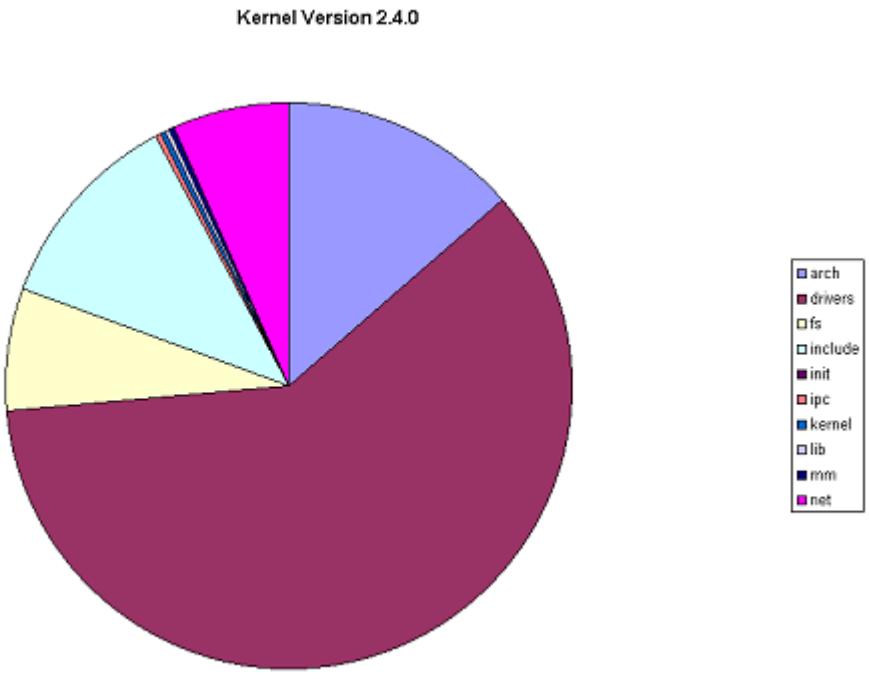
Kapitel 14: Ein-/Ausgabesubsystem

- Motivation und Einführung
- Device Categories I/O-Functionality
- Data Transfer
- I/O-Subsystem
  - Design Parameters
  - I/O Layering
  - I/O Buffering
- Disk I/O Management
  - Disk, CD-Rom, ...
  - Disk Layouts and Formats
  - Disk Scheduling
  - RAID
  - Disk Caching
- Clocks and Timer
- Serial I/O
- Display Hardware and Window Systems
- Power Management

14.1 Motivation und Einführung

Sich prinzipielle Gedanken über ein flexibles, skalierbares und zugleich effizientes Ein-/Ausgabesubsystem zu machen, ist allein schon deswegen nötig, weil beispielsweise heutige Desktop-Betriebssysteme zu ca. 60 % aus E/A-Subsystemsoftware bestehen.



Dass das Ein-/Ausgabesubsystem so viele Zeilen Programmcode umfasst, ist weniger auf dessen komplexere Funktionalität zurückzuführen, als vielmehr auf die Tatsache, dass man generische und somit möglichst portable Ein-/Ausgabetreiber zur Verfügung stellen möchte. Dieser Weg wurde von den Unix-Architekten konsequent verfolgt, die jedes Peripheriegerät als "Spezialdatei" modellierten.

Ein weiterer Grund, warum das Programmieren von Ein-/Ausgabesystemen auch heutzutage immer noch eine Herausforderung darstellt, ist die Tatsache, dass man es dabei auch in Einprozessorsystemen mit echter

Parallelität zu tun hat, in denen von der Peripherie völlig asynchron zur Aktivität auf der Prozessorplatine Ereignisse zur Kenntnis genommen werden müssen, um die relativ langsame Peripherie möglichst permanent am Laufen zu halten. Periphere Geräte können u.U. in sehr viele unterscheidbare Fehlersituationen kommen, die z.T. unabhängig voneinander, aber auch bedingt abhängig voneinander auftreten können. Alle diese Fehlersituationen in der Zentrale richtig zur Kenntnis zu nehmen, erfordert viel Programmiererfahrung an der Hardware-/Softwareschnittstelle.

Evaluationen an den gängigen PC-Systemen haben ergeben, dass die Mehrzahl aller Systemzusammenbrüche auf Fehler im E/A-System zurückzuführen ist.

