Aufgabe 1.1: Einfachauswahl-Fragen (20 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 RAID ist richtig? 2 Punkte

- ❏ Wenn bei einem RAID-4-System die Parity-Platte ausfällt, sind alle Daten verloren.
- ❏ Bei RAID 5 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.
- ❏ Bei allen RAID-Systemen ist ein höherer Schreib-Durchsatz als bei einer einzelnen Platte möglich, da alle Platten gleichzeitig beauftragt werden können.
- ❏ Bei RAID 0 werden die Datenblöcke über mehrere Festplatten verteilt und repliziert gespeichert.

b) Was versteht man unter der Second-Chance- (oder Clock-) Policy? 2 Punkte

- ❏ Eine Seitenersetzungsstrategie, bei der jeweils die älteste Seite ausgelagert wird.
- ❏ Eine Seitenersetzungsstrategie, die mit Hilfe eines Referenz-Bits eine einfacher zu implementierende Annäherung an LRU realisiert.
- ❏ Eine Scheduling-Strategie, bei der Prozesse vor der Verdrängung eine zweite Chance erhalten.
- ❏ Eine Speicherallokationsstrategie, bei der im Fehlerfall ein zweiter Allokationsversuch stattfindet.

c) Wodurch kann Nebenläufigkeit in einem System entstehen? 2 Punkte

- ❏ Durch Interrupts.
- ❏ Durch nicht-blockierende Synchronisation.
- ❏ Durch langfristiges Scheduling.
- ❏ Durch Threads auf einem Monoprozessorsystem mit kooperativem Scheduling.

d) Der Speicher eines UNIX-Prozesses ist in Text-, Daten- und Stack-(Stapel-)Segment untergliedert. Welche Aussage zur Platzierung von Daten in diesen Segmenten ist richtig? 2 Punkte

- ❏ Lokale `static`-Variablen werden beim Betreten der zugehörigen Funktion initialisiert.
- ❏ Bei einem Aufruf von `malloc(3)` wird das Stack-Segment dynamisch erweitert.
- ❏ Globale Variablen liegen im Daten-Segment.
- ❏ Der Code von Funktionen wird zusammen mit den Variablen der Funktion im Stack-Segment abgelegt.

e) Wodurch kann es zu Seitenflattern kommen? 2 Punkte

- ❏ Wenn sich zu viele Prozesse im Zustand *blockiert* befinden.
- ❏ Wenn bei einer verdrängenden Scheduling-Strategie die Zahl der von den Prozessen aktiv genutzten Seiten die Zahl der verfügbaren Seitenrahmen übersteigt.
- ❏ Durch Programme, die eine Defragmentierung auf der Platte durchführen.
- ❏ Wenn zu viele Prozesse im Rahmen der mittelfristigen Einplaung ausgelagert (*swap-out*) wurden.

f) Man unterscheidet Traps und Interrupts. Welche Aussage ist richtig? 2 Punkte

- ❏ Der Zugriff auf eine logische Adresse kann zu einem Trap führen.
- ❏ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Interrupts immer an den gleichen Stellen auf.
- ❏ Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
- ❏ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.

g) 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.

h) Was versteht man unter virtuellem Speicher?

2 Punkte

- ❏ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.
- ❏ 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.
- ❏ 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)`**).

i) Namensräume dienen u. a. der Organisation von Dateisystemen. Welche Aussage ist richtig?

2 Punkte

- ❏ Flache Namensräume sind besonders einfach implementierbar und damit vor allem für Mehrbenutzersysteme gut geeignet.
- ❏ Der Nachteil von hierarchischen Namensräumen besteht darin, dass das Dateisystem spezielle Funktionen zum Auflösen von Namenskonflikten implementieren muss.
- ❏ Hierarchische Namensräume werden erzeugt, indem man in einem Kontext symbolische Verweise auf Dateien einträgt.
- ❏ In einem hierarchisch organisierten Namensraum dürfen gleiche Namen in unterschiedlichen Kontexten enthalten sein.

j) 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

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

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 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 Threads sind richtig? 4 Punkte

