Aufgabe 1: Ankreuzfragen (22 Punkte)

1) Einfachauswahlfragen (18 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Man unterscheidet zwischen Traps und Interrupts. Welche der folgenden Aussagen ist richtig? 2 Punkte

- ☐ Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- ☐ Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
- ☐ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Interrupts immer an den gleichen Stellen auf.
- ☐ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.

b) Welche Aussage zum Thema federgewichtige Prozesse (User-Level Threads) und leichtgewichtige Prozesse (Kernel-Level Threads) ist richtig? 2 Punkte

- ☐ Zur Umschaltung von federgewichtigen Prozessen ist ein Adressraumwechsel erforderlich.
- ☐ Leichtgewichtige Prozesse können Multiprozessoren nicht ausnutzen.
- ☐ Zu jedem leichtgewichtigen Prozess gehört ein eigener isolierter Adressraum.
- ☐ Federgewichtige Prozesse blockieren sich bei blockierenden Systemaufrufen gegenseitig.

c) Man unterscheidet die Begriffe Programm und Prozess. Welche der folgenden Aussagen zu diesem Thema ist richtig? 2 Punkte

- ☐ Der Prozess ist der statische Teil (Rechte, Speicher, etc.), das Programm der aktive Teil (Programmzähler, Register, Stack).
- ☐ Mit Hilfe von Threads kann ein Prozess mehrere Programme gleichzeitig ausführen.
- ☐ Ein Programm kann durch mehrere Prozesse gleichzeitig ausgeführt werden.
- ☐ Der Binder erzeugt aus einer oder mehreren Objekt-Dateien einen Prozess.

d) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ☐ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ☐ Zeiger vom Typ `void *` existieren in C nicht, da solche „Zeiger auf Nichts“ keinen sinnvollen Einsatzzweck hätten.
- ☐ Zeiger können verwendet werden, um in C eine call-by-reference Übergabesemantik nachzubilden.
- ☐ Der Compiler erkennt bei der Verwendung eines ungültigen Zeigers die problematische Code-Stelle und generiert Code, der zur Laufzeit die Meldung „Segmentation fault“ ausgibt.

e) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux-Dateideskriptoren (file descriptors) zu? 2 Punkte

- ☐ Ein Dateideskriptor ist eine Verwaltungsstruktur, die auf der Festplatte gespeichert ist und Informationen über Größe, Zugriffsrechte, Änderungsdatum usw. einer Datei enthält.
- ☐ Es ist nicht möglich ein und dieselbe Datei in mehreren Prozessen gleichzeitig zu öffnen.
- ☐ Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei, ein Gerät, einen Socket oder eine Pipe benutzen kann.
- ☐ Ein Dateideskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.

f) Was versteht man unter virtuellem Speicher? 2 Punkte

- ☐ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.
- ☐ Speicher, der einem Prozess durch entsprechende Hardware (MMU) und durch Ein- und Auslagern von Speicherbereichen vorgespiegelt wird, aber möglicherweise größer als der verfügbare physikalische Hauptspeicher ist.
- ☐ Einen logischen Adressraum.
- ☐ Adressierbarer Speicher in dem sich keine Daten speichern lassen, weil er physikalisch nicht vorhanden ist.

g) Ein Prozess wird vom Zustand *blockiert* in den Zustand *bereit* überführt. Welche Aussage passt zu diesem Vorgang? 2 Punkte

- ☐ Der Prozess wird wegen eines ungültigen Speicherzugriffs (Segmentation Fault) beendet.
- ☐ Es ist kein direkter Übergang von *blockiert* nach *bereit* möglich.
- ☐ Der Prozess hat auf Daten von der Festplatte gewartet, die nun verfügbar sind.
- ☐ Ein anderer Prozess wurde vom Betriebssystem verdrängt und der erstgenannte Prozess wird nun auf der CPU eingelastet.

h) Welche Aussage zum Thema Systemaufrufe ist richtig?

