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) Welcher UNIX-Systemaufruf wird bei der Verwendung von Sockets auf keinen Fall gebraucht? 1 Punkt

- ❑ `shutdown()`
- ❑ `mmap()`
- ❑ `select()`
- ❑ `accept()`

b) Welche Aussage bezüglich der Freispeicherverwaltung mittels einer Bitliste ist **falsch**? 2 Punkte

- ❑ Der zu verwaltende Speicher wird in Speichereinheiten gleicher Größe unterteilt.
- ❑ Zur Suche nach freiem Speicher kann es nötig sein, die gesamte Bitliste zu durchsuchen.
- ❑ Das Zusammenfassen von benachbarten freien Speichereinheiten ist besonderst aufwendig.
- ❑ Je kleiner die Speichereinheiten sind, desto länger ist die Bitliste.

c) Welches Attribut ist **nicht** im Inode eines UNIX-Dateisystems verzeichnet. 1 Punkt

- ❑ Dateityp (Verzeichnis, normale Datei, Spezialdatei)
- ❑ Eigentümer
- ❑ Dateiname
- ❑ Zeitpunkt des letzten Dateizugriffes

d) Wann werden die Daten bei einem System mit Block-Buffer-Cache auf die Festplatte geschrieben? 2 Punkte

- ❑ Wenn die Datei neu geöffnet wird
- ❑ Nach jedem Schreibaufruf im Modus O_NONBLOCK
- ❑ Beim Systemaufruf `select()`
- ❑ Wenn die Datei geschlossen wird

e) Ein Prozess wird in den Zustand *bereit* überführt. Welche Aussage passt zu diesem Vorgang? 2 Punkte

- ❑ Ein anderer Prozess blockiert sich an einem Semaphor.
- ❑ Der Prozess hat auf Daten von der Festplatte gewartet und die Daten stehen nun zur Verfügung.
- ❑ Der Prozess hat einen Seitenfehler für eine Seite, die aber noch im Hauptspeicher vorhanden ist.
- ❑ Der Prozess wartet auf eine Tastatureingabe.

f) Welche Aussage ist bezüglich der Seitenersetzungsstrategie B0 richtig? 2 Punkte

- ❑ Die B0-Strategie nähert sich der FIFO-Strategie an, wenn alle Seiten sehr häufig benutzt werden.
- ❑ Die B0-Strategie zeigt keine FIFO-Anomalie.
- ❑ Die B0-Strategie ist die optimale Strategie zum Ersetzen von Seiten und wird daher in fast allen realen Systemen benutzt.
- ❑ Die B0-Strategie benötigt einen Referenzzähler in jedem Eintrag der Seiten-Kachel-Tabelle.

g) Für welchen Zweck wird der Systemaufruf `mmap()` benutzt? 2 Punkte

- ❏ Mit dem Systemaufruf `mmap()` kann ein Prozess einen Teil seines mit `malloc()` angeforderten Speichers exportieren, sodass andere Prozesse darauf zugreifen können.
- ❏ Mit dem Systemaufruf `mmap()` kann der Inhalt einer Datei in den logischen Adressraum des aufrufenden Prozesses eingeblendet werden.
- ❏ Der Aufruf von `mmap()` bildet einen Teil des Hauptspeichers auf einen Filedescriptor ab. Der Filedescriptor muss zuvor mit `open()` bzw. `connect()` geöffnet werden.
- ❏ Der Systemaufruf `mmap()` bildet einen im Hauptspeicher liegenden Socket auf den DNS-Namen des Servers ab.

h) Was versteht man unter Virtuellem Speicher? 2 Punkte

- ❏ Adressierbarer Speicher in dem sich keine Daten speichern lassen, weil er physikalisch nicht vorhanden ist.
- ❏ 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.
- ❏ Einen logischen Adressraum.
- ❏ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.

i) Was versteht man unter dem zweiten Leser-Schreiber-Problem? 2 Punkte

- ❏ Mehrere Prozesse greifen lesend und schreibend auf gemeinsame Datenstrukturen zu. Schreiber und Leser sollen fair behandelt werden.
- ❏ Ein Prozess greift lesend und schreibend auf geheime Datenstrukturen zu. Das Schreiben soll dabei bevorzugt werden.
- ❏ Mehrere Leser und Schreiber greifen gleichzeitig lesend und schreibend auf gemeinsame Datenstrukturen zu. Die Schreiber sollen dabei bevorzugt werden.
- ❏ Mehrere Prozesse greifen lesend und schreibend auf gemeinsame Datenstrukturen zu. Leser sollen bevorzugt werden.

