Aufgabe 1.1: Einfachauswahl-Fragen (18 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) Welche Aussage zum Thema Adressraumschutz ist **richtig**? 2 Punkte

- ❏ Beim Adressraumschutz durch Eingrenzung ist es prinzipbedingt nicht möglich, dass mehrere Prozesse auf ein Stück gemeinsamen Speichers zugreifen.
- ❏ Beim Adressraumschutz durch Abteilung wird der logische Adressraum in mehrere Segmente mit unterschiedlicher Semantik unterteilt.
- ❏ Bei allen Verfahren des Adressraumschutzes führt jeder Zugriff auf eine ungültige Speicheradresse zu einem Trap.
- ❏ In einem segmentierten Adressraum kann zur Laufzeit kein weiterer Speicher mehr dynamisch nachgefordert werden.

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

- ❏ Der Prozess ist der statische Teil (Rechte, Speicher, etc.), das Programm 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 Prozess kann mit Hilfe von Threads mehrere Programme gleichzeitig ausführen.

c) Welche der folgenden Aussagen zum Thema Adressräume ist richtig? 2 Punkte

- ❏ Der logische Adressraum ist ebenso wie der physikalische Adressraum durch die gegebene Hardwarekonfiguration definiert.
- ❏ Der virtuelle Adressraum eines Prozesses kann nie größer sein als der physikalisch vorhandene Arbeitsspeicher.
- ❏ Der physikalische Adressraum ist durch die gegebene Hardwarekonfiguration definiert.
- ❏ Die maximale Größe des virtuellen Adressraums kann unabhängig von der verwendeten Hardware frei gewählt werden.

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) 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.
- ❏ 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.
- ❏ Durch den Zugriff über den zurückgegebenen Zeiger ist es möglich, die Inode-Informationen auf dem Datenträger direkt zu verändern.

f) Was versteht man unter einem Interrupt?

2 Punkte

- ❏ Eine Signalleitung teilt dem Prozessor mit, dass er den aktuellen Prozess anhalten und auf das Ende der Unterbrechung warten soll.
- ❏ Mit einer Signalleitung wird dem Prozessor eine Unterbrechung angezeigt. Der Prozessor sichert den aktuellen Zustand bestimmter Register, insbesondere des Programmzählers, und springt eine vordefinierte Behandlungsfunktion an.
- ❏ Der Prozessor wird veranlasst eine Unterbrechungsbehandlung durchzuführen. Der gerade laufende Prozess kann die Unterbrechungsbehandlung ignorieren.
- ❏ Durch eine Signalleitung wird der Prozessor veranlasst, die gerade bearbeitete Maschineninstruktion abzubrechen.

g) Welche Aussage zum Thema Betriebsarten ist **richtig**?

2 Punkte

- ❏ 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.
- ❏ Mehrzugangsbetrieb ist nur in Verbindung mit CPU- und Speicherschutz sinnvoll realisierbar.
- ❏ Mehrprogrammbetrieb ermöglicht die simultane Ausführung mehrerer Programme innerhalb desselben Prozesses.

h) Welche Aussage zu Zeigern ist **richtig**?

2 Punkte

- ❏ Zeiger können verwendet werden, um in C eine call-by-reference Übergabesemantik nachzubilden.
- ❏ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference.
- ❏ 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.

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

2 Punkte

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

Aufgabe 1.2: Mehrfachauswahl-Fragen (4 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 zu UNIX-Dateisystemen sind richtig?

- ❍ Dateien (Name, Attribute und Inhalt) werden in Dateikatalogen abgespeichert.
- ❍ Beim Anlegen einer Datei wird die maximale Größe festgelegt. Wird sie bei einer Schreiboperation überschritten, wird ein Fehler gemeldet.
- ❍ Ein Dateikopf ist eine Verwaltungsstruktur, die vorne in der Datei gespeichert wird.
- ❍ Ein *hard link* ist ein Verweis aus einem Katalog auf eine Dateiverwaltungsstruktur (*Inode*).
- ❍ Auf einen Katalog darf immer nur ein *hard link* existieren.
- ❍ Man darf einen *hard link* auf eine Datei erzeugen, auch wenn man keine Zugriffsrechte auf diese Datei hat.
- ❍ In einem Verzeichnis darf es nicht mehrere Einträge mit gleichem Namen geben, selbst wenn diese auf verschiedene *Inodes* verweisen würden.
- ❍ Obwohl eine Datei gelöscht wurde, kann es *symbolic links* geben, die noch auf sie verweisen.

Aufgabe 2: pvt (45 Punkte)

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

Schreiben Sie ein Programm pvt (Parallel Video Transcoder), welches die Auflistung und den Abruf von Video-Dateien mit gleichzeitiger Umkodierung ermöglicht.

Das Programm erwartet genau zwei Kommandozeilenparameter: Der erste Parameter gibt das auszuführende Kommando an, der zweite Parameter ist kommandospezifisch.

Die unterstützten Kommandos und die Bedeutung ihrer Parameter:

- **`LIST`**: Gibt die Namen aller Dateien in einem Verzeichnis aus. Als Parameter wird der Name des Verzeichnisses angegeben.
- **`PLAY`**: Gibt eine Videodatei in transkodiertem Format aus. Als Parameter wird der Name der Datei angegeben.