2 Punkte

- ☐ Durch einen Systemaufruf wechselt das Betriebssystem von der Systemebene auf die Benutzerebene, um unprivilegierte Operationen ausführen zu können.
- ☐ Benutzerprogramme dürfen keine Systemaufrufe absetzen, diese sind dem Betriebssystem vorbehalten.
- ☐ Mit Hilfe von Systemaufrufen kann ein Benutzerprogramm privilegierte Operationen durch das Betriebssystem ausführen lassen, die es im normalen Ablauf nicht selbst ausführen dürfte.
- ☐ Die Bearbeitung eines Systemaufrufs findet immer im selben Adressraum statt, aus dem heraus der Systemaufruf abgesetzt wurde.

i) Prozessoren kennen oft einen unprivilegierten Benutzermodus und einen privilegierten Systemmodus. Welche Aussage ist richtig?

2 Punkte

- ☐ Wird zwischen zwei Prozessen des gleichen Benutzers umgeschaltet, so ist kein Wechsel in den privilegierten Modus nötig.
- ☐ Um zwischen mehreren federgewichtigen Prozessen (User-Level Threads) zu wechseln, ist immer ein Wechsel in den privilegierten Systemmodus erforderlich.
- ☐ Im Benutzermodus ist die Menge an ausführbaren Prozessorbefehlen eingeschränkt.
- ☐ Systemaufrufe erlauben Benutzerprogrammen, beliebige privilegierte Operationen im Systemmodus ausführen zu lassen. Dazu übergibt das Benutzerprogramm zum Beispiel den Maschinencode, der ausgeführt werden soll, als Parameter an das Betriebssystem.

2) Mehrfachauswahlfragen (4 Punkte)

Bei den Mehrfachauswahlfragen in dieser Aufgabe sind jeweils m Aussagen angegeben, davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie alle richtigen Aussagen an.
Jede korrekte Antwort in einer Teilaufgabe gibt einen Punkt, jede falsche Antwort einen Minuspunkt. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, d. h. falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche der folgenden Aussagen zum Thema Dateisysteme, symbolische Verknüpfungen/Links (Symbolic Links) und harte Links (Hard Links) sind richtig?

4 Punkte

- ○ In einem hierarchisch organisierten Dateisystem dürfen gleiche Dateinamen in unterschiedlichen Verzeichnissen enthalten sein.
- ○ Es ist möglich, einen Symbolic Link, der auf eine nicht-existierende Datei verweist, anzulegen.
- ○ Wird der letzte Hard Link auf eine Datei entfernt, so wird auch die Datei selbst gelöscht.
- ○ Wird die Datei gelöscht, auf die ein Symbolic Link verweist, so wird auch der Symbolic Link selbst gelöscht.
- ○ Ein Inode in einem UNIX-Dateisystem enthält unter anderem den Namen der Datei.
- ○ Um den Inhalt einer Datei einlesen zu können, benötigt man Leserechte für das übergeordnete Verzeichnis.
- ○ Ein Hard Link kann auf eine Datei innerhalb eines anderen Dateisystems verweisen.
- ○ Eine Symbolic Link kann auf eine Datei innerhalb eines anderen Dateisystems verweisen.

Aufgabe 2: ffp2 (45 Punkte)

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Ihre Aufgabe ist es, ffp2 (**f**ast **f**ile **p**roperty **p**rinter), eine abgewandelte Variante des Programms wc, zu implementieren. ffp2 erwartet als erstes Argument ein `fnmatch(3p)`-kompatibles glob-Muster, alle weiteren Argumente sind Verzeichnispfade. Das Programm sucht auf den Pfaden rekursiv nach Dateien, deren Name dem Suchmuster entsprechen und ermittelt Eigenschaften wie die Anzahl an darstellbaren Zeichen (`isprint(3p)`) und Zeilen innerhalb der Datei. Die gesammelte Information wird schließlich in einer nach dem Dateipfad sortierten Liste ausgegeben. Um den Ablauf zu beschleunigen, soll das Einlesen der Dateien in **je** einem eigenen Thread geschehen, wobei maximal 8 Fäden gleichzeitig arbeiten dürfen. Eine beispielhafte Nutzung und Ausgabe sieht aus wie folgt:

```
dust@i4corona: ./ffp2 '*sh' /usr/lib/python3.9 /usr/lib/sudo
/usr/lib/python3.9/config-3.9-x86_64-linux-gnu/install-sh: 14847 518
/usr/lib/python3.9/venv/scripts/posix/activate.csh: 833 25
/usr/lib/python3.9/venv/scripts/posix/activate.fish: 1934 64
/usr/lib/sudo/sesh: 6716 25
```

