**Aufgabe 1: (20 Punkte)**

Bei den Multiple-Choice-Fragen ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-Antwort korrigieren, kreisen sie bitte die falsche Antwort ein und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Wie groß ist typischerweise eine Seite bei Seitenadressierung? 1 Punkt

- ❏ 8 bis 16 Bytes
- ❏ 512 bis 8.192 Bytes
- ❏ 32.768 bis 8.388.608 Bytes

b) Welche Antwort stellt **kein** Attribut einer Datei eines UNIX-UFS-Dateisystems dar? 1 Punkt

- ❏ Zeitpunkt der letzten Änderung
- ❏ Gruppenzugehörigkeit
- ❏ Zugriffsrechte
- ❏ Dateiendung (z.B. .html oder .jpeg)

c) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (Symbolic Links) und feste Links (Hard Links) auf Dateien. Welche Aussage ist **falsch**? 2 Punkte

- ❏ Ein Symbolic Link kann auf Verzeichnisse verweisen.
- ❏ Ein Hard Link kann nur auf Dateien, jedoch nicht auf Verzeichnisse verweisen.
- ❏ Hard Links können nur vom Systemadministrator angelegt werden.
- ❏ Symbolic Links können existieren, obwohl die verwiesene Datei oder das verwiesene Verzeichnis bereits gelöscht wurde.





d) Welche Aussage bezüglich des Systemaufrufs `accept()` ist richtig? 2 Punkte

- ❏ Der Systemaufruf nimmt eine Verbindung aus einer Warteschlange von neuen Verbindungen und bekommt für diese neue Verbindung eine neue Portnummer.
- ❏ Ein `accept()` Systemaufruf kann nicht blockieren, da durch den Aufruf von `listen()` sichergestellt wurde, dass immer eine Verbindungsanfrage vorliegt.
- ❏ Der `accept()` Systemaufruf nimmt eine Verbindung aus einer mit `listen()` eingerichteten Warteschlange und erzeugt für diese Verbindung einen neuen Socket.
- ❏ Der Systemaufruf `accept()` teilt dem Betriebssystem mit, dass der Prozess bereit ist Verbindungen entgegen zu nehmen.

e) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux-Filedeskriptors zu? 2 Punkte

- ❏ Ein Filedeskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei, ein Gerät, einen Socket oder eine Pipe benutzen kann.
- ❏ Filedeskriptoren sind Zeiger auf Betriebssystemstrukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ❏ Ein Filedeskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann, und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.
- ❏ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Filedeskriptor zum Zugriff zurück.

f) Ein *laufender* Prozess wird in den Zustand *blockiert* überführt. Welche Aussage passt **nicht** zu diesem Vorgang? 2 Punkte

- ❏ Der Prozess wartet auf Daten von der Festplatte.
- ❏ Der Prozess wartet mit dem Systemaufruf `wait()` auf die Terminierung eines Sohn-Prozesses.
- ❏ Der Prozess hat einen Seitenfehler für eine Seite, die bereits in die Freiliste der Seiten eingetragen aber noch im Hauptspeicher vorhanden ist.
- ❏ Der Prozess wartet auf eine Tastatureingabe.

g) Welche Aussage ist bezüglich der Seitenersetzungsstrategie Second-Chance richtig? 2 Punkte

- ❏ Die Second-Chance-Strategie nähert sich der LRU-Strategie an, wenn alle Seiten sehr häufig benutzt werden.
- ❏ Die Second-Chance-Strategie zeigt keine FIFO-Anomalie.
- ❏ Die Second-Chance-Strategie nähert sich nahezu der optimalen $B_O$-Strategie an und wird daher in fast allen realen Systemen benutzt.
- ❏ Die Second-Chance-Strategie benötigt ein Referenzbit in jedem Eintrag der Seitenkacheltabelle.

h) Für welchen Zweck wird der Systemaufruf `shmat()` benutzt? 2 Punkte

- ❏ Mit dem Systemaufruf `shmat()` kann ein Prozess einen Teil seines Speichers exportieren, sodass andere Prozesse darauf zugreifen können.
- ❏ Der Systemaufruf `shmat()` blendet ein bestehendes Shared-Memory-Segment in den logischen Adressraum des aufrufenden Prozesses ein.
- ❏ Der Aufruf von `shmat()` erzeugt ein neues Speichersegment, auf das verschiedene Prozesse gleichberechtigt zugreifen können.
- ❏ Damit zwei Prozesse auf einen gemeinsam genutzten Speicherbereich zugreifen können müssen diese zuerst vom Betriebssysteme jeweils einen Semaphor anfordern. Mit der Hilfe des Semaphors und dem Systemaufruf `shmat()` können die Prozesse anschliessend auf den gemeinsamen Speicher zugreifen.