- ❍ Die Umschaltung von leichtgewichten Prozessen muss immer im Systemkern erfolgen.
- ❍ Bei federgewichtigen Prozessen ist die Schedulingstrategie durch das Betriebssystem vorgegeben.
- ❍ Leichtgewichtige Prozesse können Multiprozessoren ausnutzen.
- ❍ Zur Umschaltung von federgewichtigen Prozessen ist ein Adressraumwechsel erforderlich.
- ❍ Zu jedem leichtgewichtigen Prozess gehört ein eigener Adressraum.
- ❍ Federgewichtige Prozesse blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ❍ Leichtgewichtige Prozesse setzen den Einsatz von verdrängenden Scheduling-Verfahren voraus.
- ❍ Zu jedem federgewichtigen Prozess gehört ein eigener, geschützter Adressraum.

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

- ❍ Das Buddy-Verfahren verhindert internen Verschnitt.
- ❍ Die Verschmelzung benachbarter Löcher ist beim Buddy-Verfahren besonders einfach.
- ❍ Beim Buddy-Verfahren können zwei aneinandergrenzende Blöcke gleicher Größe immer 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.
- ❍ Bei allen drei listenbasierten Verfahren (First-Fit, Best-Fit und Worst-Fit) ist externer Verschnitt möglich.
- ❍ Der Systemaufruf `free(2)` sorgt dafür, dass die angegebene Seite im Freiseitenpuffer landet.
- ❍ Bei der Platzierungsstrategie *First-Fit* ist die Verschmelzung von freien Speicherbereichen einfacher als bei *Worst-Fit*.

Aufgabe 2: mops (62 Punkte)

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

a) Schreiben Sie ein Programm `mops` (**M**ulti-**O**utput **P**rint **S**erver), das als Druck-Server eingehende Druckaufträge auf mehrere physische Drucker verteilt.

`mops` ermittelt beim Start die im System verfügbaren Drucker und startet für jeden Drucker einen Arbeiter-Thread, der Aufträge auf diesem Drucker ausgibt. Der Haupt-Thread nimmt auf einem TCP/IPv6-Socket eingehende Verbindungen an und übergibt diese über einen Ringpuffer, der Platz für `BUFFER_SIZE` (= 64) Einträge bietet, an die Arbeiter-Threads.

Das Programm soll folgendermaßen arbeiten:

– Das Programm erhält als Parameter auf der Befehlszeile die Portnummer, auf der Verbindungen angenommen werden sollen.

– Zum Erzeugen der Arbeiter-Threads wird die Funktion

```
unsigned int spawnThreads(void);
```

aufgerufen. Diese sucht im Verzeichnis `/dev` nach Einträgen vom Typ *Character Device*, deren Name mit der Zeichenkette `"lp"` beginnt (also beispielsweise `/dev/lp0` und `/dev/lp1`). Für jede gefundene Drucker-Gerätedatei erzeugt die Funktion einen Arbeiter-Thread mit der Startfunktion `feedPrinter()`. Die Funktion `spawnThreads()` gibt die Anzahl der gefundenen Drucker (= Anzahl der gestarteten Threads) zurück.

– Nach dem Starten der Arbeiter-Threads gibt das Hauptprogramm auf der Standardausgabe die Anzahl der gefundenen Drucker im folgenden Format aus:

```
2 printer devices found
```

Falls im System kein Drucker gefunden werden konnte, wird nach Ausgabe dieser Meldung der Prozess beendet.
Ansonsten erzeugt das Programm einen TCP/IPv6-Socket, auf der es Verbindungen entgegennimmt. Nach jeder Verbindungsannahme wird der dazugehörige Socket-Deskriptor in den Ringpuffer eingetragen. Ist der Puffer bereits voll, soll mit dem Einfügen solange gewartet werden, bis wieder ein freier Platz vorhanden ist.

