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

b) Welche Aussage über `exec()` ist richtig? 1,5 P.

- ❏ Der an `exec()` übergebene Funktionszeiger wird durch einen neuen Thread im aktuellen Prozess ausgeführt.
- ❏ Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ❏ `exec()` erzeugt einen neuen Kind-Prozess und startet darin das angegebene Programm.
- ❏ Das im aktuellen Prozess laufende Programm wird durch das angegebene Programm ersetzt.

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 *hard-link* kann auf eine Datei innerhalb eines anderen Dateisystems verweisen.
- ❍ Ein Inode in einem UNIX-Dateisystem enthält Metadaten über eine Datei: Größe, Eigentümer, Zugriffsrechte, Dateiname usw.
- ❍ In einem hierarchisch organisierten Dateisystem dürfen gleiche Dateinamen in unterschiedlichen Verzeichnissen enthalten sein.
- ❍ Auf ein Verzeichnis darf immer nur genau ein *hard-link* verweisen.
- ❍ In einem UNIX-Dateisystem ist es möglich, dass derselbe Inode unter mehreren Dateinamen erreichbar ist.
- ❍ Wird der letzte *hard-link* auf eine Datei entfernt, so wird auch die Datei selbst gelöscht.

Aufgabe 2: build (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 **`build`**

```
int build(const char cFile[], const char oFile[]);
```

die - ähnlich zu der Funktionalität des Programmes **`make`** - eine gegebene `.c`-Datei neu übersetzt, wenn die zugehörige `.o`-Datei veraltet ist. Die Dateinamen der Quell- bzw. Zieldatei werden der Funktion dabei in den Parametern `cFile` bzw. `oFile` übergeben.

Ein Übersetzen der .c-Datei ist genau dann notwendig, wenn die Quelldatei eine reguläre Datei ist und die Modifikationszeit der Zieldatei älter ist als die Modifikationszeit der Quelldatei. Sollte keine erneute Übersetzung notwendig sein, soll die Funktion den Wert 0 zurückgeben.

Muss die `.c`-Datei übersetzt werden, so wird in einem Kindprozess der System-Compiler `cc` mit folgendem Aufruf gestartet: `/usr/bin/cc -o oFile -c cFile`

Der Vaterprozess wartet anschließend, bis der Kindprozess terminiert. Hat sich der Kindprozess regulär mit dem Exit-Status `EXIT_SUCCESS` beendet, so gibt die Funktion den Wert 1 zurück.

Tritt bei der Überprüfung der Modifikationszeiten oder beim Erzeugen des Kindprozesses ein Fehler auf oder beendet sich der Kindprozess **nicht** regulär, so gibt **`build`** den Wert -1 zurück. Die Funktion soll keine Meldungen ausgeben, auch keine Fehlermeldungen.

Anmerkungen:

- Die Modifikationszeitstempel sind in Unix-Zeit, d.h. der Anzahl der Sekunden seit dem 01. Januar 1970, 0:00 Uhr angegeben, und können wie reguläre Ganzzahlen miteinander verglichen werden.

// Includes

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


// Funktion build()

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 4 Arbeiter-Threads ihre Arbeit (Ausführung einer Funktion `doWork()`) erledigt haben. (3 Punkte)

Hauptthread:	**Arbeiter-Thread:**
`sem = sem_init(0);`	`void threadFunc(){`
`startWorkerThreads(threadFunc);`	
	`doWork();`
	`}`