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 Prozesszuständen ist richtig? 2 Punkte

- ❏ Pro Prozessor kann es stets nur einen *laufenden*, jedoch mehr als einen *bereiten* Prozess geben.
- ❏ Terminiert ein *laufender* Prozess, so wird er vom Betriebssystem in den Zustand *blockiert* überführt.
- ❏ Ein Prozess kann mit Hilfe von Spezialbefehlen selbst vom Zustand *bereit* in den Zustand *laufend* wechseln.
- ❏ Ein Prozess kann nicht direkt vom Zustand *laufend* in den Zustand *bereit* überführt werden.

b) Welche Aussage zu Semaphoren ist richtig? 2 Punkte

- ❏ Die *P*-Operation eines Semaphors erhöht den Wert des Semaphors um 1 und deblockiert gegebenenfalls wartende Prozesse.
- ❏ Die *V*-Operation eines Semaphors erhöht den Wert des Semaphors um 1 und deblockiert gegebenenfalls wartende Prozesse.
- ❏ Ein Semaphor kann nur zur Signalisierung von Ereignissen, nicht jedoch zum Erreichen gegenseitigen Ausschlusses verwendet werden.
- ❏ Die *V*-Operation eines Semaphors kann ausschließlich von einem Thread aufgerufen werden, der zuvor mindestens eine *P*-Operation auf dem selben Semaphor aufgerufen hat.

c) Welche Aussage zum Thema Speicherverwaltung ist richtig? 2 Punkte

- ❏ Bei der Verwendung segmentierter Speicherverwaltung sind alle Segmente gleich groß.
- ❏ Bei der Verwendung gekachelter Speicherverwaltung (*paging*) sind alle Kacheln gleich groß.
- ❏ Die Einträge in einer Seitentabelle können zur Laufzeit eines Programmes nicht mehr durch das Betriebssystem geändert werden.
- ❏ Bei gekachelter Speicherverwaltung müssen alle Seiten zu jeder Zeit im Hauptspeicher eingelagert sein.

d) Welche Aussage zum Thema Adressräume ist richtig? 2 Punkte

- ❏ Im realen Adressraum sind alle Adressen gültig.
- ❏ Ein Zugriff auf eine nicht abgebildete Adresse führt bei virtuellen Adressräumen immer zur Beendigung des Programms, welches den Zugriff durchführen wollte.
- ❏ Der virtuelle Adressraum kann nie größer sein als der im Rechner vorhandene Hauptspeicher.
- ❏ In einem virtuellen Adressraum sind alle Adressen gültig, es müssen jedoch nicht zu jedem Zeitpunkt alle Daten im Hauptspeicher vorliegen.

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

- ❏ Der Übersetzer (Compiler) erzeugt aus mehreren Programmteilen (Modulen) einen Prozess.
- ❏ Ein Programm kann immer nur von einem Prozess gleichzeitig ausgeführt werden.
- ❏ Ein Prozess kann mit Hilfe von Threads mehrere Programme gleichzeitig ausführen.
- ❏ Ein Prozess ist ein Programm in Ausführung - ein Prozess kann während seiner Lebenszeit aber auch mehrere verschiedene Programme ausführen.

f) Welche Aussage über Variablen in C-Programmen ist richtig? 2 Punkte

- ❏ Lokale *automatic*-Variablen, die auf dem Stack angelegt werden, werden immer mit dem Wert 0 initialisiert.
- ❏ Eine Funktion, die mit dem Schlüsselwort *static* definiert wird, kann nur innerhalb des Moduls aufgerufen werden, in dem sie definiert wurde, nicht jedoch aus einem anderen Modul heraus.
- ❏ Wird dem Parameter einer Funktion innerhalb der Funktion ein neuer Wert zugewiesen, so ändert sich auch der Wert der Variablen, welche in der aufrufenden Funktion als Parameter angegeben wurde.
- ❏ Es ist nicht möglich, Zeiger als Parameter an Funktionen zu übergeben.

g) Welche Aussage zum Thema Adressraumschutz ist richtig? 2 Punkte

- ❏ Beim Einsatz von Segmentierung ist es möglich, dass die selbe logische Adresse in unterschiedlichen logischen Adressräumen auf unterschiedliche physikalische Adresse verweist.
- ❏ Beim Einsatz von Adressraumschutz durch Eingrenzung können keine Zugriffsfehler (*segmentation faults*) auftreten.
- ❏ Bei der Verwendung von Adressraumschutz durch Segmentierung kann durch das Betriebssystem keine dynamische Speicherverwaltung betrieben werden, da die Segmentgrößen unveränderlich sind.
- ❏ Adressraumschutz durch Abteilung erlaubt es, mehrere Benutzerprozesse voneinander zu isolieren.

h) Ein Programm will die drei Zeichenketten 2 Punkte

```
char a[] = "path"; char b[] = "to"; char c[] = "file";
```

mit der Funktion `sprintf(3)` wie folgt in einen Puffer `buffer` speichern:

```
sprintf(buffer, "%s/%s/%s", a, b, c);
```

Mit welcher Länge (in Bytes) muss der Puffer `buffer` mindestens angelegt werden, damit kein Überlauf entstehen kann?

- ❏ 12
- ❏ 13
- ❏ 14
- ❏ 15

i) Welche Aussage zu Prozessen und Threads ist richtig? 2 Punkte

- ❏ Mittels *fork()* erzeugte Kindprozesse können in einem Multiprozessor-System nur auf dem Prozessor ausgeführt werden, auf dem auch der Elternprozess ausgeführt wird.
- ❏ Der Aufruf von *fork()* gibt im Elternprozess die Prozess-ID des Kindprozesses zurück, im Kindprozess hingegen den Wert 0.
- ❏ Threads, die mittels *pthread_create()* erzeugt wurden, besitzen jeweils einen eigenen Adressraum.
- ❏ Die Veränderung von Variablen und Datenstrukturen in einem mittels *fork()* erzeugten Kindprozess beeinflusst auch die Datenstrukturen im Elternprozess.