14.2 Gerätekategorien und E/A-Funktionalität

Geräte lassen sich in folgende Geräteklassen kategorisieren:

- Direkt wahrnehmbar durch den Menschen
- Maschinen lesbar (lokale Kommunikation)
- Kommunikationsgeräte

14.2.1 Direkt wahrnehmbare Geräte

- Bildschirm
- Tastatur
- Maus
- Drucker
- Kopfhörer
- Mikrophon
- Joystick
- Datenhandschuh

14.2.2 Maschinen lesbar

- Plattenlaufwerk
- Bandlaufwerk
- Kontrolleure
- Aktuatoren
- Sensoren

14.2.3 Kommunikationsgeräte

- Modems
- Netzwerkkarten
- ...

14.3 Parameter von E/A-Geräten

Bei der Anbindung der obigen Geräte ist eine Reihe von orthogonalen Entwurfsparametern zu berücksichtigen, die die jeweilige Geräteverwaltung bzw. den Gerätebetrieb beeinflussen können.

- *Data rate* (differences of several order of magnitude)
- *Application* (swap device versus file system may imply different disk access policies)
- *Complexity* of control (printer less complex than disk)
- *Units of transfer* (bytes, stream, blocks)
- *Data representation* (different encoding)
- *Error conditions*

Nimmt man die spezifischen Charakteristika der einzelnen Geräte nicht zur Kenntnis, dann kann man kaum ein effizientes E/A-Subsystem erwarten. Vivek Pai von der Rice-Universität hat bereits 1999 auf der 3. OSDI-Konferenz in New Orleans (TE) I/O-Lite vorgestellt, ein E/A-Subsystem, in dem unnötiges Kopieren von Datenpaketen auf ein Minimum reduziert wurde.

14.3.1 Entwurfsziele eines E/A-Subsystems

Geräteunabhängigkeit:

- Programme können ein abstraktes E/A-Gerät ansprechen, ohne dass es vorab genau spezifiziert sein muss, z.B. man könnte eine Datei lassen, die entweder
  - auf einer Floppy oder
  - auf einer Platte oder
  - auf einer CD-ROM

  abgespeichert ist.

Einheitliche Namensgebung:

- Ob Datei oder Gerät kann entweder ein String oder eine Ganzzahl sein
- Unabhängig vom Maschinen- und Gerätetyp

Fehlerbehandlung:

- Die Fehler sollten so nah an der Hardware wie möglich behandelt werden, so dass die Anwender nur im äußersten Notfall mit Fehlerbehandlungen belästigt werden.

Synchroner oder asynchroner Datentransfer
Zeichenorientierter oder Blockorientierter Datentransfer
Pufferung:

- Einfache Pufferung
- Mehrfach Pufferung

Exklusive oder gemeinsam benutzbare Geräte

14.3.2 Architekturparadigmen der E/A-Verwaltung

Ein-/Ausgabegeräte können mit der Zentrale prinzipiell auf dreierlei Weise angebunden werden:

Programmierte Ein-/Ausgabe

- In der Zentrale wird von einem Thread ein Ein-/Ausgabebefehl ausgeführt, der zur Folge hat, dass der Prozessor solange aktiv darauf wartet, bis dieser E/A-Befehl in der Peripherie fertig gestellt ist, d.h. der Zentralprozessor überprüft permanent im Pollingmodus, das oder die entsprechenden Geräteregister, bis dort eben durch den Eintrag der Peripherie signalisiert wird, dass der E/A-Befehl bearbeitet worden ist.

