** (void *ptr, uint32_t pages)

  Frees a previously allocated memory block.

Private Attributes

- **lib::Semaphore sema**

  Semaphore *(p. 819) used for locking.*

Detailed Description

Class **UserEnvironment** (p. 893) is the implementation of **memory::allocator::Environment** (p. 481) for the user space.

Member Function Documentation

virtual char* lib::runtime::UserEnvironment::allocateBlock (uint32_t *pages*) [virtual] Allocates a memory block.

Parameters

pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the allocated block or NULL if allocation failed.

Implements **memory::allocator::Environment** (p. 482).

virtual bool lib::runtime::UserEnvironment::freeBlock (void * *ptr*, uint32_t *pages*) [virtual] Frees a previously allocated memory block.

Parameters

ptr	The pointer to the previously allocated block.
pages	The size of the block in pages.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the block could be freed, false otherwise.

Implements **memory::allocator::Environment** (p. 483).

The documentation for this class was generated from the following file:

- UserEnvironment.h

A.4.328 task::UserModeThread Class Reference

Inherits **task::Thread**.

Public Types

- typedef void(∗ **UserModeEntryPoint**)(**ipc::CommandRelay** &relay, uint32_t **imageSize**)

  User mode thread entry point.

Public Member Functions

- **UserModeThread** (**task::Process** &parent, uint32_t **entryPoint**, uint32_t **imageSize**, ThreadPriority priority=PRIO_NORMAL)
- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- uint32_t **entryPoint**

  The address of the user-mode thread entry point.
- uint32_t **imageSize**

  The size in bytes of the image.

Additional Inherited Members

Member Function Documentation

virtual void task::UserModeThread::run ( ) [virtual] IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete **Thread** (p. 865). Place any Operation the **Thread** (p. 865) shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **UserModeThread.h**

A.4.329 boot::UserSpaceInitThread Class Reference

This thread is started at the end of the boot process.

Inherits **task::Thread**.

Public Member Functions

- **UserSpaceInitThread** (char ∗**addrAppArea**, uint32_t **sizeAppArea**)

  Constructor.

Protected Member Functions

- virtual void **run** ()

  IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

Private Attributes

- char ∗ **addrAppArea**

  The start address of the application area.

- uint32_t **sizeAppArea**

  The size of the application area in bytes.

Detailed Description

This thread is started at the end of the boot process.

It is responsible for bootstrapping the user space.

Well, for the time being, there is no much user space to be initialized or started, so simply put your kernel test code into **UserSpaceInitThread::run()** (p. 897).

Constructor & Destructor Documentation

boot::UserSpaceInitThread::UserSpaceInitThread (char * *addrAppArea,* uint32_t *sizeAppArea*) Constructor.

Parameters

addrAppArea	The start address of the application area.
sizeAppArea	The size of the application area in bytes.

Member Function Documentation

virtual void boot::UserSpaceInitThread::run ( ) [protected], [virtual]
IMPORTANT! Do not call this method directly! Use "startRunning()" instead for executing a concrete Thread!

This is a pure-virtual operation which has to be implemented by a concrete Thread. Place any Operation the Thread shall execute in this method!

Implements **task::Thread** (p. 874).

The documentation for this class was generated from the following file:

- **UserSpaceInitThread.h**

A.4.330 io::vfs::VFSManager Class Reference

Singleton manager for the virtual file system.

Public Member Functions

- virtual **object::Ref**
  < **AbstractMount** > **mount** (**FileSystem** *fs, **Volume** *v)

  Tries to mount a given volume with the given file system.

- virtual UmountError::enum_t **umount** (uint8_t mountID)

  Tries to umount the mount with the specified ID.

- virtual **object::Ref**
  < **AbstractMount** > **getMount** (uint8_t mountID)

  Returns a reference to a mount with the specified mount ID.

- virtual **tool::collection::List**
  < **object::Ref**< **AbstractMount** > > * **getListOfMounts** ()

  Returns a list of all currently mounted mounts.

Static Public Member Functions

- static **VFSManager** & **instance** ()

  Returns the only instance of the VFS manager.

Private Attributes

- **object::Ref**< **Mount** > * **myMounts**

  Internal array of mounts.

Detailed Description

Singleton manager for the virtual file system.

- mount and umount volumes
- list all mounted volumes

Member Function Documentation

static VFSManager& io::vfs::VFSManager::instance ( ) [static] Returns the only instance of the VFS manager.

@

@plus@

@plus -@

@

@

@skip
Returns

the only instance of type **VFSManager** (p. 897)

virtual object::Ref<AbstractMount> io::vfs::VFSManager::mount (FileSystem * *fs*, Volume * *v*) [virtual] Tries to mount a given volume with the given file system.

Parameters

fs	The file system to use
v	The volume to mount

@

@plus@

@plus -@

@

@

@skip
Returns

A reference to the created mount, if the mount was possible. Otherwise, null is returned.

virtual UmountError::enum_t io::vfs::VFSManager::umount (uint8_t *mountID*) [virtual] Tries to umount the mount with the specified ID.

Parameters

mountID	The ID of the mount, that should be umounted

@

@plus@

@plus -@

@

@

@skip
Returns

SUCCESS if the volume is umounted INUSE if the volume is still in use - no umount is performed NOTMOUNTED if the volume was umounted before or wasn't mounted at all

virtual object::Ref<AbstractMount> io::vfs::VFSManager::getMount (uint8_t *mountID*) [virtual] Returns a reference to a mount with the specified mount ID.

Parameters

mountID	The ID of the mount, that should be returned

@

@plus@

@plus -@

@

@

@skip
Returns

The requested mount, or null, if no mount is available with the given ID

virtual tool::collection::List<object::Ref<AbstractMount> >* io::vfs::VFSManager::getListOfMounts ( ) [virtual] Returns a list of all currently mounted mounts.

@

@plus@

@plus -@

@

@

@skip

Returns

A list of mounts

The documentation for this class was generated from the following file:

- VFSManager.h

A.4.331 io::vfs::test::VFSTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- virtual void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- VFSTestCase.h

A.4.332 io::driver::graphics::vga::VgaChar Class Reference

Represents a single element in the video buffer.

Public Member Functions

- void **setChar** (char newC)

  sets the character to display
- void **setTextColor** (io::console::Color color)

  sets the text color of the character to display
- void **setBackgroundColor** (io::console::Color color)

  sets the text background color of the character to display
- char **getChar** () const

getter for the character to display

- io::console::Color **getTextColor** () const

  getter for the text color

- io::console::Color **getBackgroundColor** () const

  getter for the text background color

- **TextChar toTextChar** () const

  returns as one text char

Private Attributes

- char **sign**

  The character.

- unsigned char **attr**

  Character attributes.

Detailed Description

Represents a single element in the video buffer.

Member Function Documentation

void io::driver::graphics::vga::VgaChar::setChar (char *newC*) [inline]
sets the character to display

Parameters

newC	the character to display

References sign.

void io::driver::graphics::vga::VgaChar::setTextColor (io::console::Color *color*) [inline] sets the text color of the character to display

Parameters

color	color to set

References attr.

void io::driver::graphics::vga::VgaChar::setBackgroundColor (io::console::Color *color*) `[inline]` sets the text background color of the character to display

Parameters

color	color to set

References attr.

The documentation for this class was generated from the following file:

- VgaChar.h

A.4.333 io::driver::graphics::vga::test::VgaCharTestCase Class Reference

Inherits **test::TestCase**.

Public Member Functions

- void **run** ()
- virtual const char ∗ **getName** ()

Additional Inherited Members

The documentation for this class was generated from the following file:

- VgaCharTestCase.h

A.4.334 io::driver::graphics::vga::VgaCursor Class Reference

Implements the **Cursor** (p. 423) interface for the VGA device.

Inherits **io::driver::graphics::Cursor**.

Public Member Functions

- **VgaCursor** (**VgaTextDevice** &**device**)

  Constructor.

- virtual **Position getPosition** () const

  Getter for the current position of the cursor.

- virtual void **moveTo** (**Position pos**)

  Moves the cursor to <pos>

Private Attributes

- **VgaTextDevice** & **device**

  *Buffer for the VGA **Device** (p. 436).*

- **Position pos**

  My current position.

Detailed Description

Implements the **Cursor** (p. 423) interface for the VGA device.

Constructor & Destructor Documentation

io::driver::graphics::vga::VgaCursor::VgaCursor (VgaTextDevice & *device*) [inline] Constructor.

Parameters

device	The VGA **Device** (p. 436) where the cursor belongs to

Member Function Documentation

virtual Position io::driver::graphics::vga::VgaCursor::getPosition ( ) const [inline], [virtual] Getter for the current position of the cursor.

@

@plus@

@plus -@

@

@

@skip
Returns

The current **Position** (p. 727) of the cursor

Implements **io::driver::graphics::Cursor** (p. 423).

References pos.

virtual void io::driver::graphics::vga::VgaCursor::moveTo (Position *pos*) `[virtual]` Moves the cursor to <pos>

Parameters

pos	**Position** (p. 727) to move the cursor to

Implements **io::driver::graphics::Cursor** (p. 423).