i) Sie kennen den Translation-Look-Aside-Buffer (TLB). Welche Aussage ist richtig? 2 Punkte

- ❏ Der TLB verkürzt die Zugriffszeit auf den physikalischen Speicher, da ein Teil des möglichen Speichers in einem sehr schnellen Pufferspeicher vorgehalten wird.
- ❏ Der TLB puffert Daten bei der Ein-, Ausgabebehandlung und beschleunigt diese damit.
- ❏ Verändert sich die Speicherabbildung von logische auf physikalische Addressen aufgrund einer Adressraumumschaltung, so werden auch die Daten im TLB ungültig.
- ❏ Wird eine Speicherabbildung im TLB nicht gefunden, wird der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (Segmentation Fault) abgebrochen.





j) Welche Aussage über UNIX-Semaphoren ist richtig? 3 Punkte

- ❏ UNIX-Semaphoren können unmittelbar das Verhalten von PV-Chunk- und UP-DOWN-Systemen nachbilden.
- ❏ Die Operation `semop()` kann sich auf maximal zwei UNIX-Semaphoren gleichzeitig beziehen.
- ❏ UNIX-Semaphoren können das Verhalten von PV-Multiple-Semaphoren nachbilden.
- ❏ UNIX-Semaphoren sind nur für die Koordinierung von Speicherzugriffen geeignet, nicht jedoch für die Koordinierung von Aktivitätsträgern in mehreren Prozessen.

k) Alle aufgeführten Koordinierungsmittel erlauben die Implementierung eines kritischen Abschnitts. Welches der aufgeführten Verfahren ist jedoch ungeeignet, weil die Prozesse aktiv warten müssen? 1 Punkt

- ❏ binärer Semaphor
- ❏ Algorithmus von Peterson
- ❏ Up-Down-Systeme
- ❏ Sperren von Unterbrechungen

**Aufgabe 2: (40 Punkte)**

a) Schreiben Sie ein Programm **`rlsd`**, das als Server-Prozess Anfragen nach Directory-Inhalten über TCP/IP beantwortet.

Das **`rlsd`**-Programm erhält als Argument die Nummer des Ports auf dem Verbindungen angenommen werden sollen.

Das Programm soll folgendermaßen arbeiten:

- Es nimmt Verbindungen über beliebige IP-Adressen auf dem angegeben Port an.
- Es wird jeweils eine Verbindung angenommen und in einem Sohnprozess bearbeitet während der Vaterprozess für die nächste Verbindung wieder bereitsteht.
- Der Client sendet über die Verbindung den Namen des aufzulistenden Directories.
- Der Server gibt die Namen aller Dateien, die in dem angefordeten Directory angelegt sind aus (unsortiert, jeweils ein Dateiname in einer Zeile) — dieser Teil der Aufgabe wird in einer Funktion mit Namen "ls" bearbeitet.
- Der bearbeitende Sohnprozess terminiert anschließend, der Vaterprozess braucht sich in der Lösung nicht um terminerende Sohnprozesse kümmern.

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.





```
/* includes */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>

/* Funktionsdeklarationen, globale Variablen */
```

```
/* Funktion main */
```

```
{
  /* lokale Variablen und was man sonst am Anfang so braucht */
```

**A:**

```
/* Socket oeffnen */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
/* Socket an angegebenen Port
   und beliebige IP-Adressen binden */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
/* Warteschlange ankommender Verbindungen
    auf 5 einstellen */
```

.....

.....

.....

.....

.....

**S:**





```
/* Server-Schleife */
```

.....

.....

.....

```
   /* Verbindung annehmen */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
   /* Prozess zur Kommandoausführung erzeugen */
```

.....

.....

.....

.....

```
   /* Fehler bei Prozesserzeugung */
```

.....

.....

.....

.....

.....

**S:**
**P:**

```
    /* Sohnprozess */
```

3P

2P

```
    /* Vaterprozess */
```

1S

```
} /* Ende Funktion main */
```

**P: S:**





```
    /* ls-Funktion */
```

d

o

r

a

**L:**

b) Schreiben Sie ein Makefile zum Erzeugen des rlsd-Programms.

Ein Aufruf von

```
make rlsd
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

soll das **`rlsd`**-Programm und evtl. erzeugte **`.o`**-Dateien entfernen.