Aufgabe 1.1: Einfachauswahl-Fragen (22 Punkte)

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

Wollen Sie eine Multiple-Choice-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) Was versteht man unter Virtuellem Speicher? 2 Punkte

- ❏ 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.
- ❏ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.
- ❏ Unter einem Virtuellen Speicher versteht man einen physikalischen Adressraum, dessen Adressen durch eine MMU vor dem Zugriff auf logische Adressen umgesetzt werden.
- ❏ Virtueller Speicher kann dynamisch zur Laufzeit von einem Programm erzeugt werden (Funktion valloc(3)).

b) Welche Aussage über das aktuelle Arbeitsverzeichnis (Current Working Directory) trifft zu? 2 Punkte

- ❏ Jedem UNIX-Benutzer ist zu jeder Zeit ein aktuelles Verzeichnis zugeordnet.
- ❏ Besitzt ein UNIX-Prozess kein Current Working Directory, so beendet sich der Prozess mit einem Segmentation Fault.
- ❏ Mit dem Systemaufruf `chdir()` kann das aktuelle Arbeitsverzeichnis durch den Vaterprozess verändert werden.
- ❏ Pfadnamen, die nicht mit dem Zeichen '/' beginnen, werden relativ zu dem aktuellen Arbeitsverzeichnis interpretiert

c) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (Symbolic Links) und feste Links (Hard Links) auf Dateien. Welche Aussage ist richtig. 2 Punkte

- ❏ Ein Symbolic Link kann nicht auf Dateien anderer Dateisysteme verweisen.
- ❏ Ein Hard Link kann nur auf Verzeichnisse nicht jedoch auf Dateien verweisen.
- ❏ Wird der letzte Symbolic Link auf eine Datei gelöscht, so wird auch die Datei selbst gelöscht.
- ❏ Für jede reguläre Datei existiert mindestens ein Hard-Link im selben Dateisystem.

d) Bei der Behandlung von Ausnahmen (Traps oder Interrupts) unterscheidet man zwei Bearbeitungsmodelle. Welche Aussage hierzu ist richtig? 2 Punkte

- ❏ Das Wiederaufnahmemodell dient zur Behandlung von Interrupts (Fortführung des Programms nach einer zufällig eingetretenen Unterbrechung). Bei einem Trap ist das Modell nicht sinnvoll anwendbar, da ein Trap deterministisch auftritt und damit eine Wiederaufnahme des Programms sofort wieder den Trap verursachen würde.
- ❏ Nach dem Beendigungmodell werden Interrupts bearbeitet. Gibt man z. B. CTRL-C unter UNIX über die Tastatur ein, wird ein Interrupt-Signal an den gerade laufenden Prozess gesendet und dieser dadurch beendet.
- ❏ Interrupts dürfen auf keinen Fall nach dem Beendigungsmodell behandelt werden, weil überhaupt kein Zusammenhang zwischen dem unterbrochenen Prozess und dem Grund des Interrupts besteht.
- ❏ Das Betriebssystem kann Interrupts, die in ursächlichem Zusammenhang mit dem gerade laufenden Prozess stehen, nach dem Beendigungsmodell behandeln, wenn eine sinnvolle Fortführung des Prozesses nicht mehr möglich ist.

e) 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*.
- ❏ Ist zu einem Zeitpunkt kein Prozess im Zustand *bereit*, so ist auch kein Prozess im Zustand *laufend*.
- ❏ In den Zustand *blockiert* gelangen Prozesse nur aus dem Zustand *bereit*.

f) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux Filedeskriptors zu? 2 Punkte

- ❏ Ein Filedeskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei benutzen kann.
- ❏ Filedeskriptoren sind Zeiger auf Betriebssystem-interne Strukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ❏ Ein Filedeskriptor 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.
- ❏ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Filedeskriptor zum Zugriff zurück.