The documentation for this class was generated from the following file:

- **VgaCursor.h**

A.4.335 io::driver::graphics::vga::VgaDriver Class Reference

Driver (p. 455) for the VGA **Device** (p. 436).

Inherits **io::driver::Driver**.

Public Member Functions

- virtual void **checkDev** ()

  Checks if a new device for this driver is available and initiates a corresponding device-object.

Static Private Attributes

- static uint32_t const **PhysicalStartAddressOfVideoRAM** = 0xB8000

  Physical start address of the Video RAM (no plug-and-play yet).

- static uint32_t const **NumCols** = 80

  Number of columns.

- static uint32_t const **NumLines** = 25

  Number of lines.

Additional Inherited Members

Detailed Description

Driver (p. 455) for the VGA **Device** (p. 436).

The documentation for this class was generated from the following file:

- VgaDriver.h

A.4.336 io::driver::graphics::vga::VgaTextDevice Class Reference

Console that using the VGA **Device** (p. 436).

Inherits **io::driver::graphics::TextDevice**.

Public Member Functions

- **VgaTextDevice** (uint32_t physStartVideoRam, uint32_t numCols, uint32_t numLines)

  Constructor.

- bool **isValid** () const

  Returns true if device initialization has succeeded.

- virtual void **writeAt** (**Position** pos, **TextChar** ch)

  Writes the.

- virtual **VgaCursor** ∗ **getCursor** ()

  Returns the associated cursor.

- virtual **TextChar getCharForPosition** (**Position** pos)
- virtual void **activate** ()

  Activates the device.

- virtual void **deactivate** ()

  Deactivates the device.

Private Member Functions

- void **moveCursor** (**Position** pos)

Static Private Member Functions

- static **Position getCursorPositionFromCRT** ()

Private Attributes

- **task::lock::SpinLock mutex**

  Spin lock protecting concurrent access to the CRT controller.
- **VgaCursor cursor**
- **VgaChar** ∗ **ramStartPosition**
- **VgaChar** ∗ **ramEndPosition**

Static Private Attributes

- static uint16_t const **CRTAddressPort** = 0x3d4

  CRT address port.
- static uint16_t const **CRTDataPort** = **CRTAddressPort** + 1

  CRT data port.
- static uint16_t const **CRTCursorHigh** = 0xE

  *CRT **Cursor** (p. 423) High.*
- static uint16_t const **CRTCursorLow** = **CRTCursorHigh** + 1

  *CRT **Cursor** (p. 423) Low.*

Friends

- class **VgaCursor**

Detailed Description

Console that using the VGA **Device** (p. 436).

Constructor & Destructor Documentation

io::driver::graphics::vga::VgaTextDevice::VgaTextDevice (uint32_t *physStartVideoRam,* uint32_t *numCols,* uint32_t *numLines*) Constructor.

Parameters

physStartVideoRam	Video RAM begin in the linear address space
numCols	Number of columns to display
numLines	Number of lines to display

Member Function Documentation

virtual void io::driver::graphics::vga::VgaTextDevice::writeAt (Position *pos,* TextChar *ch*) `[virtual]` Writes the.

Parameters

ch	to position (
pos).	

Implements **io::driver::graphics::TextDevice** (p. 864).

virtual TextChar io::driver::graphics::vga::VgaTextDevice::getCharForPosition (Position *pos*) `[virtual]` @

@plus@

@plus -@

@

@

@skip
Returns

the **TextChar** (p. 857) for the position (

Parameters

pos)	

Implements **io::driver::graphics::TextDevice** (p. 864).

The documentation for this class was generated from the following file:

- VgaTextDevice.h

A.4.337 io::vfs::Volume Class Reference

Inherited by **io::vfs::BlockDeviceVolume**.

Public Member Functions

- virtual bool **isReadOnly** ()=0

  Indicates if this device is a read only device.

- virtual **io::driver::block::Block** ∗ **read** (**io::driver::block::LBAddress** address)=0

  Reads the block at the.

- virtual void **write** (**io::driver::block::Block** ∗block, **io::driver::block::-LBAddress** address)=0

  Writes the.

- virtual **io::driver::block::Block** ∗ **reads** (**io::driver::block::LBAddress** address, uint16_t numberOfBlocksToRead)=0

  Reads the blocks at the.

- virtual void **writes** (**io::driver::block::Block** ∗blocks, **io::driver::block-::LBAddress** address, uint16_t numberOfBlocksToWrite)=0

  Writes the.

- virtual uint16_t **maximalNumberOfBlocksToReadOrWriteAtOnce** ()=0
- virtual **io::driver::block::LBAddress getMinAddress** ()=0
- virtual **io::driver::block::LBAddress getMaxAddress** ()=0

Member Function Documentation

virtual bool io::vfs::Volume::isReadOnly ( ) [pure virtual] Indicates if this device is a read only device.

@

@plus@

@plus -@

@

@

@skip
Returns

true -> read-only false -> write and readable

Implemented in **io::vfs::BlockDeviceVolume** (p. 356).

virtual io::driver::block::Block* io::vfs::Volume::read (io::driver::block::LBAddress *address*) [pure virtual] Reads the block at the.

Parameters

address	and returns it.

@

@plus@

@plus -@

@

@

@skip
Returns

the block which are stored at the

Parameters

address	

Implemented in **io::vfs::BlockDeviceVolume** (p. 356).

virtual void io::vfs::Volume::write (io::driver::block::Block * *block,* io::driver::block::LBAddress *address*) [pure virtual] Writes the.

Parameters

block	to the device at the given
address	

Implemented in **io::vfs::BlockDeviceVolume** (p. 357).

virtual io::driver::block::Block∗ io::vfs::Volume::reads (io::driver::block::LBAddress *address,* uint16_t *numberOfBlocksToRead*) [pure virtual]
Reads the blocks at the.

Parameters

address	and following and returns them.

@

@plus@

@plus -@

@

@

@skip
Returns

the block which are stored at the

Parameters

address	

Implemented in **io::vfs::BlockDeviceVolume** (p. 357).

virtual void io::vfs::Volume::writes (io::driver::block::Block ∗ *blocks,* io::driver::block::LBAddress *address,* uint16_t *numberOfBlocksToWrite*) [pure virtual] Writes the.

Parameters

blocks	to the device at the given
address	and the following

Implemented in **io::vfs::BlockDeviceVolume** (p. 358).

virtual uint16_t io::vfs::Volume::maximalNumberOfBlocksToReadOrWriteAtOnce ( ) [pure virtual] @

@plus@

@plus -@

@

@

@skip
Returns

the maximal number of blocks which can be read or written by the Device at once.

Implemented in **io::vfs::BlockDeviceVolume** (p. 358).

virtual io::driver::block::LBAddress io::vfs::Volume::getMinAddress ( ) `[pure virtual]` @

@plus@

@plus -@

@

@

@skip
Returns

the minimal LBAddress of the address range of this device

Implemented in **io::vfs::BlockDeviceVolume** (p. 358).

virtual io::driver::block::LBAddress io::vfs::Volume::getMaxAddress ( ) `[pure virtual]` @

@plus@

@plus -@

@

@

@skip
Returns

the maximal LBAddress of the address range of this device

Implemented in **io::vfs::BlockDeviceVolume** (p. 359).

The documentation for this class was generated from the following file:

- Volume.h

A.4.338 io::time::WaitingQueue Class Reference

Priority Queue for the Threads waiting to wake up.

Public Member Functions

- virtual ~**WaitingQueue** ()

  *Destructor for **WaitingQueue** (p. 914).*
- void **waitUntil** (uint64_t wakeTime)

  Make the current thread wait until the specified time.
- void **waitFor** (uint64_t timeInMs)

  wait for timeInMs
- void **waitToNextSecond** ()

  wait till next second is reached
- void **wakeUp** ()

  Wake up all threads with (wakeTime <= currentTime)

Static Public Member Functions

- static **WaitingQueue** * **instance** ()

  Getter for the singleton instance.

Data Fields

- **Time** * **time**

Private Attributes

- **tool::collection::LinkedList**
  < **WaitingQueueEntry** * > **waitingList**

- **task::lock::Semaphore sem**

Static Private Attributes

- static **WaitingQueue** ∗ **theInstance**

Detailed Description

Priority Queue for the Threads waiting to wake up.

Constructor & Destructor Documentation

virtual io::time::WaitingQueue::∼WaitingQueue ( ) [inline], [virtual]
Destructor for **WaitingQueue** (p. 914).

This will automatically wake up any threads remaining in the queue.

References tool::collection::LinkedListIterator< T >::moveNext(), tool::collection-::LinkedListIterator< T >::remove(), and io::time::WaitingQueueEntry::wakeUp().

Member Function Documentation

static WaitingQueue∗ io::time::WaitingQueue::instance ( ) [static]
Getter for the singleton instance.

void io::time::WaitingQueue::waitUntil (uint64_t *wakeTime*) Make the current thread wait until the specified time.