– Jeder der Arbeiter-Threads startet in der Funktion

```
void *feedPrinter(const char device[]);
```

die als Parameter den Namen der Drucker-Gerätedatei erhält. Der Arbeiter-Thread öffnet das Gerät zum Schreiben im *Append*-Modus (wie eine normale Datei). Anschließend werden in einer Endlosschleife Druckaufträge abgearbeitet. Zur Annahme eines Auftrags wird ein Socket-Deskriptor aus dem Ringpuffer entnommen. Ist der Ringpuffer leer, blockiert die Funktion solange, bis ein Auftrag eintrifft.
Alle Daten, die der Client über die Socket-Verbindung sendet, werden auf den Drucker ausgegeben. Danach wird die Verbindung beendet, womit die Bearbeitung des Druckauftrags abgeschlossen ist.

Synchronisieren Sie den Ringpuffer mit zählenden Semaphoren.

Schnittstelle:

```
SEM *semCreate(int initial_value);
void P(SEM *sem);
void V(SEM *sem);
```

Die Semaphor-Funktionalität ist in ein separates Modul `sem.c` ausgegliedert, das ebenfalls von Ihnen zu implementieren ist. Die Schnittstelle ist in der existierenden Datei `sem.h` deklariert.

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 Anweisungen 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.


```
/*************************************************************
 * Datei mops.c
 *************************************************************/
#include <dirent.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/stat.h>


// weitere Includes, Konstanten
// globale Variablen, Funktionsdeklarationen usw.
```

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

I:

```
// Funktion main()

    // Befehlszeilenargumente pruefen und
    // weitere allgemeine Vorbereitungen

    // Druckergeraete suchen und Threads starten
```



```
    // Socket erstellen und auf Verbindungsannahme vorbereiten

    //  Verbindungen annehmen und in den Ringpuffer einfuegen

// Ende Funktion main
```

A:

```
// Funktion  spawnThreads
```

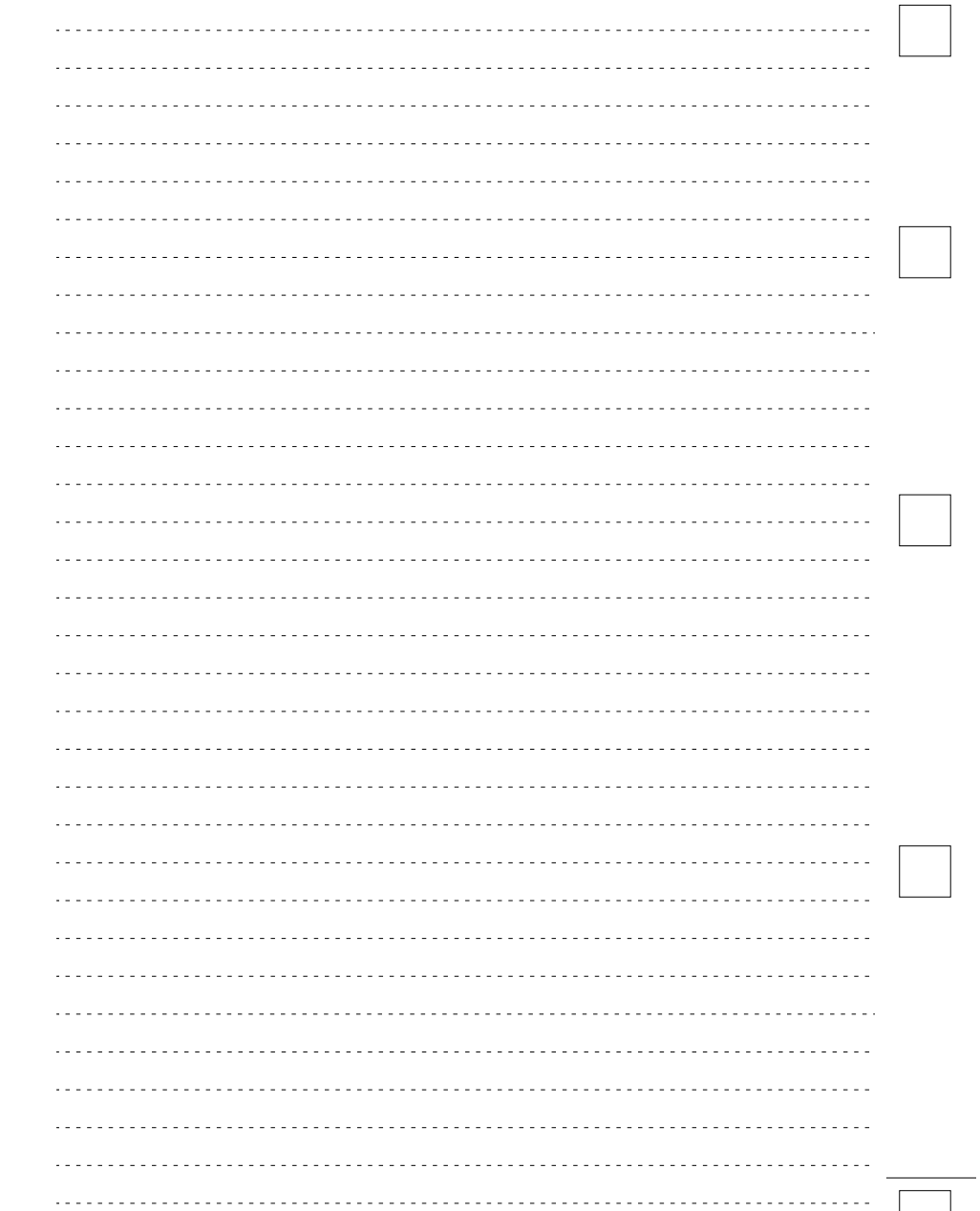
```
 // Drucker-Gerätedateien suchen
```


```
// jeweils Arbeiter-Thread dazu erzeugen
```

```
// Ende Funktion  spawnThreads
```