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

- ❏ Das Programm ist der statische Teil (Rechte, Speicher, etc.), der Prozess der aktive Teil (Programmzähler, Register, Stack).
- ❏ Wenn ein Programm nur einen aktiven Ablauf enthält, nennt man diesen Prozess, enthält das Programm mehrere Abläufe, nennt man diese Threads.
- ❏ Ein Prozess ist ein Programm in Ausführung - ein Prozess kann aber auch mehrere verschiedene Programme ausführen
- ❏ Ein Programm kann immer nur von einem Prozess ausgeführt werden

h) Für welchen Zweck wird der Systemaufruf `listen()` benutzt? 2 Punkte

- ❏ Der Aufruf von `listen()` wartet solange an einem Socket, bis eine einkommende Verbindungsanfrage vorliegt.
- ❏ Damit das Betriebssystem überhaupt Systemaufrufe annimmt, muss es erst mit `listen()` in einen Modus des „Zuhörens“ gebracht werden.
- ❏ Mit `listen()` wird ein Socket für die Verbindungsannnahme vorbereitet. Ein Parameter gibt an, wieviele Verbindungsanfragen vor deren Annahme gepuffert werden können.
- ❏ Mit `listen()` wird ein Socket für die Verbindungsannahme vorbereitet. Ein Parameter gibt an, wieviele laufende Verbindungen maximal möglich sind.

i) Ein System setzt Segmentierung und Seitenadressierung ein. Zwei Prozesse sollen ein Segment gemeinsam benutzen. Wie geht das Betriebssystem vor, um das gemeinsame Segment einzurichten? 2 Punkte

- ❏ Das Betriebssystem legt eine Seitentabelle für das gemeinsame Segment an und trägt diese bei beiden Prozessen in die jeweiligen Segmenttabellen ein.
- ❏ Das Betriebssystem trägt jeweils eine Seitentabelle in die prozesseigene Segmenttabelle ein und sorgt dafür, dass die Einträge jeweils auf die gleichen Seitenrahmen
  im Hauptspeicher verweisen.
- ❏ Gemeinsame Segmente können nur durch den Prozess jedoch nicht vom Betriebssystem eingerichtet werden.
- ❏ Die beiden Prozesse bekommen die gleiche Segmenttabelle zugewiesen.

j) Welche Aussage zu einem RAID-1-Plattensystem ist richtig? 2 Punkte

- ❏ Es müssen mindestens drei Festplatten an einem RAID-1-Verbund beteiligt sein.
- ❏ Der Schreibzugriff auf ein RAID-1-Plattensystem ist schneller, da mehrere Platten gleichzeitig beauftragt werden können.
- ❏ Die Paritätsinformation wird gleichmäßig über alle beteiligten Platten verteilt.
- ❏ Ein aus drei Festplatten bestehendes RAID-1-Plattensystem kann den Ausfall zweier Festplatten ohne Datenverlust verkraften.

k) Wozu benötigt man Bedingungsvariablen (condition variables)? 2 Punkte

- ❏ Bei if-Abfragen in C-Programmen.
- ❏ Um in einem kritischen Abschnitt auf ein Ereignis zu warten und den kritischen Abschnitt während der Wartezeit freizugeben.
- ❏ Zur Vermeidung von aktivem Warten in kritischen Abschnitten.
- ❏ Zur Erkennung von Verklemmungen.

Aufgabe 1.2: Mehrfachauswahl-Fragen (8 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe sind jeweils m Aussagen angegeben, n ($0 \leq n \leq m$) Aussagen davon sind richtig. Kreuzen Sie **alle richtigen** Aussagen an. Jede korrekte Antwort in einer Teilaufgabe gibt einen halben Punkt, jede falsche Antwort einen halben Minuspunkt. Eine Teilaufgabse 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 Threads sind richtig? 4 Punkte

- ❍ Bei User-Threads ist die Schedulingstrategie keine Funktion des Betriebssystemkerns.
- ❍ Kern-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ❍ Kern-Threads können Multiprozessoren nicht ausnutzen.
- ❍ Zur Umschaltung von User-Threads ist ein Adressraumwechsel erforderlich.
- ❍ Die Umschaltung von User-Threads ist sehr effizient.
- ❍ Zu jedem Kern-Thread gehört ein eigener Adressraum.
- ❍ Bei Kern-Threads ist die Schedulingstrategie meist durch das Betriebssystem vorgegeben.
- ❍ Die Umschaltung von Kern-Threads muss immer im Systemkern erfolgen.