The documentation for this class was generated from the following file:

- WaitingQueue.h

A.4.339 io::time::WaitingQueueEntry Class Reference

Public Member Functions

- **WaitingQueueEntry** (uint64_t pWakeTime)
- virtual ∼**WaitingQueueEntry** ()

  *Destructor for **WaitingQueueEntry** (p. 915).*

- void **sleep** ()

  Go to sleep.

- void **wakeUp** ()

  Wake the thread now.

- uint64_t **getWakeTime** ()

  The absolute point in time to wake the sleeping thread.

Private Attributes

- uint64_t **wakeTime**
- **task::lock::Semaphore sem**

Constructor & Destructor Documentation

virtual io::time::WaitingQueueEntry::~WaitingQueueEntry ( ) [inline], [virtual] Destructor for **WaitingQueueEntry** (p. 915).

This will NOT wake up the sleeping thread, so make sure to call **wakeUp()** (p. 916) before deleting this entry!

Member Function Documentation

void io::time::WaitingQueueEntry::sleep ( ) Go to sleep.

void io::time::WaitingQueueEntry::wakeUp ( ) Wake the thread now.

Referenced by io::time::WaitingQueue::~WaitingQueue().

uint64_t io::time::WaitingQueueEntry::getWakeTime ( ) [inline] The absolute point in time to wake the sleeping thread.

The documentation for this class was generated from the following file:

- WaitingQueueEntry.h

A.5 File Documentation

A.5.1 a20.h File Reference

Contains functions related to the A20 line.

Functions

- void **enableA20Line** ()

  Enables the A20 line.

Detailed Description

Contains functions related to the A20 line.

A.5.2 AddressSpaceRange.h File Reference

Data Structures

- class **memory::lmm::AddressSpaceRange**

  *Represents a range of linear address space managed by a **LinearAddressSpaceManager*** (p. 597).

Enumerations

- enum **Type** : uint32_t {
  SystemSegment16Bit = 0, **SystemSegment32Bit** = 1, **DataSegment** = 2, **CodeSegment** = 3,
  INVALID = 0, **FREE**, **ALLOCATED**, **RESERVED**,
  Free = 1, **ReservedBySystem**, **ReservedByACPIReclaimable**, **ReservedByACPI_NVS** }

  Enumeration representing all possible types of reservations which can be acquired from the LinearAddressSpaceManager.

Functions

- class **memory::lmm::AddressSpaceRange** **memory::lmm::__attribute__** ((packed))
- bool **isValid** () const
- void **setInvalid** ()

  Marks this AddressSpaceRange as INVALID.
- uint32_t **getStartAddress** () const
- uint32_t **getEndAddress** () const
- uint32_t **getSize** () const
- uint32_t **getSizeInPages** () const
- **Type getType** () const
- char ∗ **getStartPointer** () const
- char ∗ **getEndPointer** () const
- bool **contains** (void const ∗address) const

  Determines if this range contains some address.
- bool **contains** (AddressSpaceRange const &range) const

  Determines if this range contains another one.
- uint32_t **getOffset** (void const ∗address) const

  Returns the offset within this range for some address.
- void ∗ **getPointer** (uint32_t offset) const

  Maps an offset relative to the start of this range to an absolute pointer.
- bool **map** (paging::PageDirectory &pageDir, paging::PageFlags const &flags=paging::PageFlags()) const

  Maps the range somewhere into the physical address space.
- bool **map** (paging::PageDirectory &pageDir, void ∗physAddr, paging::PageFlags const &flags=paging::PageFlags()) const

  Maps the range into a certain region on physical address space.
- bool **map** (paging::PageDirectory &pageDir, AddressSpaceRange const &otherRange, paging::PageDirectory &otherPageDir) const

Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

- bool **map** (paging::PageDirectory &pageDir, AddressSpaceRange const &otherRange, paging::PageDirectory &otherPageDir, paging::PageFlags const &flags) const

  Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

- void **unmap** (paging::PageDirectory &pageDir) const

  Unmaps the range from the physical address space.

- AddressSpaceRange **createAlias** (LinearAddressSpaceManager &dest, LinearAddressSpaceManager &src)

  Creates an alias mapping.

- AddressSpaceRange **createAlias** (LinearAddressSpaceManager &dest, LinearAddressSpaceManager &src, paging::PageFlags flags)

  Creates an alias mapping.

- void ∗ **transform** (void const ∗srcPointer, AddressSpaceRange const &targetRange) const

  Transforms an address pointing into this AddressSpaceRange into an address pointing into an alias range.

- void **setStartAddress** (uint32_t **startAddress**)

  Sets the start address.

- void **setEndAddress** (uint32_t **endAddress**)

  Sets the end address.

- void **setType** (**Type type**)

  Sets the range type.

- AddressSpaceRange ∗ **getPrevious** () const
- void **setPrevious** (AddressSpaceRange ∗**previous**)

  Sets the previous AddressSpaceRange.

- AddressSpaceRange ∗ **getNext** () const
- void **setNext** (AddressSpaceRange ∗**next**)

  Sets the next AddressSpaceRange.

- bool **doMap** (paging::PageDirectory &pageDir, AddressSpaceRange const &otherRange, paging::PageDirectory &otherPageDir, paging::PageFlags const *flags) const

  Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

- bool **memory::lmm::operator==** (AddressSpaceRange const &lhs, AddressSpaceRange const &rhs)

  Returns true if both ranges are equal, false otherwise.

- static AddressSpaceRange **create** (uint32_t **startAddress**, uint32_t **endAddress**)

  Creates an address range.

- static AddressSpaceRange **create** (void const ***startAddress**, void const ***endAddress**)
- static AddressSpaceRange **create** (void const ***startAddress**, uint32_t sizeInBytes)

Variables

- static uint32_t const **PageOffsetMask** = (1 << memory::PAGE_OFFSET_LEN) - 1

  Mask for the lower PAGE_OFFSET_LEN bits.

- uint32_t **startAddress**

  The start address of this range.

- uint32_t **endAddress**

  The first address behind this range.

- **Type type**

  The type of this range.

- AddressSpaceRange * **previous**

  Points to the AddressSpaceRange ending at our start address.

- AddressSpaceRange * **next**

  Points to the AddressSpaceRange starting at our end address.

Enumeration Type Documentation

enum Type : uint32_t Enumeration representing all possible types of reservations which can be acquired from the LinearAddressSpaceManager.

Enumerator

INVALID The range is invalid.

FREE The range describes a memory range that is unused.

ALLOCATED The range describes a memory range that is in use.

RESERVED The range describes a memory range that is in use by the LinearAddressSpace Manager.

Function Documentation

static AddressSpaceRange __attribute__::create (uint32_t *startAddress*, uint32_t *endAddress*) [static] Creates an address range.

Parameters

start-Address	The start address. It is aligned on the nearest page boundary less than or equal to the address.
endAddress	The end address, i.e. the first address behind the range. If is aligned on the nearest page boundary greater than or equal to the address.

@

@plus@

@plus -@

@

@

@skip

Returns

The AddressSpaceRange object. Its type is set to ALLOCATED.

bool __attribute__::isValid ( ) const Returns true if this range is valid.

uint32_t __attribute__::getStartAddress ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

The start address of this range.

uint32_t __attribute__::getEndAddress ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

The first address behind this range.

uint32_t __attribute__::getSize ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

The size of the range in bytes.

uint32_t __attribute__::getSizeInPages ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

The size of the range in pages.

Type __attribute__::getType ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

The type of this range.

char* __attribute__::getStartPointer ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the memory region described.

char* __attribute__::getEndPointer ( ) const @

@plus@

@plus -@

@

@

@skip
Returns

A pointer to the following memory region.

bool __attribute__::contains (void const * *address*) const Determines if this range contains some address.

Parameters

address	The address to check.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this range contains the address passed, else false.

bool __attribute__::contains (AddressSpaceRange const & *range*) const Determines if this range contains another one.

Parameters

range	The other range to check.

@

@plus@

@plus -@

@

@

@skip

Returns

True if this range contains the range passed, else false.

uint32_t __attribute__::getOffset (void const * *address*) const Returns the offset within this range for some address.

Parameters

address	The address.

@

@plus@

@plus -@

@

@

@skip

Returns

The number of bytes from the start of this range to the address passed. Only meaningful if this->contains(address) returns true.

void* __attribute__::getPointer (uint32_t *offset*) const Maps an offset relative to the start of this range to an absolute pointer.

Parameters

offset	The offset to add to the start address of this range.

@

@plus@

@plus -@

@

@

@skip

Returns

The resulting pointer. Only meaningful if the offset is less than the size of this range.

bool __attribute__::map (paging::PageDirectory & *pageDir,* paging::PageFlags const & *flags* = `paging::PageFlags()` **) const** Maps the range somewhere into the physical address space.

Parameters

pageDir	The page directory to use.
flags	The page flags to use.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

bool __attribute__::map (paging::PageDirectory & *pageDir,* void * *physAddr,* paging::PageFlags const & *flags* = `paging::PageFlags()`) const Maps the range into a certain region on physical address space.

Parameters

pageDir	The page directory to use.
physAddr	The physical start address of the memory to be used for the mapping.
flags	The page flags to use.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

bool __attribute__::map (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir*) const Maps the range into a certain region on physical address space,

described by another address space range, thereby performing a copy of an address space mapping.

