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) Welche Seitennummer und welcher Versatz gehören bei einer Seitengröße von 1024 Bytes zu folgender logischer Adresse? 0xcafe

2 Punkte

- ☐ Seitennummer 0x32, Versatz 0x2fe
- ☐ Seitennummer 0xc, Versatz 0xafe
- ☐ Seitennummer 0xca, Versatz 0xfe
- ☐ Seitennummer 0x19, Versatz 0x2fe

b) Welche Aussage zum Thema Betriebsarten ist richtig?

2 Punkte

- ☐ Mehrzugangsbetrieb ist nur in Verbindung mit CPU- und Speicherschutz sinnvoll realisierbar.
- ☐ Mehrprogrammbetrieb ermöglicht die simultane Ausführung mehrerer Programme innerhalb desselben Prozesses.
- ☐ Beim Stapelbetrieb können keine globalen Variablen existieren, weil alle Daten im Stapel-Segment (Stack) abgelegt sind.
- ☐ Echtzeitsysteme findet man hauptsächlich auf großen Serversystemen, die eine enorme Menge an Anfragen zu bearbeiten haben.

c) Wodurch kann es in einem System zu Nebenläufigkeit kommen?

2 Punkte

- ☐ Durch dynamische Bibliotheken
- ☐ Durch Interrupts
- ☐ Durch langfristiges Scheduling
- ☐ Durch Traps

d) Welche Aussage ist in einem Monoprozessor-Betriebssystem richtig?

2 Punkte

- ☐ Ein Prozess im Zustand blockiert muss warten, bis der laufende Prozess den Prozessor abgibt und kann dann in den Zustand laufend überführt werden.
- ☐ Es befindet sich zu einem Zeitpunkt maximal ein Prozess im Zustand laufend.
- ☐ In den Zustand blockiert gelangen Prozesse nur aus dem Zustand bereit.
- ☐ Ist zu einem Zeitpunkt kein Prozess im Zustand bereit, so ist auch kein Prozess im Zustand laufend.

e) Welche Aussage über den Rückgabewert von `fork()` ist richtig?

2 Punkte

- ☐ Im Fehlerfall wird im Kind-Prozess -1 zurückgeliefert.
- ☐ Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ☐ Der Kind-Prozess bekommt die Prozess-ID des Vater-Prozesses.
- ☐ Der Rückgabewert ist in jedem Prozess (Kind und Vater) jeweils die eigene Prozess-ID.

f) Nehmen Sie an, der Ihnen bekannte Systemaufruf `stat(2)` wäre analog zu der Funktion `readdir(3)` mit folgender Schnittstelle implementiert: `struct stat *stat(const char *path);` Welche Aussage ist richtig?

2 Punkte

- ☐ Der Systemaufruf liefert einen Zeiger zurück, über den die aufrufende Funktion direkt auf eine Datenstruktur zugreifen kann, die die Dateiattribute enthält.
- ☐ Durch den Zugriff über den zurückgegebenen Zeiger ist es möglich, die Inode-Informationen auf dem Datenträger direkt zu verändern.
- ☐ Der Aufrufer muss sicherstellen, dass er den zurückgelieferten Speicher mit `free(3)` wieder freigibt, wenn er die Dateiattribute nicht mehr weiter benötigt.
- ☐ Ein Zugriff über den zurückgelieferten Zeiger liefert völlig zufällige Ergebnisse oder einen Segmentation fault.

g) Was versteht man unter virtuellem Speicher?

2 Punkte

- ☐ Virtueller Speicher kann dynamisch zur Laufzeit von einem Programm erzeugt werden (Funktion `valloc(3)`).
- ☐ 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.
- ☐ Unter einem virtuellen Speicher versteht man einen physikalischen Adressraum, dessen Adressen durch eine MMU vor dem Zugriff auf logische Adressen umgesetzt werden.

h) Welche Aussage zu Terminvorgaben in Echtzeitsystemen ist korrekt?

2 Punkte

- ☐ Bei festen Terminvorgaben ist eine Terminverletzung tolerierbar, das Ergebnis verliert im Laufe der Zeit aber an Wert.
- ☐ Das Überschreiten einer harten Terminvorgabe ist nicht tolerierbar; daher muss das System in so einem Fall heruntergefahren werden.
- ☐ Beim Überschreiten einer weichen Terminvorgabe wird das Berechnungsergebnis wertlos; die Ausführung wird daher abgebrochen.
- ☐ Das Überschreiten einer harten Terminvorgabe kann zur Katastrophe führen; daher muss für die Anwendung eine Ausnahmebehandlung durchgeführt werden, die zu einem sicheren Zustand führt.

i) Welche der folgenden Aussagen zu statischem bzw. dynamischem Binden ist richtig?