b) Welche der folgenden Aussagen zum Thema Speicherverwaltung sind richtig? 4 Punkte

- ❍ Das Buddy-Verfahren verhindert externen Verschnitt.
- ❍ Das Buddy-Verfahren verhindert internen Verschnitt.
- ❍ Die Adressen zweier Buddies unterscheiden sich in genau einem Bit.
- ❍ Die Verschmelzung benachbarter Löcher ist beim Buddy-Verfahren besonders einfach.
- ❍ Benachbarte Löcher gleicher Größe können beim Buddy-Verfahren in jedem Fall verschmolzen werden.
- ❍ Bei einer Speicheranforderung muss bei Worst-Fit u.U. die gesamte Freispeicherliste durchlaufen werden.
- ❍ Der Verschmelzungsaufwand bei Best-Fit ist verglichen mit Worst-Fit erhöht.
- ❍ Ziel von First-Fit ist es, den Verwaltungsaufwand gering zu halten.

Aufgabe 2: pete (60 Punkte)

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

Schreiben Sie ein Programm **`pete`** (ParalEl Test Executor), das alle ausführbaren Dateien in einem Verzeichnis (=Testfälle) ausführt und die Anzahl der erfolgreich ausgeführten Testfälle ausgibt.

Das Programm bekommt auf der Befehlszeile vom Benutzer die maximale Anzahl an gleichzeitig laufenden Prozessen (**`maxProcs`**) und eines oder mehrere Verzeichnisse übergeben. In jedem Verzeichnis befinden sich ausführbare Dateien (Testfälle), die durch `pete` parallel in Prozessen ausgeführt werden sollen. Mit Rücksicht auf andere Benutzer des Systems wird die Anzahl der gleichzeitig laufenden Testfälle auf `maxProcs` limitiert. Nach Abschluss aller Testfälle eines Verzeichnisses gibt pete eine kurze Statistikmeldung aus. Um eine spätere Analyse jedes Testfalles zu ermöglichen, wird der Standardfehlerkanal jedes Testfalles in jeweils eine eigene Datei umgeleitet. Zum Schluss gibt `pete` eine Gesamtstatistik über alle Testfälle aus.

Das Programm soll folgendermaßen arbeiten:

- Die main()-Funktion ruft die Funktion **`processDirContent`** für jedes übergebene Verzeichnis auf und wartet anschließend passiv, bis sich alle gestarteten Prozesse beendet haben.

  Wenn `processDirContent` Testfälle gestartet hat, wird eine Statistik mit der Anzahl der erfolgreich ausgeführten Testfälle und der Gesamtzahl aller in diesem Verzeichnis gestarteten Testfälle ausgegeben. Ein Testfall gilt als erfolgreich ausgeführt, wenn der zugehörige Prozess sich mit dem Exitstatus 0 beendet hat.
  Die Ausgabe soll z. B. wie folgt aussehen:

  ```
  9 of 11 testcases successful in directory foo
  ```

  Sollte die Funktion `processDirContent` einen Fehler melden, wird keine Statistik ausgegeben und das Verzeichnis wird in der Gesamtstatistik ignoriert.

  Danach wird mit dem nächsten Verzeichnis weitergemacht. Sobald alle Verzeichnisse abgearbeitet sind, wird die Gesamtzahl aller erfolgreich ausgeführten Testfälle und die Gesamtzahl aller insgesamt gestarteten Testfälle in folgendem Format ausgegeben:

  ```
  27 of 30 testcases successful in all directories
  ```