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPage-Dir	The page directory the other address space range belongs to.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

bool __attribute__::map (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir,* paging::PageFlags const & *flags*) const Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

The page flags can be overridden for the copy (such that e.g. a read-only ring 3 alias can be created for a read/write ring 0 buffer).

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPage-Dir	The page directory the other address space range belongs to.
flags	The page flags to use for the mapping.

@

@plus@

@plus -@

@

@

@skip
Returns

True if the whole address range could be mapped, else false.

void __attribute__::unmap (paging::PageDirectory & *pageDir*) const
Unmaps the range from the physical address space.

Parameters

pageDir	The page directory to use.

AddressSpaceRange __attribute__::createAlias (LinearAddressSpaceManager & *dest,* LinearAddressSpaceManager & *src*) Creates an alias mapping.

Parameters

dest	The LinearAddressSpaceManager to use in order to create the alias mapping.
src	The LinearAddressSpaceManager this AddressSpaceRange belongs to.

@

@plus@

@plus -@

@

@

@skip
Returns

The new AddressSpaceRange living in the memory space of the destination LASM, mapped to the same memory this AddressSpaceRange living in the memory space of the source LASM. If the mapping failed, the range returned is of type INVALID.

AddressSpaceRange __attribute__::createAlias (LinearAddressSpaceManager & *dest,* LinearAddressSpaceManager & *src,* paging::PageFlags *flags*) Creates an alias mapping.

Parameters

dest	The LinearAddressSpaceManager to use in order to create the alias mapping.
src	The LinearAddressSpaceManager this AddressSpaceRange belongs to.
flags	The flags for the alias mapping.

@

@plus@

@plus -@

@

@

@skip

Returns

The new AddressSpaceRange living in the memory space of the destination LASM, mapped to the same memory this AddressSpaceRange living in the memory space of the source LASM. If the mapping failed, the range returned is of type INVALID.

void∗ __attribute__::transform (void const ∗ *srcPointer,* AddressSpaceRange const & *targetRange*) const Transforms an address pointing into this AddressSpaceRange into an address pointing into an alias range.

Parameters

srcPointer	The pointer within this AddressSpaceRange.
targetRange	The alias range.

@

@plus@

@plus -@

@

@

@skip

Returns

The transformed pointer or NULL if srcPointer does not point into this

AddressSpaceRange.

void __attribute__::setStartAddress (uint32_t *startAddress*) [private]
Sets the start address.

Parameters

start-Address	The start address of this range.

void __attribute__::setEndAddress (uint32_t *endAddress*) [private]
Sets the end address.

Parameters

endAddress	The first address behind this range.

void __attribute__::setType (Type *type*) [private] Sets the range type.

Parameters

type	The type of this range.

AddressSpaceRange* __attribute__::getPrevious ( ) const [private]
@

@plus@

@plus -@

@

@

@skip
Returns

The previous AddressSpaceRange.

void __attribute__::setPrevious (AddressSpaceRange * *previous*) [private]
Sets the previous AddressSpaceRange.

Parameters

previous	The previous AddressSpaceRange.

AddressSpaceRange* __attribute__::getNext ( ) const [private] @

@plus@

@plus -@

@

@

@skip

Returns

The next AddressSpaceRange.

void __attribute__::setNext (AddressSpaceRange * *next*) [private]

Sets the next AddressSpaceRange.

Parameters

next	The next AddressSpaceRange.

bool __attribute__::doMap (paging::PageDirectory & *pageDir,* AddressSpaceRange const & *otherRange,* paging::PageDirectory & *otherPageDir,* paging::PageFlags const * *flags*) const [private] Maps the range into a certain region on physical address space, described by another address space range, thereby performing a copy of an address space mapping.

The page flags can be overridden for the copy (such that e.g. a read-only ring 3 alias can be created for a read/write ring 0 buffer).

Parameters

pageDir	The page directory to use for the new mapping.
otherRange	The other linear address space range to copy.
otherPageDir	The page directory the other address space range belongs to.

flags	The page flags to use for the mapping. If NULL, the page flags are taken unchanged from the original mapping.

@

@plus@

@plus -@

@

@

@skip

Returns

True if the whole address range could be mapped, else false.

A.5.3 Allocator.h File Reference

Provides a memory allocator to manage a heap.

Data Structures

- class **memory::allocator::Allocator**

  *An **Allocator** (p. 286) object manages a heap.*

- class **memory::allocator::Allocator::CriticalSection**

  *Helper class for entering/leaving the **Allocator** (p. 286)'s critical section.*

Detailed Description

Provides a memory allocator to manage a heap.

A.5.4 AllocatorOutOfMemoryTestCase.h File Reference

Data Structures

- class **memory::allocator::test::AllocatorOutOfMemoryTestCase**

  Tests whether operator new throws an exception in out-of-memory situations.

A.5.5 AtaBusDevice.h File Reference

Data Structures

- class **io::driver::block::ata::AtaBusDevice**

  Class which represent a ata bus.

Variables

- const uint8_t **io::driver::block::ata::PRIMARY_ATA_BUS_MASTER_PIC_PIN_NUMBER** = 6
- const uint8_t **io::driver::block::ata::PRIMARY_ATA_BUS_SLAVE_PIC_PIN_NUMBER** = 7
- const uint16_t **io::driver::block::ata::PRIMARY_ATA_BUS_PORT_START** = 0x1f0
- const uint16_t **io::driver::block::ata::PRIMARY_ATA_BUS_ALTERNATE_PORT_START** = 0x3f0
- const uint16_t **io::driver::block::ata::SECONDARY_ATA_BUS_PORT_START** = 0x170
- const uint16_t **io::driver::block::ata::SECONDARY_ATA_BUS_ALTERNATE_PORT_START** = 0x370

A.5.6 AtaBusDriver.h File Reference

Data Structures

- class **io::driver::block::ata::AtaBusDriver**

  Driver *(p. 455) which creates the AtaBus.*

A.5.7 AtaBusIrqHandler.h File Reference

Data Structures

- class **io::driver::block::ata::AtaBusIrqHandler**

  Interrupt handler for Interrupt which are thrown from an ATA-Bus.

A.5.8 AtaBusWorkerThread.h File Reference

Data Structures

- class **io::driver::block::ata::AtaBusWorkerThread**

  ATA bus worker thread which continuously requests the ATA bus device to initiate the processing of pending ATA commands.

A.5.9 AtaCommand.h File Reference

Data Structures

- class **io::driver::block::ata::commands::AtaCommand**

  Represents a command which can be sent to a hard disk over the ATA bus.

A.5.10 AtaCommandRegister.h File Reference

Created on: 04.09.2013 Author: hfi410.

Data Structures

- struct **io::driver::block::ata::AtaCommandRegister**

Detailed Description

Created on: 04.09.2013 Author: hfi410.

A.5.11 AtaDevice.h File Reference

Data Structures

- class **io::driver::block::ata::AtaDevice**

  A storage block device which is connected to an ATA-Bus.

A.5.12 AtaDigitalOutputRegister.h File Reference

Data Structures

- struct **io::driver::block::ata::AtaDigitalOutputRegister**

A.5.13 AtaDriveLbaHighest.h File Reference

Data Structures

- struct **io::driver::block::ata::AtaDriveLbaHighest**

A.5.14 AtaDriver.h File Reference

Data Structures

- class **io::driver::block::ata::AtaDriver**

  Driver *(p. 455) which creates the AtaDevices.*

A.5.15 AtaErrorRegister.h File Reference

Data Structures

- struct **io::driver::block::ata::AtaErrorRegister**

A.5.16 AtaIdentifyCommand.h File Reference

Data Structures

- class **io::driver::block::ata::commands::AtaIdentifyCommand**

  Command for executing the identify command.

A.5.17 AtaIdentifyResult.h File Reference

Data Structures

- class **io::driver::block::ata::AtaIdentifyResult**

Result of an AtaIdentifyCommand.

A.5.18 AtaIoPorts.h File Reference

Data Structures

- class **io::driver::block::ata::AtaIoPorts**

  *The io ports of an **AtaBusDevice** (p. 305).*

A.5.19 AtaReadCommand.h File Reference

Data Structures

- class **io::driver::block::ata::commands::AtaReadCommand**

  Command which execute a read sectors on the ata bus.

A.5.20 AtaReadWriteTest.h File Reference

Data Structures

- class **io::driver::block::ata::test::AtaReadWriteTest**

A.5.21 AtaReadWriteTest3.h File Reference

Data Structures

- class **io::driver::block::ata::test::AtaReadWriteTest3**

A.5.22 AtaTasklet.h File Reference

Data Structures

- class **io::driver::block::ata::AtaTasklet**

  *Tasklet for the ATA Bus **Device** (p. 436).*

A.5.23 AtaWriteCommand.h File Reference

Data Structures

- class **io::driver::block::ata::commands::AtaWriteCommand**

  Command which execute a write sectors command on the ata bus.

A.5.24 BitField.h File Reference

Data Structures

- class **tool::BitField**< **Base, Index, BitPosition, Length** >

  Template for simple reading/writing bit fields from arrays.