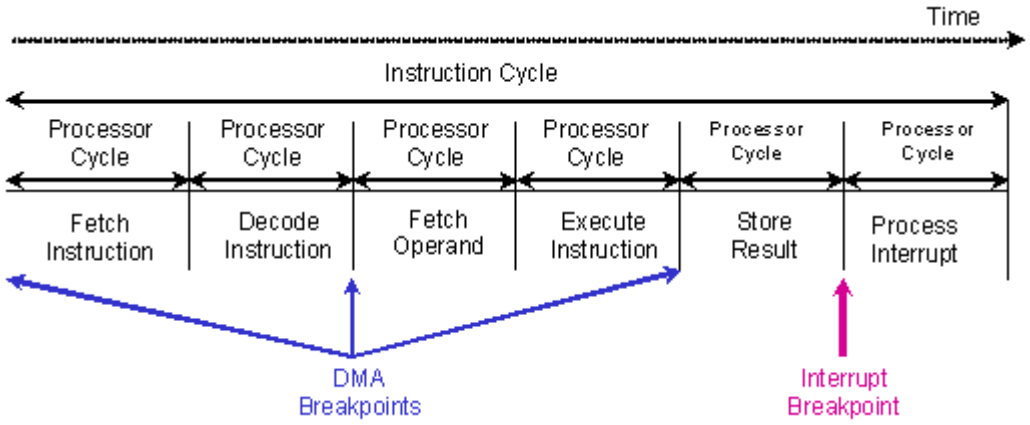
Unterbrechungsgesteuerte E/A

- In der Zentrale wird der E/A-Befehl abgesetzt, allerdings kann der Prozessor auf einen anderen Thread umschalten, im Fall einer synchronen E/A, oder am gleichen Thread weitermachen, im Fall einer asynchronen E/A.
- Nach Abarbeitung des E/A-Befehls sendet die Peripherie ein Unterbrechungssignal an die zentrale, die zu gegebener Zeit (meist unmittelbar) auf dieses Signal reagieren kann.

Direct Memory Access (DMA)

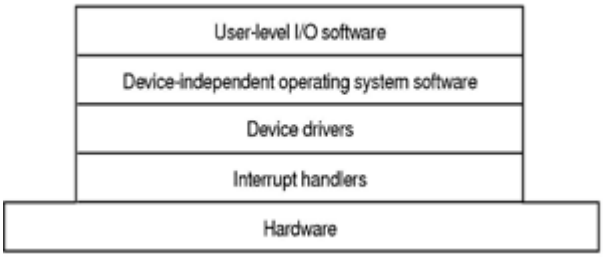
- In der Zentrale wird der E/A-Befehl an den DMA-Baustein abgesetzt, die Zentrale macht wie oben weiter, der DMA-Baustein kontrolliert die weitere E/A-Aktivität.

Es ist zu beachten, dass sich DMA-Baustein und Zentrale u.U. wechselseitig am Systembus behindern, je nach Prozessortyp gibt es im Rahmen einer Befehlspipeline mehrere Zeitpunkte, in denen ein DMA-Baustein einen Buszyklus der Zentrale vor der Nase wegschnappen kann, wohingegen es i.d.R. weniger Zeitpunkte gibt, in denen die Befehlsausführung wegen eines Unterbrechungssignals unterbrochen werden darf, siehe folgende Skizze:



14.3.3 Architekturparadigmen der E/A-Verwaltung

Ein-Ausgabe-Subsysteme sind geeignet, um Software je nach Nähe zur Anwendung bzw. zur Hardware in unterscheidbare Schichten zu gliedern:



14.3.3.1 Unterbrechungsbehandlungen

Stellt die unterste Schicht dar, wird i.d.R. innerhalb des Kerns und auch Mikrokerns implementiert, da meistens auch privilegierte Befehle beim Zugriff auf Geräteregister etc. benötigt werden. Die korrekte Programmierung von Unterbrechungsbehandlungsprogrammen stellt weniger eine Herausforderung dar, wenn die unterbrochene Aktivität stets nur ein Anwenderprogramm im User-Modus wäre. Leider treten aber Unterbrechungssignale völlig asynchron auf, also auch während der Abarbeitung eines Systemaufrufs oder während der Behandlung einer Ausnahmebehandlung oder während einer kurz zuvor aufgetretenen Unterbrechung.

14.3.3.2 Gerätetreiber

Bei der Vielzahl der heutigen Peripheriegeräte werden häufig genug von den Hardwareherstellern auch Gerätetreiber für die gängigen Standardbetriebssysteme angeboten. Aus der Sicht des Betriebssystem-herstellers stellt diese Treibersoftware, insbesondere dann wenn sie aus sonstigen Architekturüberlegungen in den Kern integriert werden muss, eine erhebliche Bedrohung des Gesamtsystems dar.

Aus diesem Grund benötigen wir für diesen Fall ein besseres Konzept, Linux bietet deswegen so genannte "Erweiterbare Kernmodule" an, die bei Bedarf zur Laufzeit geladen werden können und nur in ihrem spezifisch zugewiesenen Teil des Kernadressraums Schaden anrichten können, falls sie fehlerhaft programmiert worden sind.