T:

```
// Funktion feedPrinter
```



```
// Ende Funktion feedPrinter
```

F:


```
/*************************************************************
 * Datei sem.c
 *************************************************************/
```

```
/* Funktion semCreate */
```

```
/* Funktion P */
```

```
/* Funktion V */
```

S:

b) Schreiben Sie ein Makefile, das die beiden Programm-Module `mops.c` und `sem.c` (unter Berücksichtigung der Header-Datei `sem.h`) übersetzt und zu einer ausführbaren Datei `mops` zusammenbindet. Das Makefile soll die beiden Standard-Pseudo-Targets `all` und `clean` enthalten.

M:

Aufgabe 3: (20 Punkte)

a) Beschreiben Sie die Prozesszustände bei kurz- und mittelfristiger Einplanung sowie die Ereignisse, die jeweils zu den Zustandsübergängen führen (Skizze mit kurzer Erläuterung der Zustände und Übergänge).

b) Wann erfolgt hierbei eine Einplanung (Scheduling) oder eine Umplanung (Rescheduling)?

c) Man kann Scheduling-Verfahren nach verschiedenen Kriterien in Kategorien einteilen. Beschreiben Sie für die Kategorisierung *kooperatives vs. präemptives Scheduling* jeweils die Eigenschaften entsprechender Scheduling-Strategien, nennen Sie konkrete Beispiele für solche Strategien und geben Sie an, in welchen Anwendungssituationen entsprechende Strategien sinnvoll eingesetzt werden.

d) Welche der in Teilaufgabe a) beschriebenen Zustandsübergänge sind durch den Unterschied zwischen kooperativem und präemptiven Scheduling betroffen und durch welche Ereignisse werden diese Zustandsübergänge dann jeweils ausgelöst?

Aufgabe 4: (10 Punkte)

Bei virtuellen Adressräumen können Teile des Speichers auf Hintergrundspeicher ausgelagert sein.

a) Wie und durch wen wird im System bei einem Speicherzugriff erkannt, dass der entsprechende Speicherbereich ausgelagert ist?

b) Was läuft im System nach dieser Erkennung ab? Beschreiben Sie die einzelnen Schritte, die im Betriebssystem abgewickelt werden, um den Speicher verfügbar zu machen. Geben Sie an, welche Prozesszustände der auslösende Prozess dabei jeweils einnimmt.