A.5.25 Block.h File Reference

Data Structures

- struct **io::driver::block::Block**

  A struct which represents a 512 byte large block of a block-oriented device.

Variables

- const uint32_t **io::driver::block::BLOCK_SIZE** = 512

A.5.26 BlockDevice.h File Reference

Data Structures

- class **io::driver::block::BlockDevice**

  Class which represent a block-oriented storage device.

A.5.27 BootConsole.h File Reference

Data Structures

- class **boot::BootConsole**

Class for the output during the boot.

A.5.28 BootManager.h File Reference

Data Structures

- class **boot::BootManager**

  Controls the boot process.

A.5.29 BootThread.h File Reference

Data Structures

- class **boot::BootThread**

  Performs the second part of the boot process in a separate boot thread.

A.5.30 bpb.h File Reference

Contains the definition of a **BPB** (p. 375).

Data Structures

- struct **BPB**

  *Describes a **BPB** (p. 375) (BIOS parameter block).*

Detailed Description

Contains the definition of a **BPB** (p. 375).

A.5.31 CleanupTasklet.h File Reference

Data Structures

- class **boot::CleanupTasklet**

  Cleans up some resources which are only used while booting.

A.5.32 Command.h File Reference

Macros

- #define **KERNEL_IPC_DEFINE_COMMAND**(cmd, req)

  Defines a command class.

Typedefs

- typedef bool(∗ **ipc::RequestHandler**)(**ipc::Request** &request, **ipc::Participant** sender)

  Type of a request handler.

Macro Definition Documentation

#define KERNEL_IPC_DEFINE_COMMAND(*cmd, req*) Value:

```
class cmd { \
public : \ \
        static bool handle(ipc::Request &request, ipc::Participant sender) { \
                return cmd().execute(static_cast<req &>(request), sender); \
        } \ \
        bool execute(req &request, ipc::Participant sender) const; \
}
```

Defines a command class.

Parameters

cmd	The name of the Command class (without any namespace qualification; the class is created in that namespace this macro is invoked in).
req	The name of the concrete Request subclass (including any qualifying namespace).

A.5.33 CommandRelay.h File Reference

Data Structures

- class **ipc::CommandRelay**

  *A **CommandRelay** (p. 405) is responsible for sending requests from one process to another or to the kernel.*

A.5.34 Comparator.h File Reference

Data Structures

- class **tool::collection::Comparator**< **T** >

  Abstracts from operator==().

A.5.35 cpu.h File Reference

Contains structures and functions related to the CPU.

Data Structures

- struct **Descriptor**

  Describes a segment descriptor.

Functions

- unsigned long **toLinearAddress** (void const __far *p)

  Convers a segmented pointer to a linear address.

- void **loadGDT** (**Descriptor** const *gdt, std::size_t size)

  Loads the GDT register.

- void **switchToProtectedMode** (unsigned selCode, unsigned selData, unsigned long addrStart, unsigned long addrStackEnd, unsigned long kernelSize, **E820_entry** __far *mmEntry)

  Switches the CPU from Real Mode to Protected Mode.

- template<typename T >
  T const __far * **normalizeSegmentedPointer** (T const __far *p)

  Normalizes a segmented pointer such that its offset is less 0x10.

- template<typename T >
  T __far * **normalizeSegmentedPointer** (T __far *p)

Detailed Description

Contains structures and functions related to the CPU.

Data Structure Documentation

struct Descriptor Describes a segment descriptor.

Data Fields

	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	
	__unnamed-__	

Data Fields

unsigned	limitLow	Bits 0..15 of the segment limit.
unsigned	baseLow	Bits 0..15 of the linear segment base address.
unsigned char	baseHigh1	Bits 16..23 of the linear segment base address.
unsigned char	type1	The first segment type byte.
unsigned char	limitHigh:4	Bits 16..19 of the segment limit.
unsigned char	type2:4	The second segment type byte.

unsigned char	baseHigh2	Bits 24..31 of the linear segment base address.

Function Documentation

unsigned long toLinearAddress (void const __far * *p*) `[inline]` Convers a segmented pointer to a linear address.

Parameters

p	The pointer.

@

@plus@

@plus -@

@

@

@skip

Returns

Its linear address.

template<typename T > T const __far* normalizeSegmentedPointer (T const __far * *p*) `[inline]` Normalizes a segmented pointer such that its offset is less 0x10.

Parameters

p	The pointer to normalize.

@

@plus@

@plus -@

@

@

@skip

Returns

The normalized pointer.

void loadGDT (Descriptor const ∗ *gdt,* std::size_t *size*) Loads the GDT register.

Parameters

gdt	Points to the GDT to load.
size	The size of the GDT in bytes.

void switchToProtectedMode (unsigned *selCode,* unsigned *selData,* unsigned long *addrStart,* unsigned long *addrStackEnd,* unsigned long *kernelSize,* E820_entry __far * *mmEntry*) Switches the CPU from Real Mode to Protected Mode.

Parameters

selCode	The selector for the code segment.
selData	The selector for the data and stack segment.
addrStart	The linear address of the code to be executed after the switch.
addrStack-End	The linear address of the stack pointer to be used after the switch.
kernelSize	The kernel size in bytes.
mmEntry	Points to the first memory map entry. The FAR pointers in the list will be converted to flat pointers, and the linear address of the first memory map entry will be passed to the kernel's entry point as the first argument.

A.5.36 CreateSemaphoreCommand.h File Reference

Functions

- **api::task::lock::KERNEL_IPC_DEFINE_COMMAND** (CreateSemaphore-Command, CreateSemaphoreRequest)

  Handles CreateSemaphoreRequests.

A.5.37 CreateSemaphoreRequest.h File Reference

Data Structures

- class **api::task::lock::CreateSemaphoreRequest**

  Creates a Semaphore.

A.5.38 Cursor.h File Reference

Data Structures

- class **io::driver::graphics::Cursor**

  Class encapsulating a cursor.

A.5.39 DefaultConsole.h File Reference

Data Structures

- class **io::console::DefaultConsole**

  The Console before the VGA Device initialized.

A.5.40 defs.h File Reference

Contains general definitions.

Detailed Description

Contains general definitions.

A.5.41 Delta.h File Reference

Data Structures

- class **io::driver::graphics::Delta**

  Represents a difference between two positions.

A.5.42 dir.h File Reference

Contains functions for loading and traversing the root directory.

Data Structures

- class **RootDirectory**

  Represents the root directory.

- struct **RootDirectory::Entry**

  Describes a directory entry.

- struct **RootDirectory::Entry::Callback**

  Describes a callback. This is an interface.

Detailed Description

Contains functions for loading and traversing the root directory.

A.5.43 disk.h File Reference

Contains functions for loading kernel from disk.

Data Structures

- class **DiskAddress**

  Represents a disk address.

- class **Disk**

  Represents a disk.

Detailed Description

Contains functions for loading kernel from disk.

A.5.44 Environment.h File Reference

Data Structures

- class **memory::allocator::Environment**

  *Allows the **Allocator** (p. 286) to communicate with its environment.*

A.5.45 error.h File Reference

Contains functions for error handling.

Functions

- void **stop** ()

  Stops processor execution.

- void **error** (char const ∗fmt...)

  Prints an error message on the console.

- void **fatal_error** (char const ∗fmt...)

  Prints an error message on the console and stops processor execution.

Detailed Description

Contains functions for error handling.

Function Documentation

void stop ( ) Stops processor execution.

Does not return.

void error (char const ∗ *fmt...*) Prints an error message on the console.

Parameters

fmt	The format of the error message.
...	Additional arguments as required by the message format.

void fatal_error (char const ∗ *fmt...*) Prints an error message on the console and stops processor execution.

Parameters

fmt	The format of the error message.
...	Additional arguments as required by the message format.

A.5.46 ExitThreadCommand.h File Reference

Functions

- **api::task::scheduler::KERNEL_IPC_DEFINE_COMMAND** (ExitThreadCommand, ExitThreadRequest)

  Handles ExitThreadRequests.

A.5.47 ExitThreadRequest.h File Reference

Data Structures

- class **api::task::scheduler::ExitThreadRequest**

  Exits the current thread.

A.5.48 fat.h File Reference

Contains functions for loading and traversing the **FAT** (p. 497).

Data Structures

- class **FAT**

  *Represents a **FAT** (p. 497) (File Allocation Table).*

- class **FAT::FATType**

  ***FAT** (p. 497) type abstraction.*

- class **FAT::FAT12Type**

  Handles FAT12.

- class **FAT::FAT16Type**

  Handles FAT16.

Detailed Description

Contains functions for loading and traversing the **FAT** (p. 497).

A.5.49 GetVersionCommand.h File Reference

Functions

- **api::kernel::KERNEL_IPC_DEFINE_COMMAND** (GetVersionCommand, GetVersionRequest)

  Handles GetVersionRequests.

A.5.50 GetVersionRequest.h File Reference

Data Structures

- class **api::kernel::GetVersionRequest**

  Determines the kernel version.