j) Welche Aussage über aktiv wartende Koordinierungsmittel ist **falsch**? 2 Punkt

- ❏ Auf einem Monoprozessorsystem ist das aktive Warten auf die Freigabe eines Betriebsmittels schlecht, da ein anderer Prozess die Zeit sinnvoll nutzen könnte.
- ❏ Der Algorithmus von Peterson ist ein Koordinierungsmittel, welches aktiv auf die Freigabe von Betriebsmitteln wartet.
- ❏ Koordinierungsmittel die aktiv warten sind nur sehr aufwändig zu realisieren, da eine zusätzliche Warteschlange für alle Prozesse benötig wird.
- ❏ Auf Systemen mit mehreren Prozessoren sind Programme mit aktiv wartenden Koordinierungsmitteln häufig schneller.

k) Sie kennen den Begriff Demand-Paging. Welche Aussage dazu ist richtig? 2 Punkte

- ❏ Demand-Paging setzt eine segmentierte Speicherverwaltung voraus.
- ❏ Demand-Paging lädt eine Seite erst dann in den Hauptspeicher, wenn sie tatsächlich angesprochen wird. Nicht benutzte Seiten werden unter Umständen aus dem Hauptspeicher ausgelagert.
- ❏ Demand-Paging benötigt keinerlei Hardware-Unterstützung, da sich alle benötigten Mechanismen auch ohne MMU realisieren lassen.
- ❏ Demand-Paging erlaubt es, größere logische Adressräume anzulegen, als Hauptspeicher vorhanden ist. Allerdings muss vorausgesetzt werden, dass ein Prozess nicht alle Seiten des logischen Adressraums tatsächlich anspricht.

Aufgabe 2: (60 Punkte)

a) Schreiben Sie ein Programm `rshd` (Remote-Shell-Daemon) das Kommandos, die über eine Netzwerkverbindung übergeben werden, ausführt.

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

Das Programm soll folgendermaßen arbeiten:

- Es nimmt Verbindungen von beliebigen 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 auszuführenden Kommandos (einfaches Kommando ohne Argumente).
- Auf der Serverseite wird das angegebene Kommando in dem Sohnprozess ausgeführt (das Kommando soll über die PATH-Variable gesucht werden).
- Der Programmteil der sich um die Sohnprozesserzeugung und Kommandoausführung kümmert, ist in einer eigenen Funktion namens `execute_job` zu programmieren. Die Funktion erhält als Parameter den socket-Deskriptor der Verbindung zum Client.
- Die Standard-E/A-Kanäle (stdin, stdout und stderr) für die Kommanoausführhung werden auf die Socket-Verbindung umgeleitet. Diese Umleitung brauchen Sie nicht selbst zu programmieren - gehen Sie statt dessen davon aus, dass Ihnen eine entsprechende Funktion `void setstdio(int socket)` vorcompiliert in einer Datei `setstdio.o` zur Verfügung steht.
- Am Ende der Kommandoausführung terminiert der Sohnprozess. Der `rshd`-Prozess erfährt dies durch ein SIGCHLD-Signal, fragt daraufhin den Exit-Status des Sohn-Prozesses ab und trägt eine Zeile mit PID und exit-Status des Sohnprozesses am Ende einer Datei namens `logfile` ein.

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 - Sie können aber nicht davon ausgehen, dass Sie ausschließlich nur diese Funktionen benötigen.


```
/* includes */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.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:


```
/* Signal-Handler für SIGCHLD einrichten */
```

i

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

a

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

s

```
    /* Kommando ausführen */
```

a

s

A:
I:
S:

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

```
/* execute_job-Funktion */
```

a

```
/* lokale Variablen */
{
```

```
    /* Kommando von Socket einlesen */
```

a

```
    /* Sohnprozess erzeugen und Kommando ausführen */
```

p

```
}
```

A:
P:


```
/* signal-handler für SIGCHLD*/
```

i

```
/* lokale Variablen */
{
```

```
    /* logfile öffnen */
```

i

```
    /* terminierte Sohnprozesse abfragen und Status
       in logfile schreiben */
```

p

i

```
}
```

I:
P:

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

Ein Aufruf von

```
make rshd
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

soll das **`rshd`**-Programm und evtl. bei vorherigen make-Läufen erzeugte **`.o`**-Dateien entfernen.