Multi-Server-Betriebssysteme vermeiden diese Problematik gänzlich, in dem sie jedem Treiber (somit auch allen Fremdprodukten) in einen eigenen logischen Adressraum als Task (oder Prozess) implementieren, und somit eine unerwünschte Beeinflussung des Gesamtsystems ausschalten.

14.3.3.3 Geräteunabhängige Ein-/Ausgabe

Zwischen den einzelnen Geräten mag es mitunter einige Gemeinsamkeiten geben, z.B. bei den diversen Platten, die an ein System angeschlossen werden können, könnte man ein geräte-unabhängiges Caching entwerfen. Geräte unabhängige Software, die sich einer Standardschnittstelle bedient, ist demzufolge leichter auf den verschiedenen Systemplattformen zu integrieren als eine ohne standardisierte Treiberschnittstelle. Genauso wichtig wie eine standardisierte Treiberschnittstelle ist jedoch auch eine standardisierte Kernschnittstelle, d.h. alle Dienste, die ein Treiber vom Kern in Anspruch nehmen will, sind leichter zu implementieren, wenn sie in allen Systemen auf mehr oder weniger gleicher Art erfolgen. Trotz der POSIX-Schnittstelle sind wir jedoch in der Realität doch noch ziemlich weit von dieser standardisierten Kernschnittstelle entfernt.

14.3.3.4 E/A-Pufferung

Das Pufferungskonzept wurde im Zusammenhang mit einer beschleunigten Ein-/Ausgabe erfunden und hat sich mittlerweile auch auf Belange der Anwenderprogrammierung durchgesetzt. Ringpuffer (circular buffering) von mehr als p>2 Pufferelementen sind grundsätzlich dann zu befürworten, wenn die Aktivitätszeiten der Peripherie und der Zentrale abwechselnd mal kürzer oder mal länger dauern, bzw. wenn ein limitierter Burst von E/A-Aktivitäten zu unterstützen ist, unabhängig davon, ob die E/A ständig länger dauert als der zentrale Anteil an der E/A-Durchführung.

14.4 Platten-Ein-/Ausgabe

Der Umgang mit der Systemplatte oder den Systemplatten ist deswegen besonders wichtig, weil neben Dateien auch andere Backup-Information dort untergebracht ist, u.a. kann dort der Hintergrundspeicher für das virtuelle Speichersystem (swap device) realisiert sein. Der Zugriff auf Dateien kann u.U. ganz andere Anforderungen an den Plattentreiber stellen, als dies ein Seitentauscher tun würde.

14.4.1 Entwurfsparameter für einen Plattentreiber (Plattenkontrolleur)

Das erste Problem, das man als Architekt eines Plattentreibers behandeln muss, ist die Frage, ob man überhaupt einen Einfluss auf das Plattenscheduling besitzt, oder ob an der Geräteschnittstelle nur noch eine logische Platte mit b>>1 Blöcken angeboten wird, über deren geometrische Positionierung auf den Plattenoberflächen man überhaupt keine Aussagen mehr treffen kann. In diesem Fall muss jedoch der Plattencontroller entsprechend eingerichtet werden, also ebenfalls "programmiert" werden, so dass die folgenden Überlegungen im Prinzip auf die Gestaltung des Plattencontrollers übertragbar sind.

Entscheidende Parameter für ein Plattenscheduling sind folgende Zeiten:

- Positionierzeit des Lese-/Schreibkopfs über der entsprechenden Plattenspur (seek time)
- Rotationsverzögerung, d.h. die Zeit die vergeht, bis der gewünschte Plattenblock unter dem Lese-/Schreibkopf vorbeikommt

Der mechanische Zeitbedarf für das genaue Positionierung über der gewünschten Spur wird wohl auf absehbare Zeit der dominante Verzögerungsfaktor beim Lesen von bzw. Schreiben auf Platte sein, so dass

man aus diesem Grunde evtl. dazu übergehen könnte, nicht nur einen Plattenblock zu übertragen, sondern gleich mehrere konsekutive Plattenblöcke, sofern dies überhaupt Sinn macht. Bei einer wahllosen Belegung von Plattenblöcken durch eine sequentielle Datei, wird man demzufolge wenig Leistungsgewinn erzielen können.