A.5.51 highmem.h File Reference

Contains functions for dealing with high memory (>= 1 MiB).

Functions

- bool **copyToHighMemory** (void const *srcBuf, std::size_t size, unsigned long linDestAddr)

  Copies bytes above the 1 MiB border.

Detailed Description

Contains functions for dealing with high memory (>= 1 MiB).

Function Documentation

bool copyToHighMemory (void const ∗ *srcBuf,* std::size_t *size,* unsigned long *linDestAddr*) Copies bytes above the 1 MiB border.

Parameters

srcBuf	The source buffer.
size	The number of bytes to copy.
linDestAddr	The linear address of the destination buffer.

@

@plus@

@plus -@

@

@

@skip

Returns

True if successful, false otherwise.

A.5.52 HighPrioThread.h File Reference

Data Structures

- class **task::priorityinheritance::test::HighPrioThread**

  A high prioritized thread that wants to access a resource that a lower prioritized thread has locked exclusively.

A.5.53 InterruptManager.h File Reference

Data Structures

- class **cpu::interrupt::InterruptManager**

  Manages the handler for the hardware interrupts.

Enumerations

- enum **exception** {
  DivisionZero, **Debug**, **NMIInterrupt**, **BreakpointINT3**, **OverflowINTO**, **BOUND**, **InvalidOPCode**, **DeviceNotAvailable**, **DoubleError**, **CoProcessorSegmentOverflow**, **InvalidTaskSwitch**, **Segment-**

NotAvailable,
StackSegmentError, **CommonProtectionFault**, **PageError**, **X86Floating-PointError**,
AlignmentCheck, **MashineCheck**, **SIMDFloatingPointError** }

A.5.54 IRQHandler.h File Reference

Data Structures

- class **io::driver::interrupt::IRQHandler**

  Encapsulates an IRQ handler.

A.5.55 KernelEnvironment.h File Reference

Data Structures

- class **memory::KernelEnvironment**

  Implements an Allocator environment for the kernel mode.

A.5.56 LASMTest.h File Reference

Data Structures

- class **memory::lmm::test::LASMTest**

  *Tests the **LinearAddressSpaceManager*** (p. 597).

A.5.57 LowPrioThread.h File Reference

Data Structures

- class **task::priorityinheritance::test::LowPrioThread**

  A low prioritized thread that has access to a resource that a higher prioritized thread wants to have access to.

A.5.58 memmap.h File Reference

Contains functions for retrieval of the system memory map.

Data Structures

- struct **E820_entry**

  Describes a memory block.

Functions

- **E820_entry** __far ∗ **get_memory_map** ()

  Returns a linked list of memory block descriptors.

- char const ∗ **map_e820_type** (**E820_entry::Type type**)

  Maps an E820 memory type to a user-readable string.

Detailed Description

Contains functions for retrieval of the system memory map.

Data Structure Documentation

struct E820_entry Describes a memory block.

Data Fields

	Type	Possible types of a memory block.

Data Fields

unsigned long long	base	The base address of the memory block.
unsigned long long	length	The length of the memory block in bytes.

unsigned long	type	The type of the memory block.
E820_entry __far *	next	Points to the next entry if available.

Function Documentation

E820_entry __far* get_memory_map ( ) Returns a linked list of memory block descriptors.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to the first memory block described by the E820 BIOS function.

char const* map_e820_type (E820_entry::Type *type*) Maps an E820 memory type to a user-readable string.

Parameters

type	The type to map.

@

@plus@

@plus -@

@

@

@skip

Returns

A pointer to a static string describing the type.

A.5.59 MemoryConstants.h File Reference

Variables

- uint32_t const **memory::KERNEL_SPACE_START** = 0xC0000000
- uint32_t const **memory::USER_SPACE_START** = 0x00100000
- uint32_t const **memory::PAGING_SPACE_START** = 0xFFC00000
- uint32_t const **memory::PAGES_MAPPED_FOR_STACK** = 4

  Synchronize this with KernelStackSize in osloader/main.cc!
- uint32_t const **memory::KERNEL_STACK_SIZE** = PAGES_MAPPED_FOR_STACK ∗ PAGE_SIZE
- uint32_t const **memory::KERNEL_STACK_END** = PAGING_SPACE_START

  *Synchronize this with KernelStackEnd in **kernel/memory/paging/PagingConstants.h** (p. **??**)!*
- uint32_t const **memory::KERNEL_STACK_START** = KERNEL_STACK_END - KERNEL_STACK_SIZE
- uint32_t const **memory::PAGES_MAPPED_FOR_TLS** = 1

  Synchronize this with TLSSize in osloader/main.cc!
- uint32_t const **memory::INITIAL_TLS_SIZE** = PAGES_MAPPED_FOR_TLS ∗ PAGE_SIZE
- uint32_t const **memory::INITIAL_TLS_SPACE_START** = KERNEL_STACK_START - INITIAL_TLS_SIZE
- uint32_t const **memory::VIDEO_AREA_SIZE** = 1 ∗ PAGE_SIZE

  Map only the first video page. Must be a multiple of PAGE_SIZE!
- uint32_t const **memory::VIDEO_AREA_END** = INITIAL_TLS_SPACE_START
- uint32_t const **memory::VIDEO_AREA_START** = VIDEO_AREA_END - VIDEO_AREA_SIZE
- uint32_t const **memory::PHYSICAL_KERNEL_START** = 0x100000
- uint32_t const **memory::PHYSICAL_VIDEO_START** = 0xB8000

  Linear address of video RAM before paging subsystem is initialized.

A.5.60 ModifySemaphoreCommand.h File Reference

Functions

- **api::task::lock::KERNEL_IPC_DEFINE_COMMAND** (ModifySemaphore-Command, ModifySemaphoreRequest)

  Handles ModifySemaphoreRequests.

A.5.61 ModifySemaphoreRequest.h File Reference

Data Structures

- class **api::task::lock::ModifySemaphoreRequest**

  Modifies a Semaphore.

A.5.62 nmi.h File Reference

Contains functions related to masking/unmasking the NMI.

Functions

- void **maskNMIs** ()

  Masks NMIs.

Detailed Description

Contains functions related to masking/unmasking the NMI.

A.5.63 NormalThread.h File Reference

Data Structures

- class **task::priorityinheritance::test::NormalThread**

  A normal thread doing some computations, running at PRIO_NORMAL.

A.5.64 OutputStringCommand.h File Reference

Functions

- **api::io::console::KERNEL_IPC_DEFINE_COMMAND** (OutputStringCommand, OutputStringRequest)

  Handles OutputStringRequests.

A.5.65 OutputStringRequest.h File Reference

Data Structures

- class **api::io::console::OutputStringRequest**

  Prints a string on the console.

A.5.66 Page.h File Reference

Data Structures

- struct **memory::allocator::PageHeader**

  Put at the beginning of a managed page.

- class **memory::allocator::Page**

  *A **Page** (p. 663) applies sub-allocation techniques in order to divide a page into smaller blocks.*

- class **memory::allocator::Page::PartDescriptor**

  *If valid, a **PartDescriptor** (p. 697) describes a contiguous range of parts that are either allocated or free.*

A.5.67 PageFaultExceptionHandler.h File Reference

Data Structures

- class **memory::paging::PageFaultExceptionHandler**

  Handles page fault exceptions.

Variables

- **memory::paging::PageFaultExceptionHandler memory::paging::_-_attribute__**

A.5.68 PageTable.h File Reference

Data Structures

- class **memory::allocator::PageTable**

  *PageTables constitute the first-order organization structure of an **Allocator*** *(p. 286).*

A.5.69 paging.h File Reference

Contains structures and functions related to paging.

Functions

- void **initPagingDirectory** (unsigned long __far ∗pageDir)

  Initializes the page directory in low memory.

- void **addPageMapping** (unsigned long physStart, unsigned long physEnd, unsigned long linStart)

  Adds a mapping for a range of physical pages to the page tables.

Variables

- unsigned long const **PageSize** = 4096

  The page size in bytes.

Detailed Description

Contains structures and functions related to paging.

Function Documentation

void initPagingDirectory (unsigned long __far * *pageDir*) Initializes the page directory in low memory.

Parameters

pageDir	A pointer to the page directory. Page tables will be allocated directly after the page directory.

void addPageMapping (unsigned long *physStart,* unsigned long *physEnd,* unsigned long *linStart*) Adds a mapping for a range of physical pages to the page tables.

Parameters

physStart	The start of the physical address range to be mapped (included).
physEnd	The end of the physical address range to be mapped (excluded).
linStart	The linear address which is to be mapped to the start of the physical address range.

A.5.70 Participant.h File Reference

Data Structures

- class **ipc::Participant**

  Represents an IPC participant.

A.5.71 PICDevice.h File Reference

Data Structures

- class **io::driver::interrupt::PICDevice**

  Top Class for both PIC Devices (Primary and Slave)

A.5.72 PICInterruptHandler.h File Reference

Data Structures

- class **io::driver::interrupt::PICInterruptHandler**

  Interrupt handler installed by the PIC devices that automatically send EOI commands to the relevant PICs.

A.5.73 Position.h File Reference

Data Structures

- class **io::driver::graphics::Position**

  Represents a position.