Implementieren Sie dazu folgende Funktionen:

`int main(int argc, char **argv)`
initialisiert die globalen Daten und ruft für jedes übergebene Verzeichnis die Funktion `find_files()` auf. Wurde kein Verzeichnispfad angegeben, wird das aktuelle Ausführungsverzeichnis genutzt. Anschließend wartet die Funktion, bis alle Arbeiterfäden ihre Daten geschrieben haben und sortiert die in dem globalen Array `fileinfos` gesammelten Ergebnisse. Gültige Ergebnisse werden auf `stdout` zeilenweise ausgegeben. Die Ausgabe besteht aus dem Dateipfad, der durch einen Doppelpunkt von der Anzahl an druckbaren Zeichen und Zeilen getrennt ist.

`void find_files(char *path)`
durchsucht rekursiv das übergebene Verzeichnis. Für jede reguläre Datei, deren Name dem Suchmuster entspricht, wird mithilfe von `dispatch_file()` ein eigener Arbeiterfaden gestartet. Das Suchmuster soll auch ohne Angabe eines führenden Punkts auf versteckte Dateien zutreffen. Symbolischen Verknüpfungen soll **nicht** gefolgt werden.

`void dispatch_file(char *path)`
erweitert das dynamische Array `fileinfos` um ein zusätzliches Element und übergibt dieses als Arbeitsauftrag einem Arbeiterfaden (`gather_fileinfo`), der die damit assoziierte Datei bearbeitet.

`void *gather_fileinfo(void *arg)`
erhält in `arg` einen Zeiger auf das zu bearbeitende **`struct`** `fileinfo`. Die Funktion versucht die entsprechende Datei zu öffnen und liest diese aus. Dabei speichert sie die Anzahl an druckbaren Zeichen (`isprint(3p)`) sowie die Anzahl an Zeilen innerhalb der Datei. Im Erfolgsfall wird das Strukturmitglied `valid` auf `1` gesetzt. Der Thread gibt die durch ihn belegten Ressourcen selbst frei.

`int compare(const void *, const void *)`
kann von `qsort(3p)` genutzt werden, um die gefundenen Ergebnisse (**`struct`** `fileinfo`) zu sortieren. Die Sortierung wird durch den Pfad der Datei bestimmt, wobei diese in aufsteigend alphabetischer Reihenfolge geschehen soll.

Hinweis: Nicht-fatale Fehler, wie unzureichende Dateizugriffsrechte, sollen mit einer Fehlermeldung quittiert werden

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

```c
#include <ctype.h>
#include <dirent.h>
#include <errno.h>
#include <fnmatch.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define MAXTHREAD 8

struct fileinfo {
  char *path;
  unsigned print, line;
  int valid;
};

/* @brief Prints error message based on ‘errno‘ and terminates the process.
 * @param msg Part of the error message  */
static void die(const char * const msg);

/* @brief Prints short usage and and exits indicating an error */
static void exit_usage(void);

/* @brief Creates a new semaphore.
 * @param initVal The initial value of the semaphore.
 * @return Hanlde for the created sempahore, or NULL if an error occured. */
SEM *semCreate(initVal);

/* @brief Destroys a semaphore and frees all associated ressources.
 * @param sem Handle of the semaphore to destroy.
 */
void semDestroy(SEM *sem);

/* @brief P-operation
 * @param sem Handle of the semaphore to decrement */
P(SEM *sem);

/* @brief V-operation
 * @param sem Handle of the semaphore to increment */
V(SEM *sem);
```

```
// Vorausdeklarationen
static int compare(const void *a, const void *b);
static void find_files(char *path);
static void dispatch_file(char *path);
static void *gather_fileinfo(void *arg);

// Glob. Variablen
static struct fileinfo ** fileinfos;
```

G:

```
static int compare(const void *a, const void *b) {
}
```

C:

```
int main(int argc, char *argv[]) {
  // Argumente prüfen
  if (argc < 2)
    exit_usage();

  // Globalen Zustand initialisieren
```



```
  // Verzeichnisse durchsuchen
```

```
  // Informationen sortieren und ausgeben
```

```
}
```