14.4.2 Plattenzugriffsschedulingstrategien

Folgende Zugriffsstrategien haben ihre Bedeutung:

- FCFS, ist fair, wenn gleich bei Bursts u.U. zu langsam
- SCAN, bei Bursts besser als FCFS, aber schlechte Unterstützung der sequentiellen Abarbeitung, wenn der Kopf von den inneren zu den äußeren Spuren bewegt wird (gerade inverse Reihenfolge der sequentiellen Zugriffe)
- C-SCAN, besser als SCAN, da sequentielle Reihenfolge unterstützt wird
- N-Step SCAN, wie SCAN, jedoch wird ein Burst auf einer Spur nur bedingt unterstützt
- Minimal Seek Time First, kann zum Verhungern von Requests auf inneren Zylindern führen
- SSvTF, wie oben
- Proportional-share scheduling gibt jedem Anwenderprozess einen fairen Anteil an Plattenrequests, d.h. wenn einer zu viele Requests pro Zeiteinheit erhalten hat, aber andere zu kurz gekommen sind, dann werden diese bevorzugt behandelt, bis wieder Fairness erzielt wird.
- Anticipatorisches Scheduling favorisiert zunächst einen weiteren Request der gleichen Anwendung, d.h. durch verzögerte Behandlung eines konkurrenten Requests, erhofft man sich, dass in der Zwischenzeit die zu bevorzugende Anwendung einen weiteren Request durchführen will, so dass dadurch der Plattenkopf erst gar nicht bewegt werden muss.

14.4.3 RAIDS

RAID0 erhöht lediglich die Wahrscheinlichkeit, dass man bei sequentiellem Zugriff quasi im Pipelineverfahren schon die nächsten Plattenblöcke von den p Platten anfordern kann, während man noch an den vorigen p Plattenblöcken in der Zentrale arbeitet. RAID0 reduziert eher die Verfügbarkeit, da nun statt einer Platte p Platten mit gleicher Wahrscheinlichkeit ausfallen können.

RAID1 verwendet eine komplette Spiegelung aller Daten, was natürlich die Verfügbarkeit erhöht. Andererseits muss jedes Update eines Plattenblocks nun auf zwei Platten durchgeführt werden.

RAID5 verwendet zur Erhöhung der Ausfallsicherheit Blockparitäts-Information, die hierbei nicht auf einer einzigen Platte gehalten wird, sondern gleichmäßig über alle Platten verteilt wird. Moderne SAN-Systeme verwenden meistens RAID1 für alle Systemdaten (da diese seltener modifiziert werden) und RAID5 für die Nutzerdaten.

14.4.4 Plattenpufferspeicher

In aller Regel werden heutzutage üppig ausgestattete Plattenpufferspeicher (disk caches) angelegt, um beispielsweise dem Dateisystem zu gestatten, die Aktualisierung der Plattenblöcke verzögert auszuführen (*lazy disk update*). Anders als bei den Hardware-unterstützten Seitentauschalgorithmen, kann man beim softaremäßig realisierten Plattenpufferspeicher sehr wohl ein exaktes LRU-Verfahren praktizieren, was auch in vielen Systemen so gemacht wird.

14.5 Uhren und Zeitgeberbausteine

In heutigen Rechnerarchitekturen gibt es i.d.R. mehrere unabhängig von einander agierende Zeitgeberbausteine, die für unterschiedliche Zeiterfassungen herangezogen werden können. I.d.R. gibt es eine fortlaufende absolute Uhr (batteriegepuffert), die für die Datumsfortschreibung verantwortlich zeichnet (Auflösung im Sekundenbereich). Diese Uhr kann allerdings auch modifiziert werden, wenn beispielsweise

der ihr zugrunde liegende Kristalloszillator sich verspätet oder "verfrüht". Weitere programmierbare Zeitgeberbausteine werden u.a. dazu benutzt, in regelmäßigen Zeitabständen einen Zeitunterbrechung (*timer interrupt*) auszulösen, mit dessen Hilfe ein Zeitscheibenverfahren ermöglicht wird.

14.6 Serielle Ein-/Ausgabe

14.7 Bildschirme und Fenstersystem

14.8 Powermanagement