Aufgabe 1.1: Einfachauswahl-Fragen (3 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) Ein *laufender* Prozess wird in den Zustand *bereit* überführt. Welche Aussage passt zu diesem Vorgang? 1,5 P.

- ❑ Der Prozess wartet auf Daten von der Festplatte.
- ❑ Der Prozess wartet mit dem Systemaufruf `waitpid()` auf die Beendigung eines anderen Prozesses.
- ❑ Der Prozess wird durch einen anderen Prozess verdrängt (*preemption*) oder gibt die CPU freiwillig ab (*yield*).
- ❑ Es ist kein direkter Übergang von *laufend* nach *bereit* möglich.

b) Bei Programmunterbrechungen (Ausnahmen) unterscheidet man zwischen Traps und Interrupts. Welche Aussage zu Traps ist richtig? 1,5 P.

- ❑ Ein Trap steht nicht zwangsläufig in ursächlichem Zusammenhang mit dem unterbrochenen Programm.
- ❑ Der Zugriff auf eine physikalische Speicheradresse kann zu einem Trap führen.
- ❑ Traps werden immer nach dem Beendigungsmodell behandelt.
- ❑ Normale Ganzzahl-Rechenoperationen (z. B. Addition, Division) können nicht zu einem Trap führen.

Aufgabe 1.2: Mehrfachauswahl-Fragen (3 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 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 sind richtig? 3 Punkte

- ❍ Ein Hardlink kann auf eine Datei innerhalb eines anderen Dateisystems verweisen.
- ❍ Wird der letzte Hardlink auf eine Datei entfernt, so wird auch die Datei selbst gelöscht.
- ❍ Ein Inode in einem UNIX-Dateisystem enthält Metadaten über eine Datei: Größe, Eigentümer, Zugriffsrechte, Dateiname usw.
- ❍ In einem UNIX-Dateisystem ist es möglich, dass derselbe Inode unter mehreren Dateinamen erreichbar ist.
- ❍ In einem hierarchisch organisierten Dateisystem dürfen gleiche Dateinamen in unterschiedlichen Verzeichnissen enthalten sein.
- ❍ Um den Inhalt einer Datei einlesen zu können, benötigt man Leserechte für das übergeordnete Verzeichnis.

Aufgabe 2: spawn (15 Punkte)

Sie dürfen diese Seite und die Manual-Seite am Ende der Klausur zur besseren Übersicht bei der Programmierung heraustrennen!

Schreiben Sie eine Funktion **spawn**

```
int spawn(const char *app, char *argv[], const char *dirs[]);
```

die eine Programmdatei (Parameter **app**) in einer Menge von Verzeichnissen (gegeben durch ein **NULL**-terminiertes Array von Strings, Parameter **dirs**) sucht.

Die erste hierbei gefundene reguläre Datei, die für jedermann ausführbar ist, wird in einem neuen Kindprozess mit den bereitgestellten Argumenten (Parameter **argv**) ausgeführt. Anschließend wartet der Vaterprozess, bis sein Kindprozess terminiert, und gibt dessen Exit-Status zurück.

Tritt beim Erzeugen des Kindprozesses ein Fehler auf oder beendet sich der Kindprozess **nicht** regulär, so gibt **spawn** den Wert -1 zurück. Falls die Programmdatei in keinem der Verzeichnisse gefunden wurde, wird -2 zurückgegeben. Die Funktion soll keine Meldungen ausgeben, auch keine Fehlermeldungen.

Anmerkungen:

- Pfad-Konkatenationen können entweder mit **strcpy(3)** und **strcat(3)** oder mit **sprintf(3)** durchgeführt werden.
- Zum Test, ob eine Datei für jedermann ausführbar ist, können Sie das vorgegebene Makro **IS_EXECUTABLE(mode)** verwenden. Dieses erwartet als Parameter den Wert des **st_mode**-Felds der **stat**-Struktur.
- Ein Prozess gilt als regulär beendet, wenn das Makro **WIFEXITED** einen Wert ungleich 0 zurückliefert.

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

// Hilfsmakro IS_EXECUTABLE()
#define EXEC_ALL (S_IXUSR | S_IXGRP | S_IXOTH)
#define IS_EXECUTABLE(mode) (((mode) & EXEC_ALL) == EXEC_ALL)
```



```
// Funktion spawn()
```

Aufgabe 3: (9 Punkte)

a) Erläutern Sie das Konzept *Semaphor*. Welche Operationen sind auf Semaphoren definiert und was tun diese Operationen? (6 Punkte)

b) Skizzieren Sie in Programmiersprachen-ähnlicher Form, wie mit Hilfe eines zählenden Semaphors das folgende Szenario korrekt synchronisiert werden kann: Ein Hauptthread soll so lange warten, bis 7 Arbeiter-Threads ihre Arbeit (Ausführung einer Funktion `doJob()`) erledigt haben. (3 Punkte)

Hauptthread:	**Arbeiter-Thread:**
`sem = sem_init(0);`	`doJob();`
`startWorkerThreads();`	