- Funktion **`int processDirContent(const char *dir)`**: Durchsucht das übergebene Verzeichnis nicht-rekursiv nach ausführbaren, regulären Dateien (Testfällen). Für jeden Testfall wird die Funktion **`executeFile`** aufgerufen. Gibt **`executeFile`** 0 zurück, wird der Testfall als gestartet gezählt. Die Funktion `processDirContent` gibt die Anzahl der gestarteten Testfälle zurück. Sollte ein Fehler auftreten wird eine passende Fehlermeldung ausgegeben und -1 zurückgegeben.
- Funktion **`int executeFile(const char *fileName)`**: Sollten aktuell mehr Prozesse gestartet worden sein als das Befehlszeilen-Argument **`maxProcs`** vorgibt, wird passiv gewartet, bis wieder neue Prozesse erzeugt werden können. Dann wird das übergebene Programm (Testfall) in einem neuen Prozess ausgeführt. Hierbei wird der Standardfehlerkanal in die Datei *`filename`***`.err`** (*fileName* durch den Namen des Testfalls ersetzen!) umgeleitet. Sollte in der Funktion `executeFile` ein Fehler auftreten, wird eine passende Fehlermeldung ausgegeben und -1 zurückgegeben. Im Erfolgsfall wird 0 zurückgegeben.

Auf den folgenden Seiten finden Sie ein Gerüst für das beschriebene Programm. In den Kommentaren sind nur die wesentlichen Aufgaben der einzelnen zu ergänzenden Programmteile beschrieben, um Ihnen eine gewisse Leitlinie zu geben. Es ist überall sehr großzügig Platz gelassen, damit Sie auch weitere notwendige Programmanweisungen entsprechend Ihrer Programmierung einfügen können.

Einige wichtige Manual-Seiten liegen bei - es kann aber durchaus sein, dass Sie bei Ihrer Lösung nicht alle diese Funktionen oder ggf. auch weitere Funktionen benötigen.

```
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

#define S_ISEXEC(mode) (((S_IXUSR|S_IXGRP|S_IXOTH)&mode) > 0 )

/* Funktions- und Strukturdeklarationen, globale Variablen */
```

```
/* Funktion main */
```

A:


```
/* Argumente verarbeiten */
```

```
/*  Signalbehandlung und Signalmasken initialisieren */
```

S:

```
    /* uebergebene Verzeichnisse abarbeiten */
    ...
    ...
    ...
    ...
       /* Terminieren der Testfälle abwarten */
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
        /* Statistik fuer Directory ausgeben */
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    /* Gesamtstatistik ausgeben */
    ...
    ...
    ...
    ...
}
/* Ende Funktion main */
```

D:


```
/* Signalbehandlung fuer SIGCHLD */
...
...
...
...
   /* Prozesszahl und erfolgreich ausgef. Testfaelle erfassen */
...
...
...
...
...
...
...
...
...
...
/* Funktion processDirContent */
...
...
...
...
...
...
...
...
...
...
```

S:

```
    /* Verzeichnis durchlaufen */

        /* falls Datei = Testfall, Datei ausfuehren */

    /* aufraeumen */

} /* Ende Funktion processDirContent */
```

P:


```
/* Funktion executeFile */

    /* Prozess erzeugen und Testfall starten */
```

```
/* Fortsetzung ... */
```

E:

Aufgabe 3: (21 Punkte)