M:

```
static void find_files(char *path) {
  // Verzeichnis öffnen

    // Einträge verarbeiten
```



```
}
```

I:

```
static void dispatch_file(char *path) {
```

```
}
```

D:


```
static void *gather_fileinfo(void *arg) {
```

```
}
```

R:

Aufgabe 3: Ausnahmen (12 Punkte)

1) Nennen Sie die zwei Modelle der Ausnahmebehandlung. Wie unterscheiden sich diese Modelle? Nennen Sie für jedes der Modelle je einen Auslösungsgrund. (5 Punkte)

2) Sie haben in der Vorlesung zwei Arten von Ausnahmen von der normalen Programmausführung kennengelernt. Wie lauten diese verschiedenen Arten von Ausnahmen? Nennen sie zwei Eigenschaften, durch die sie sich voneinander unterscheiden. (3 Punkte)

3) Untenstehender Code wird auf einem x86-Linux-System ausgeführt. (4 Punkte)

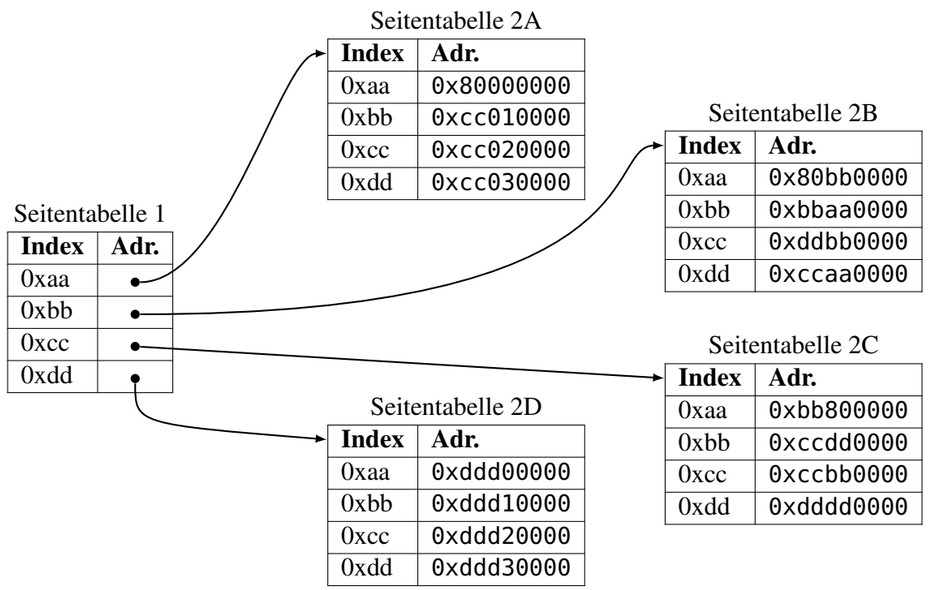
```
char *ptr = NULL;
*ptr = 'c';
```

a) Was für ein Fehler tritt auf? Warum tritt dieser Fehler auf? (2 Punkte)

b) Welche Hardwarekomponente entdeckt diesen Fehler? Wie signalisiert diese Komponente den Fehler an das Betriebssystem? (2 Punkte)

Aufgabe 4: Paging (4 Punkte)

Gegeben sei unten dargestellte Hierarchie zweistufiger Seitentabellen. Die Adresslänge des genutzten Systems sei 32 Bit, die Größe einer Seite 64 Kibibyte (i.e., 2^{16} Byte). Für die Indizierung der zweistufigen Abbildung werden pro Stufe 8 Bit genutzt.



1) Bestimmen Sie die **reale Adresse** zur logischen Adresse `0xccddbb80`. Geben Sie hierbei die zur Bestimmung notwendigen Zwischenschritte stichpunktartig an! (4 Punkte)

Aufgabe 5: Prozesszustände (7 Punkte)

1) Beschreiben Sie die Prozesszustände bei der Einplanung von Prozessen sowie die Ereignisse, die jeweils zu Zustandsübergängen führen (Skizze mit kurzer Erläuterung der Zustände und Übergänge). (7 Punkte)