A.5.74 ProcessManager.h File Reference

Data Structures

- class **task::ProcessManager**

A.5.75 Proxy.h File Reference

Data Structures

- class **ipc::Array**< **T** >

  Encapsulates an array for IPC.

- class **ipc::Attribute**< **T** >

  *Encapsulates a normal attribute in a **Request** (p. 775).*

- class **ipc::Attribute**< **T &** >

  *Encapsulates a reference attribute in a **Request** (p. 775).*

- class **ipc::Attribute**< **T const** >

  *Encapsulates a const attribute in a **Request** (p. 775).*

- class **ipc::Attribute**< **T const &** >

  *Encapsulates a reference-to-const attribute in a **Request** (p. 775).*

- class **ipc::Attribute**< **Array**< **T** > >

  *Encapsulates an array attribute in a **Request** (p. 775).*

- class **ipc::Attribute**< **Array**< **T const** > >

  *Encapsulates an array-of-const attribute in a **Request** (p. 775).*

- class **ipc::Result**< **T** >

  Encapsulates the result of an IPC request.

- class **ipc::Result**< **void** >

Encapsulates a void result of an IPC request.

- class **ipc::Result**< **Array**< **T** > >

  *Encapsulates an **Array** (p. 292) result of an IPC request.*

Macros

- #define **IPC_PROXY_DECLARE_CLASS**(clsName, numRequests)

  An IPC proxy.
- #define **IPC_PROXY_DECLARE_OPERATION0**(id, retType, name)
- #define **IPC_PROXY_DECLARE_VOID_OPERATION0**(id, name)
- #define **IPC_PROXY_DEFINE_OPERATION0**(clsName, name)
- #define **IPC_PROXY_DEFINE_VOID_OPERATION0**(clsName, name)
- #define **IPC_PROXY_DECLARE_OPERATION1**(id, retType, name, p1Type, p1)
- #define **IPC_PROXY_DECLARE_VOID_OPERATION1**(id, name, p1Type, p1)
- #define **IPC_PROXY_DEFINE_OPERATION1**(clsName, name)
- #define **IPC_PROXY_DEFINE_VOID_OPERATION1**(clsName, name)
- #define **IPC_PROXY_DECLARE_OPERATION2**(id, retType, name, p1Type, p1, p2Type, p2)
- #define **IPC_PROXY_DECLARE_VOID_OPERATION2**(id, name, p1Type, p1, p2Type, p2)
- #define **IPC_PROXY_DEFINE_OPERATION2**(clsName, name)
- #define **IPC_PROXY_DEFINE_VOID_OPERATION2**(clsName, name)
- #define **IPC_PROXY_END_CLASS**(clsName)
- #define **IPC_PROXY_DEFINE_CLASS**(clsName) clsName ## Proxy::CommandBase ∗clsName ## Proxy::commands[clsName ## Proxy::NumRequests];
- #define **IPC_PROXY_REGISTER**(clsName, object)
- #define **IPC_PROXY_UNREGISTER**(clsName, object)

Macro Definition Documentation

#define IPC_PROXY_DECLARE_CLASS(*clsName, numRequests*) An IPC proxy.

The sender uses it to access the remote interface. The receiver uses it to map the incoming request to the correct method.

#define IPC_PROXY_DECLARE_OPERATION0(*id, retType, name*) Value:

```
::ipc::Result<retType> name(); \
        static uint32_t const Id_ ## name = id; \
        typedef retType RetType_ ## name;
```

#define IPC_PROXY_DECLARE_VOID_OPERATION0(*id, name*) Value:

```
::ipc::Result<void> name(); \
        static uint32_t const Id_ ## name = id;
```

#define IPC_PROXY_DECLARE_OPERATION1(*id, retType, name, p1Type, p1*) Value:

```
::ipc::Result<retType> name(p1Type p1); \
        static uint32_t const Id_ ## name = id; \
        typedef retType RetType_ ## name; \
        typedef p1Type P1Type_ ## name;
```

#define IPC_PROXY_DECLARE_VOID_OPERATION1(*id, name, p1Type, p1*) Value:

```
::ipc::Result<void> name(p1Type p1); \
        static uint32_t const Id_ ## name = id; \
        typedef p1Type P1Type_ ## name;
```

#define IPC_PROXY_DECLARE_OPERATION2(*id, retType, name, p1Type, p1, p2Type, p2*) Value:

```
::ipc::Result<retType> name(p1Type p1, p2Type p2); \
        static uint32_t const Id_ ## name = id; \
        typedef retType RetType_ ## name; \
        typedef p1Type P1Type_ ## name; \
        typedef p2Type P2Type_ ## name;
```

#define IPC_PROXY_DECLARE_VOID_OPERATION2(*id, name, p1Type, p1, p2Type, p2*) Value:

```
::ipc::Result<void> name(p1Type p1); \
        static uint32_t const Id_ ## name = id; \
        typedef p1Type P1Type_ ## name; \
        typedef p2Type P2Type_ ## name;
```

#define IPC_PROXY_END_CLASS(*clsName*) Value:

```
private : \
        static CommandBase *commands[]; \
}; \
template<> class clsName ## Proxy::Request<0> { \
public : \
        static void initializeRequest() {} \
        static void finalizeRequest() {} \
};
```

#define IPC_PROXY_REGISTER(*clsName, object*) Value:

```
clsName ## Proxy::Request<clsName ## Proxy::NumRequests>:: \
                initializeRequest()
```

#define IPC_PROXY_UNREGISTER(*clsName, object*) Value:

```
clsName ## Proxy::Request<clsName ## Proxy::NumRequests>:: \
                finalizeRequest()
```

A.5.76 Ref.h File Reference

Data Structures

- class **object::Ref**< **T** >

  Represents a reference to a reference-counted object.

Variables

- class **object::Ref object::__attribute__**

A.5.77 RefCountedObject.h File Reference

Data Structures

- class **object::RefCountedObject**

  Represents a reference-counted object.

A.5.78 Registry.h File Reference

Data Structures

- class **ipc::Registry**

  Manages remotely accessible objects.

A.5.79 RemoteObject.h File Reference

Data Structures

- class **ipc::RemoteObject**

A.5.80 Request.h File Reference

Data Structures

- class **ipc::MapRequest**

  Encapsulates a request to map some memory region into some target process.

- class **ipc::Request**

  Represents a request.

A.5.81 runtime.h File Reference

Data Structures

- struct **runtime::TCB**

  *The **TCB** (p. 847) (Thread Control Block).*

A.5.82 runtime.h File Reference

Functions

- void **runtime::initBSS** (uint32_t imageSize, uint32_t *realImageSize=NULL, uint32_t *bssSize=NULL)

  Initializes the BSS section.

A.5.83 SpinLockTestThread.h File Reference

Data Structures

- class **task::spinlock::test::SpinLockTestThread**

A.5.84 StringComparator.h File Reference

Data Structures

- class **tool::collection::Comparator**< **char const** ∗ >

  *Specializes **Comparator*** (p. *407) for char const ∗, using strcmp() for the comparison.*

A.5.85 SysCallHandler.h File Reference

Data Structures

- class **ipc::SysCallHandler**

A.5.86 TextChar.h File Reference

Data Structures

- class **io::driver::graphics::TextChar**

  Class for the characters to print at the console.

A.5.87 ThreadPriority.h File Reference

Enumerations

- enum **ThreadPriority** {
  task::PRIO_LOWEST = 0, **task::PRIO_LOW** = 8, **task::PRIO_NORMAL** = 16, **task::PRIO_HIGH** = 24,
  task::PRIO_HIGHEST = 31, **task::PRIO_NUM** = PRIO_HIGHEST + 1, **task::PRIO_WORKER** = PRIO_HIGH }

  Describes the priority of a thread.

Functions

- ThreadPriority **task::operator+** (ThreadPriority level, int offset)
- ThreadPriority **task::operator-** (ThreadPriority level, int offset)
- ThreadPriority & **task::operator+=** (ThreadPriority &level, int offset)
- ThreadPriority & **task::operator-=** (ThreadPriority &level, int offset)
- ThreadPriority & **task::operator++** (ThreadPriority &level)
- ThreadPriority **task::operator++** (ThreadPriority &level, int)
- ThreadPriority & **task::operator--** (ThreadPriority &level)
- ThreadPriority **task::operator--** (ThreadPriority &level, int)

A.5.88 ThreadQueue.h File Reference

Data Structures

- class **task::scheduler::ThreadQueue**

  Represents a queue of threads.