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? 4 Punkte

- ❍ In einem Verzeichnis darf es keinen Eintrag geben, der auf das Verzeichnis selbst verweist.
- ❍ Die Anzahl der *hard-links*, die auf ein Verzeichnis verweisen, hängt von der Anzahl seiner Unterverzeichnisse ab.
- ❍ Auf eine Datei in einem Dateisystem verweisen immer mindestens zwei *hard-links*.
- ❍ Auf jedes Verzeichnis in einem Dateisystem verweisen immer mindestens zwei *hard-links*.
- ❍ Der Name einer Datei wird in ihrem Dateikopf (*inode*) gespeichert.
- ❍ Wird eine Datei gelöscht, so werden auch alle *symbolic links* gelöscht, die auf diese Datei verweisen.
- ❍ *Hard links* auf Dateien können nur innerhalb des Dateisystems angelegt werden, in dem auch die Datei selbst liegt.
- ❍ Innerhalb eines Verzeichnisses können mehrere Verweise auf den selben *inode* existieren, sofern diese unterschiedliche Namen haben.

Aufgabe 2: chatroom-logger (45 Punkte)

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

Schreiben Sie ein Programm **`chatroom-logger`**, das Nachrichten von mehreren UNIX-Pipes einliest, puffert und gebündelt auf die Standardausgabe ausgibt.

Mehrere Lese-Threads lesen die Nachrichten der Benutzer parallel aus separaten UNIX-Pipes ein und ein Thread (Haupt-Thread) gibt diese gesammelt aus. Zum Austausch der Logmeldungen zwischen den Lesern und dem Schreiber wird eine verkettete Liste mit FIFO-Eigenschaft (Listenelemente: `struct chatLine`) verwaltet. Der Zugriff auf die verkettete Liste ist geeignet zu synchronisieren.

Das Programm soll folgendermaßen arbeiten:

- Der Haupt-Thread sucht im aktuellen Arbeitsverzeichnis nach UNIX-Pipes (`S_ISFIFO`) und startet für jede Pipe einen neuen Lese-Thread, der die Funktion `readThread` ausführt. Danach wartet der Haupt-Thread passiv, bis er von einem Lese-Thread benachrichtigt wird, dass die verkettete Liste genügend Elemente enthält.
  Nun sichert der Haupt-Thread die verkettete Liste und erzeugt eine neue, leere Liste. (Tipp: Es ist nicht notwendig die einzelnen Listenelemente zu kopieren. Es reicht aus den Kopfzeiger geeignet zu verändern).
  Danach gibt er alle in der Liste gepufferten Chatnachrichten aus (Format: "*NameDerPipe*: *Chatnachricht*") und wartet wieder passiv auf weitere Nachrichten.
- Funktion **`void *readThread(void *argument)`**: Öffnet die als Parameter übergebene Pipe zum Lesen und liest zeilenweise (Sie können davon ausgehen, dass Chatnachrichten maximal 4095 Zeichen lang sind). Jede eingelesene Zeile wird zusammen mit dem Dateinamen der UNIX-Pipe in die verkettete Liste eingefügt. Sobald die Liste 20 Elemente umfasst, wird der Haupt-Thread benachrichtigt.

Ihnen steht das aus der Übung bekannte Semaphor-Modul zur Verfügung. Die Schnittstelle finden Sie im folgenden 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.


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

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

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

struct chatLine {
   char source[256]; // Name der Pipe, von der die Zeile stammt
   char line[4097];  // Inhalt der Zeile
   struct chatLine* next;
};

// Makros, Funktionsdeklarationen, globale Variablen
```

```
// Funktion main()
```

```
   // Verzeichnis öffnen und nach Pipes durchsuchen
```


```
   // Threads starten
```

```
   // Auf Chatnachrichten warten, entnehmen und ausgeben
```

```
// Ende Funktion main
```

A:

```
// Funktion readThread()
```

```
  // Chatnachrichten zeilenweise einlesen
```


```
    // gelesene Nachricht in Liste einfuegen
```

```
// Ende Funktion readThread
```

R:

Aufgabe 3: (11 Punkte)

a) Gegegeben sei das unten stehende Programm. Skizzieren Sie den Aufbau des logischen Adressraums eines Prozesses, der dieses Programm ausführt.
   - Welche Segmente werden angelegt? Was enthalten diese Segmente jeweils? (6 Punkte)

b) Zeichnen Sie für jede Variable ein, wo diese ungefähr im logischen Adressraum zu finden sein wird. Wohin zeigen die Zeiger s, p und q (ungefähre Position einzeichnen!)? (5 Punkte)

```
static char *s = "Hello World!";
int glob_a;
static int glob_b = 42;

int main() {
  int a = 0;
  static int b = 2;
  int *p;
  static double *q;

  p = malloc(64 * sizeof(int));
  q = malloc(32 * sizeof(double);

}
```


Aufgabe 4: (12 Punkte)

a) 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). (7 Punkte)

b) Sie haben in der Vorlesung zwei Arten von Programmunterbrechungen kennengelernt. Nennen Sie diese, beschreiben Sie mindestens zwei Eigenschaften, in denen sie sich voneinander unterscheiden und nennen Sie jeweils ein Beispiel. (5 Punkte)