Detailierte Beschreibung der zu implementierenden Funktionen:

- **`main`**-Funktion: Abhängig vom angegebenen Kommando werden die Funktionen **`printDirContent()`** oder **`printVideoFile()`** aufgerufen.
- **`void printDirContent(char *)`**: Die Funktion gibt die Dateinamen aller Einträge des übergebenen Verzeichnisses zeilenweise aus. Dabei werden alle Einträge ignoriert, bei denen es sich nicht um eine reguläre Datei oder ein Verzeichnis handelt. Handelt es sich um den Namen eines Verzeichnisses, so wird dies bei der Ausgabe durch ein zusätzliches "`/`"-Zeichen nach dem Dateinamen verdeutlicht.
- **`void printVideoFile(char *)`**: Die Funktion öffnet die angeforderte Videodatei und übergibt diese an die Funktion **`transcodeVideo()`**. Anschließend wartet die Funktion passiv, bis das komplette Video ausgegeben wurde.
- **`SEM *transcodeVideo(FILE *)`**: Die Funktion liest Video-Frames mit Hilfe der Funktion **`fread(3)`** aus der übergebenen Videodatei aus und erzeugt für jeden einen Thread (Funktion **`tstart()`**), der das Transkodieren und die Ausgabe vornimmt.
  Da die Frames in derselben Reihenfolge ausgegeben werden müssen, in der sie auch eingelesen wurden, muss die Ausgabe entsprechend synchronisiert werden. Dabei muss ein Thread warten bis der vorherige Frame ausgegeben wurde und, nachdem er selbst seinen Frame ausgegeben hat, dies dem nächsten Thread signalisieren.
  Dies kann mit Hilfe eines Semaphors pro Thread erreicht werden, den der Thread zur Signalisierung des nächsten Threads benutzt. Beachten Sie die ggf. gesonderte Behandlung am Anfang und am Ende.
  Nach dem Auslesen des letzten Frames gibt die Funktion den Semaphor zurück, mit dem auf die Beendigung des letzten Threads gewartet werden kann.
  Sollte bei der Ausführung ein Fehler auftreten, soll – wenn möglich – der aktuelle Frame verworfen werden und mit dem nächsten fortgefahren werden.
- Thread-Funktion **`void *tstart(threadArg *)`**: Die Funktion ruft für den übergebenen Frame die Funktion **`transcodeFrame()`** auf, die daraus einen neuen Frame berechnet. **`tstart`** gibt diesen anschließend mit Hilfe der Funktion **`fwrite(3)`** auf stdout aus.
- **`Frame *transcodeFrame(Frame *)`**: **Nehmen Sie diese Funktion als gegeben an!**
  Die Funktion transkodiert den übergebenen Frame, schreibt das Ergebnis in einen neu erzeugten Frame und gibt einen Zeiger auf diesen zurück. Die Funktion kann nicht fehlschlagen. Für die Freigabe des zurückgegebenen Frames ist der Aufrufer zuständig.

Ihnen steht das aus der Übung bekannte Semaphoren-Modul zur Verfügung. Die Schnittstelle finden Sie im folgenen Programmgerüst nach den #include-Anweisungen.

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 gegebenenfalls auch weitere Funktionen benötigen.

```
/* includes */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "sem.h"

typedef struct Frame {
        char pixels[PIXEL_COUNT];
} Frame;

typedef struct threadArg {
        SEM* waitFor;
        SEM* signal;
        Frame frame;
} threadArg;

SEM *semCreate(int initVal);
int semDestroy(SEM *sem);
void P(SEM *sem);
void V(SEM *sem);

Frame *transcodeFrame(Frame *);

static void die(const char message[]) {
        perror(message);
        exit(EXIT_FAILURE);
}
```

// Makros, Funktionsdeklarationen, Strukturen, globale Variablen

// Funktion main

// Argumente auswerten und entspr. Funktionen aufrufen

// Ende Funktion main

A:

```
// Funktion printDirContent
```

D:


```
// Funktion printVideoFile
```

V:

```
// Funktion transcodeVideo
```

C:


```
// Thread-Funktion
```

T:

Aufgabe 3: (20 Punkte)

a) Gegegeben sei das unten stehende Programm.

Skizzieren Sie den Aufbau des logischen Adressraums eines Prozesses, der dieses Programm ausführt.

– Wozu werden welche Segmente angelegt?

– Zeichnen Sie für jede Variable ein, wo diese ungefähr im logischen Adressraum zu finden sein wird.

– Wohin zeigen die Zeiger m, p und q (ungefähre Position einzeichnen!)?

```
int ga;
static int gb;
char *m = "message";

int main() {
  int la;
  static int lb;
  int *p;
  static double *q;

  p = malloc(100 * sizeof(int));
  q = malloc(100 * sizeof(double);

}
```


b) Welche Arten von Schutz gewährleisten logische Adressräume?

c) Wenn das folgende Programmstück in einem UNIX-System abläuft, wird ein Fehler auftreten.

```
...
int *p = NULL;
...
*p = -1;
...
```

Bitte möglichst knappe, aber präzise Antworten:

c1) Welcher Fehler tritt auf?

c2) Warum tritt der Fehler auf?

c3) Der Fehler wird von einer Hardwarekomponente zuerst entdeckt - welche Komponente ist das?

c4) Mit welchem Mechanismus wird der Fehler dem Betriebssystem mitgeteilt?

c5) Was macht das Betriebssystem mit dem Prozess, der gerade das Programmstück ausführt?