2 Punkte

- ☐ Bei dynamischem Binden müssen zum Übersetzungszeitpunkt alle Adressbezüge vollständig aufgelöst werden.
- ☐ Dynamisch gebundene Programme können auch noch zur Laufzeit durch das Nachladen neuer Programmmodule (Plug-ins) ergänzt werden.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst.
- ☐ Bei statischem Binden werden durch den Compiler alle Adressbezüge vollständig aufgelöst.

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) Man unterscheidet Traps und Interrupts. Welche Aussage ist richtig?

4 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.
- ○ Normale Rechenoperationen können zu einem Trap führen.
- ○ Ein Trap wird immer unmittelbar durch eine Aktivität des aktuell laufenden Prozesses ausgelöst.
- ○ Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie Interrupt einzuordnen.
- ○ Der Zugriff auf eine physikalische Adresse kann zu einem Trap führen.
- ○ Traps dürfen nicht nach dem Wiederaufnahmemodell behandelt werden, da ein Trap immer einen schwerwiegenden Fehler signalisiert.
- ○ Ganzzahl-Rechenoperationen können nicht zu einem Trap führen.

Aufgabe 2: cottbus (49 Punkte)

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

Schreiben Sie ein Programm *cottbus* (**co**ncurren**t** **t**ape **b**ack**u**p **s**olution), das alle regulären Dateien eines Verzeichnisses rekursiv auf ein Bandlaufwerk sichert.

Das Programm arbeitet mit mehreren POSIX-Threads, wobei jeder Thread die Sicherung der Dateien genau eines Verzeichnisses übernimmt. Der Inhalt einer zu sichernden Datei wird jeweils zusammen mit dem Dateinamen und der tatsächlichen Größe der Datei über einen Ringpuffer (Auftragspuffer, Größe `BUFSIZE`) als Backup-Auftrag an den Hauptthread übergeben.

Dieser sichert alle Backup-Aufträge nacheinander auf das Bandgerät. Die Speicherbereiche für die Backup-Aufträge werden wiederverwendet. Hierfür wird beim Programmstart eine feste Zahl von JOB-Strukturen allokiert und in einen zweiten Ringpuffer (Vorhaltepuffer, Größe `BUFSIZE`) eingetragen und somit den Arbeiterthreads zur Verfügung gestellt. Der Hauptthread trägt nach dem Sichern eines Auftrags den zugehörigen Speicherbereich wieder in den Vorhaltepuffer ein.

Das Programm soll folgendermaßen arbeiten:

- Das Programm bekommt als Argumente den Gerätenamen des Bandlaufwerks und das zu sichernde Verzeichnis übergeben. Der Hauptthread ruft zur Erzeugung des ersten Arbeiterthreads die Funktion `spawnThread` auf und füllt den Vorhaltepuffer. Anschließend entnimmt der Hauptthread die abzuarbeitenden Aufträge aus dem Auftragspuffer und schreibt sie mit allen Informationen (Dateiname, Größe und Daten) auf das Bandlaufwerk (Wegschreiben der ganzen Struktur via `fwrite(3)`). Das Schreiben auf das Bandlaufwerk funktioniert wie das Schreiben in eine reguläre Datei. Das Programm beendet sich, wenn keine Arbeiterthreads mehr existieren und alle Aufträge auf das Band gesichert wurden.
- Funktion **`void`** `spawnThread(`**`char`** `*dirname)`: Erzeugt einen neuen Arbeiterthread zum Durchsuchen des Verzeichnisses `dirname`.
- Thread-Startfunktion **`void`** `*tstart(`**`void`** `*dirname)` Durchsucht das übergebene Verzeichnis und ignoriert hierbei die Einträge `.` und `..` sowie alle Einträge, bei denen es sich nicht um eine reguläre Datei oder ein Verzeichnis handelt. Für jede gefundene reguläre Datei wird die Funktion `backupFile` aufgerufen. Für jedes Unterverzeichnis wird die Funktion `spawnThread` zur Erzeugung eines neuen Arbeiterthreads für dieses Unterverzeichnis aufgerufen.
- Funktion **`void`** `backupFile(`**`char`** `*filename)`: Liest die übergebene Datei ein und trägt Aufträge in den Auftragspool ein. Gehen Sie zur Vereinfachung davon aus, dass alle Dateien die Größe von `DATASIZE` Bytes nicht überschreiten.