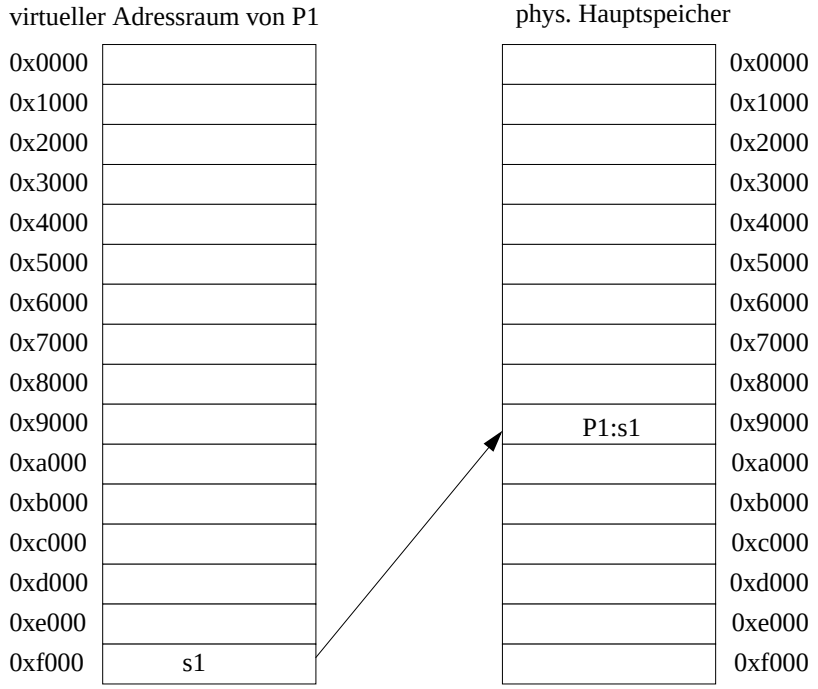
In einem System mit Seitenadressierung (paged address space), Adresslänge = 16Bit, Seitengröße = 4 KiBi Bytes, Hauptspeichergröße = 64 KiBi Bytes wird ein Programm durch einen Prozess P1 ausgeführt. *(zur Erinnerung :-) : $4096_{10} = 1000_{16}$)*

Die erste Seite des virtuellen Adressraums wird nicht genutzt. Das Textsegment des Prozesses umfasst 2 Seiten (t1 und t2), das Datensegment umfasst ebenfalls 2 Seiten (d1 und d2), direkt im Anschluss daran. Das Stacksegment umfasst eine Seite (s1) ganz am Ende des virtuellen Adressraums.

Die Seiten t1 und t2 liegen im Hauptspeicher in den Seitenrahmen (=Kacheln) auf Adresse 0x4000 und 0x7000, die Seite d1 liegt in Seitenrahmen 0x6000, die Seite d2 ist ausgelagert und liegt auf der Platte im Block 0x37000, die Seite s1 liegt in Seitenrahmen 0x9000.

Das (sehr kleine) Betriebssystem (BS) belegt die ersten 4 Seitenrahmen, die Seitenrahmen von 0xa000 bis 0xf000 sind durch einen anderen Prozess (P2) belegt.

a) Tragen Sie die den Aufbau des virtuellen Adressraums (wo liegen welche Seiten) von P1, die Belegung aller Seitenrahmen des phys. Hauptspeichers und die Abbildung (Pfeile) analog zu dem Beispiel von Seite s1 in der nachfolgenden Skizze ein. (5 Punkte)



b) Die Seiten des Textsegments seien read-only in den Adressraum abgebildet, alle anderen Seiten zum Lesen und Schreiben. Die Ausführung von Maschinenbefehlen aus dem Daten- oder Stacksegment ist nicht zulässig.
Skizzieren Sie die Datenstrukturen, die die MMU für die Umsetzung des beschriebenen virtuellen Adressraums benötigt und wie dabei die logische Adresse 0x2260 umgesetzt wird. (11 Punkte)

c) In dem Programm stehen folgende Anweisungen:

```
1   int a;
2   static int b;
3   void f1() {
4      static int *p = (int *)0x4018;
5      int i;
       ...
6      *p = 41;
7      i = 5;
       ...
```

Beantworten Sie die folgenden Frage in kurzen Stichworten (6 Punkte):

c1) Wo im virtuellen Adressraum (in welchen Seiten) liegen die Variablen a, b, p und i?

c2) Was passiert bei der Ausführung der Anweisung in Zeile 6? Warum?

c3) Wie geht das Betriebssystem mit dieser Situation um?

Aufgabe 4: (9 Punkte)

a) Was versteht man unter einer Verklemmung und was sind die Voraussetzungen, damit eine Verklemmung auftreten kann? (6 Punkte)

b) Eine Maßnahme gegen Verklemmungen ist Verklemmungsvorbeugung (*deadlock prevention*). Beschreiben Sie eine Methode, wie man hierbei vorgehen kann und geben Sie ein konkretes Beispiel für eine Vorbeugungsmaßnahme an. (3 Punkte)