A.5.89 types.h File Reference

Data Structures

- union **uint128_t**
- struct **TypeInfo**< **T** >
- struct **TypeInfo**< **uint8_t** >
- struct **TypeInfo**< **int8_t** >
- struct **TypeInfo**< **uint16_t** >
- struct **TypeInfo**< **int16_t** >
- struct **TypeInfo**< **uint32_t** >
- struct **TypeInfo**< **int32_t** >
- struct **TypeInfo**< **uint64_t** >
- struct **TypeInfo**< **int64_t** >

Macros

- #define **NULL** 0

Typedefs

- typedef signed char **int8_t**
- typedef unsigned char **uint8_t**
- typedef signed short **int16_t**
- typedef unsigned short **uint16_t**
- typedef signed int **int32_t**
- typedef unsigned int **uint32_t**
- typedef long long **int64_t**
- typedef unsigned long long **uint64_t**

Data Structure Documentation

union uint128_t

Data Fields

uint8_t	as8Bit-Array[16]	
uint16_t	as16Bit-Array[8]	
uint32_t	as32Bit-Array[4]	
uint64_t	as64Bit-Array[2]	

struct TypeInfo

template<typename T>struct TypeInfo< T >

A.5.90 UserModeThread.h File Reference

Data Structures

- class **task::UserModeThread**

A.5.91 UserSpaceInitThread.h File Reference

Data Structures

- class **boot::UserSpaceInitThread**

  This thread is started at the end of the boot process.

A.5.92 utils.h File Reference

Contains utility functions.

Functions

- template<typename T >
  T **min** (T const &a, T const &b)

  Returns the minimum of two values.

- template<typename T >
  T **max** (T const &a, T const &b)

  Returns the maximum of two values.

- std::size_t **strlen** (char const ∗s)
- int **strcmp** (char const ∗s1, char const ∗s2)
- void ∗ **memcpy** (void ∗dest, void const ∗src, std::size_t n)
- int **memcmp** (void const ∗s1, void const ∗s2, std::size_t n)
- char ∗ **strcat** (char ∗destination, const char ∗source)

  Appends a copy of the source string to the destination string.

- char ∗ **strconcat** (const char ∗string1, const char ∗string2)

  Concatenate two strings together and returns a new concatenated string instance which lives on the heap.

- char ∗ **strdup** (char const ∗s)

  Performs a copy of passed string using operator new[].

- char ∗ **strappend** (const char ∗destination, char append)
- void ∗ **memmove** (void ∗dest, void const ∗src, std::size_t n)

  Copies the bytes in a memory region into another one.

- void **fatalError** (const char ∗errorMessage,...) __attribute__((noreturn))

Prints an error message and halts the system.

- char * **unsigned_long_to_str** (unsigned long val, int base)

  Converts an unsigned long value to string.

Detailed Description

Contains utility functions.

Function Documentation

template<typename T > T min (T const & *a*, T const & *b*) [inline]

Returns the minimum of two values.

Parameters

a	The first value.
b	The second value.

@

@plus@

@plus -@

@

@

@skip

Returns

The minimum of a and b.

template<typename T > T max (T const & *a*, T const & *b*) [inline]

Returns the maximum of two values.

Parameters

a	The first value.
b	The second value.

@

@plus@

@plus -@

@

@

@skip
Returns

The maximum of a and b.

char∗ strcat (char ∗ *destination,* const char ∗ *source*) Appends a copy of the source string to the destination string.

The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination.

destination and source shall not overlap.

Parameters

destination	Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string.
source	C string to be appended. This should not overlap destination.

@

@plus@

@plus -@

@

@

@skip
Returns

destination is returned.

char∗ strconcat (const char ∗ *string1,* const char ∗ *string2*) Concatenate two strings together and returns a new concatenated string instance which lives on the heap.

Parameters

string1	The first string which should be concatenated with the second string

string2	The second string which should be concatenated with the first string

@

@plus@

@plus -@

@

@

@skip
Returns

A new string instance on the heap which contains the concatenated string of

Parameters

string1	and
string2	

char∗ strdup (char const ∗ *s*) Performs a copy of passed string using operator new[].

Parameters

s	The string to copy.

@

@plus@

@plus -@

@

@

@skip
Returns

NULL if s==NULL, otherwise a valid array containing the contents of s.

void∗ memmove (void ∗ *dest,* void const ∗ *src,* std::size_t *n*) Copies the bytes in a memory region into another one.

Both memory regions may overlap.

Parameters

dest	Points to the target region.
src	Points to the source region.
n	The number of bytes to copy.

@

@plus@

@plus -@

@

@

@skip

Returns

dest

void fatalError (const char ∗ *errorMessage,* ...) Prints an error message and halts the system.

Parameters

error-Message	Gets printed before the system is halted.

Referenced by tool::collection::ArrayList< T >::operator[ ]() , and tool::collection::LinkedList< T >::operator[ ]().

char∗ unsigned_long_to_str (unsigned long *val,* int *base*) Converts an unsigned long value to string.

Parameters

val	The value to convert.
base	The numeric base to be used. It must lie in the range [2..16].

A.5.93 Version.h File Reference

Variables

- uint32_t const **kernel::Version** = 0x01000000

The kernel version.

A.5.94 VgaCursor.h File Reference

Data Structures

- class **io::driver::graphics::vga::VgaCursor**

  *Implements the **Cursor** (p. 423) interface for the VGA device.*

Variables

- **io::driver::graphics::vga::VgaCursor io::driver::graphics::vga::__attribute-__**

B Index

Index – Handbuch

Active-Object, 188
Adresse
    lineare, 9, 26
    physikalische, 9, 26
Adressraum
    virtueller, 32
Allokator, 14, 17
Alternation
    strenge, 81
Anwendungen, 136
APIC, 68
ATA Festplatte, 182
    Master/Slave Laufwerke, 183
    primärer/PIO Modus, 182
    primärer/sekundärer Bus, 182
Ausführungslevel, 122, 127
    Dispatch, 124, 128
    Passive, 124, 128
Auslastung
    von Seiten, 18
Ausnahme, 66, 69–70
    Abort, 70
    Fault, 70
    Referenz, 72
    Trap, 70

Bibliothek, 136, 139
Bitmap, 26
Bootvorgang, 218
Buddy-Algorithmus, 15

C++
    const, 204
    delete, 197
    Destruktor, 206
    Header, 199
    Heap, 196
    include, 200
    Klasse, 201
    Konstruktor, 202
    new, 197
    Operator, 208
    Referenz, 193
    Vererbung, 205
    virtual, 205
    Zeiger, 194
Call Gate, 46, 110, 113, 119
CMOS, 176
CPL, 44
Critical Region, *see* Kritischer Abschnitt
CRT-Controlle, 167

Destruktor, 203
Dispatcher, 99, 103

Echtzeituhr(RTC), 176, 178

FAT16, 55, 58
    Bootsektor, 55
    Datenregion, 55
    File Allocation Table, 55
    Reservierte Sektoren, 55
    Wurzelverzeichnis, 55
Festplattentreiber, 188
First-Fit-Algorithmus, 14

GDT, *see* Global Descriptor Table
Gemeinsamer Speicher, 113
Global Descriptor Table, 97–99

Handbuch, 224
Heap, 20

Interprozesskommunikation, 108, 113
Interrupt, 67
    Behandler, 74, 77
    Descriptor Table, 66
    maskierbar, 68

nicht maskierbar, 68
Referenz, 72
Interruptkontext, 127
IRQLevel, *see* Ausführungslevel

Kommando Entwurfsmuster, 112
Kommandozeilen Interpreter(CLI), 141
clear, 145
echo, 145
help, 144
Kommandoverzeichnis, 144
Time, 145
version, 145
Konsole, 169, 218
Kontextwechsel, 99
Kritischer Abschnitt, 80
Kritischer Wettlauf, 79

Linear Memory Management, 117
Lock, 82
Logische Blockadressierung (LBA), 182

makeindex, 223
maskieren, 68
Multitasking
kooperatives, 101, 104
präemptives, 80, 105
Mutex, 88
mutual exclusion, *see* Wechselseitiger Ausschluss

Namensräume, 213
new, 20
NMI, 68

Page Directory, 34
Paging, 9, 35
Directory, 35
Table, 35
PIT8254, 171, 176
Preemptives Multitasking, 126, 132
Privileg Level, 113
Privilege Level, 42–45, 110
Current, 44
Descriptor, 44
Requested, 44
Privilegierungsstufen, 107
Privilegstufe, *see* Privilege Level
Programmlader, 136, 138
Prozess, 96
Prozesswechsel, 34
PS2Tastatur, 153
AT, 154
Keycode, 159
MF II, 154
PC/XT, 154
Scancode, 156
Makecode, 156
Scanmatrix, 153
PS2TastaturF
Scancode
Breakcode, 156

Race Condition, *see* Kritischer Wettlauf

Scheduler, 101
Segment, 43
Semaphore, 83, 90
Shared Memory Management, 118
Stack, 196
Stack Wechsel, 46
Synchronisation
Primitive, 83–87

Task
Register, 93
Task-Gate, 95
Tasklets, 126, 132
TaskStateSegment, 93
Taskwechsel, 95
Testframework, 215
Thread, 96

Zustand, 95, 100
Treiber Framework, 147
Geräte Klasse, 150
Treiberimplementierung, 151
Treibernutzung, 151
TSS, *see* TaskStateSegment

Unterbrechung, *see* Interrupt

Vektornummer, 66
Verzeichnisstruktur, 213
Video-RAM, 166
Virtuelles Dateisystem, 53, 56

Warten, 177
aktives, 81
passives, 83
Wechselseitiger Ausschluss, 80