Hinweise:

- Das `bbuffer`-Modul ist intern synchronisiert und kann nebenläufig verwendet werden. Eine Dokumentation des Ringpuffers finden Sie in der Manpage `bbuffer(3)`.
- `bbGet()` blockiert, falls der Ringpuffer leer ist. Daher schreibt der letzte Arbeiterthread den Wert `NULL` in den Auftragspuffer (*poison pill*) und signalisiert dem Hauptthread so, dass keine weiteren Arbeitspakete vorliegen.
- Sie dürfen annehmen, dass eine Pfadlänge von `PATH_MAX` Zeichen nicht überschritten wird.
- Im Fehlerfall dürfen Sie die Ausführung (bspw. mithilfe von `die()` und `fail()`) abbrechen.
- Ihnen steht das aus der Übung bekannte Semaphor-Modul zur Verfügung. Die Schnittstelle finden Sie im folgenden Programmgerüst nach den **`#include`**-Anweisungen.


```c
/* weitere #includes... */

#define BUFSIZE  32
#define DATASIZE 1048576

#include "bbuffer.h"
#include "sem.h"

/* Funktionen aus sem.h */
SEM *semCreate(int initVal);
void semDestroy(SEM *sem);
void P(SEM *sem);
void V(SEM *sem);


typedef struct JOB {
  char   filename[PATH_MAX]; //< Dateiname der gesicherten Datei
  size_t size;               //< Größe der Daten dieses JOBs
  char   data[DATASIZE];     //< Dateidaten
} JOB;

static void die(const char *msg) {
  perror(msg); exit(EXIT_FAILURE);
}

static void fail(const char *msg) {
  fprintf(stderr, "%s\n", msg); exit(EXIT_FAILURE);
}

// Funktions- & Strukturdekl., globale Variablen, etc.
```

```
// Hauptfunktion
int main(int argc, char *argv[]) {
  if(argc != 3) {
    fail("Usage: ./cottbus <tapedev> <dir>");
  }

  // Puffer und Sem. allokieren
```



```
  // JOBs wegschreiben

  return EXIT_SUCCESS;
} // Ende der Hauptfunktion
```

M:

```
// Einstiegsfunktion der Arbeiterthreads (tstart)
```



```
// Ende Einstiegsfunktion der Arbeiterthreads
```

T:

```
// Funktionen spawnThread & backupFile
```

H:

Aufgabe 3: Dateisystem (12 Punkte)

Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/sp/` (rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/sp/` mit Angabe der Inode-Nummer, des Referenzzählers und der Dateigröße) auf einem Linux-System.

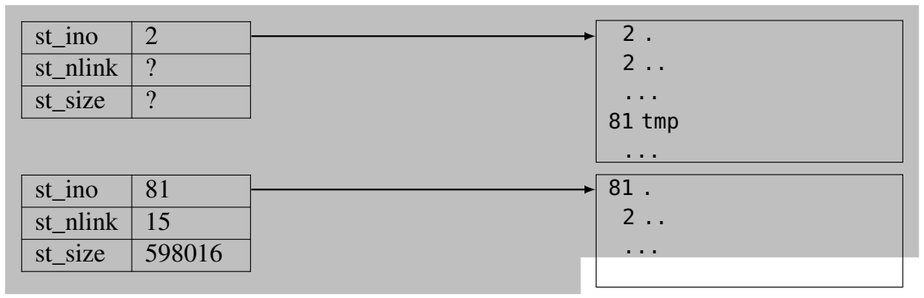
```
chris@legio:~$ ls -aoRi /tmp/sp/
/tmp/sp/:
total 596
39 drwxr-xr-x  3 chris   4096 Jul 26 12:45 ./
81 drwxrwxrwt 15 root  598016 Jul 26 12:34 ../
45 drwxr-xr-x  3 chris   4096 Jul 26 12:43 folder/
67 lrwxrwxrwx  1 chris     12 Jul 26 12:35 important -> folder/file3

/tmp/sp/folder:
total 16
45 drwxr-xr-x 3 chris 4096 Jul 26 12:43 ./
39 drwxr-xr-x 3 chris 4096 Jul 26 12:45 ../
75 drwxr-xr-x 2 chris 4096 Jul 26 12:43 deeper/
49 -rw-r--r-- 2 chris   42 Jul 26 12:38 file1
73 lrwxrwxrwx 1 chris   14 Jul 26 12:43 file2 -> deeper/thefile

/tmp/sp/folder/deeper:
total 16
75 drwxr-xr-x 2 chris 4096 Jul 26 12:43 ./
45 drwxr-xr-x 3 chris 4096 Jul 26 12:43 ../
49 -rw-r--r-- 2 chris   42 Jul 26 12:38 fileX
53 -rw-r--r-- 1 chris  666 Jul 26 12:43 thefile
```

Ergänzen Sie im weißen Bereich die auf der folgenden Seite im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.

Inodes



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

Aufgabe 4: Prozesszustände (7 Punkte)

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).