Aufgabe 1: (30 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) Welche Aussage bezüglich der Freispeicherverwaltung mittels einer Bitliste ist **falsch**? 2 Punkte

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

b) In welcher der folgenden Situationen wird **kein** Trap ausgelöst? 2 Punkte

- ❏ Beim Aufruf einer dynamisch hinzugebundenen Funktion aus einer "shared library".
- ❏ Wenn bei der Ausführung eines Anwendungsprogramms auf eine ungültige Adresse zugegriffen wird.
- ❏ Bei einem Systemaufruf.
- ❏ Bei rekursiven Funktionsaufrufen, wenn das Ende des bereitgestellten Stacks erreicht ist.

c) In einem Segmentdeskriptor werden verschiedene Informationen über ein Segment eines logischen Adressraums gehalten. Was gehört sicher **nicht** dazu? 2 Punkte

- ❏ Die physikalische Adresse des Segmentanfangs im Hauptspeicher.
- ❏ Zugriffsrechte (z. B. lesen, schreiben, ausführen).
- ❏ Die Benutzer-Ids, für die diese Zugriffsrechte gelten.
- ❏ Die Länge des Segments.

d) Wodurch kann es zu Seitenflattern kommen? 2 Punkte

- ❏ Wenn bei der Ersetzungsstrategie Second Chance (SC) bei einem Zugriff auf eine Seite das Referenzbit gesetzt und die Seite danach längere Zeit nicht angesprochen wird, so dass das Bit wieder gelöscht wird. Wenn sich diese Situation mehrfach wiederholt, so dass das Bit ständig den Wert ändert, spricht man von Seitenflattern.
- ❏ Durch Programme, die eine Defragmentierung auf der Platte durchführen.
- ❏ Wenn sich zu viele Prozesse im Zustand blockiert befinden.
- ❏ Wenn bei einer nicht-verdrängenden Scheduling-Strategie die Zahl der von den Prozessen aktiv genutzten Seiten die Zahl der verfügbaren Seitenrahmen übersteigt.

e) Welches der folgenden Verfahren trägt in der Praxis am besten dazu bei, Seitenfehler und ihre Auswirkungen zu minimieren? 2 Punkte

- ❏ Man ermittelt, welche der Seiten eines Prozesses in Zukunft am längsten nicht angesprochen wird und lagert genau diese aus (OPT-Strategie).
- ❏ Man lagert regelmäßig länger nicht genutzte Seiten aus und trägt sie in einem Freiseitenpuffer ein.
- ❏ Man übergibt Prozesse, die einen Seitenfehler verursachen der mittelfristigen Prozesseinplanung, damit sie in nächster Zeit nicht wieder aktiv werden.
- ❏ Man setzt eine Segmentierung in Kombination mit Seitenadressierung ein.

f) Wie wird erkannt, dass eine Seite eines virtuellen Adressraums gerade ausgelagert ist? 2 Punkte

- ❏ Im Seitendeskriptor wird ein spezielles Bit geführt, das der MMU zeigt, ob eine Seite eingelagert ist oder nicht. Falls die Seite nicht eingelagert ist, löst die MMU einen Trap aus.
- ❏ Bei Programmen, die in virtuellen Adressräumen ausgeführt werden sollen, erzeugt der Compiler speziellen Code, der vor Betreten einer Seite die Anwesenheit überprüft und ggf. die Einlagerung veranlasst.
- ❏ Die MMU erkennt bei der Adressumsetzung, dass die physikalische Adresse ungültig ist und löst einen Trap aus.
- ❏ Das Betriebssystem erkennt die ungültige Adresse vor Ausführung eines Maschinenbefehls und lagert die Seite zuerst ein bevor ein Fehler passiert.

g) 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.
- ❏ Flache Namensräume erlauben pro Benutzer nur einen Kontext.
- ❏ 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.

h) Nehmen Sie an, der Ihnen bekannte Systemaufruf stat(2) wäre analog zu der Funktion readdir(3) mit folgender Schnittstelle implementiert: 3 Punkte

```
struct stat *stat(const char *path);
```

Welche Aussage ist **richtig**?

- ❏ Der Systemaufruf liefert einen Zeiger zurück, über den die aufrufende Funktion direkt auf eine Datenstruktur zugreifen kann, die die Dateiattribute enthält.
- ❏ Solch eine Schnittstelle ist nicht schön, da dadurch die aufrufende Funktion auf internen Speicher des Betriebssystems zugreifen könnte.
- ❏ 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.

i) In welcher der folgenden Situationen wird ein *laufender* Prozess in den Zustand *blockiert* überführt? 3 Punkte

- ❏ Der Prozess hat einen Seitenfehler für eine Seite, die bereits in den Freiseitenpuffer eingetragen, aber noch im Hauptspeicher vorhanden ist.
- ❏ Ein Kindprozess des Prozesses terminiert.
- ❏ Der Prozess greift lesend auf eine Datei zu und der entsprechende Datenblock ist noch nicht im Hauptspeicher vorhanden.
- ❏ Der Prozess ruft eine V-Operation auf einen Semaphor auf und der Semaphor hat gerade den Wert 0.

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

- ❏ Das Programm ist der statische Teil (Rechte, Speicher, etc.), der Prozess der aktive Teil (Programmzähler, Register, Stack).
- ❏ Wenn ein Programm nur einen aktiven Ablauf enthält, nennt man diesen Prozess, enthält das Programm mehrere Abläufe, nennt man diese Threads.
- ❏ Ein Prozess ist ein Programm in Ausführung - ein Prozess kann aber auch mehrere verschiedene Programme ausführen.
- ❏ Ein Programm kann immer nur von einem Prozess ausgeführt werden.

k) Welche der folgenden Aussagen über Schedulingverfahren ist **richtig**? 2 Punkte

- ❏ Preemptives Scheduling ist für Mehrbenutzerbetrieb geeignet.
- ❏ Kooperatives Scheduling ist für Steuerungssysteme mit Echtzeitanforderungen völlig ungeeignet.
- ❏ Bei kooperativem Scheduling sind Prozessumschaltungen unmöglich wenn ein Prozess in einer Endlosschleife läuft, selbst wenn er bei jedem Schleifendurchlauf einen Systemaufruf macht.
- ❏ Bei preemptivem Scheduling sind Prozessumschaltungen unmöglich, wenn ein Prozess in einer Endlosschleife läuft.

l) Wozu dient der Maschinenbefehl cas (compare-and-swap)? 2 Punkte

- ❏ Um bei Monoprozessorsystemen Interrupts zu sperren.
- ❏ Um auf einem Multiprozessorsystem einfache Modifikationen an Variablen ohne Sperren implementieren zu können.
- ❏ Um bei der Implementierung von Schlossvariablen (Locks) aktives Warten zu vermeiden.
- ❏ Zur Realisierung einer effizienten Verdrängungssteuerung bei einseitiger Symchronisation.

m) Wozu benötigt man Bedingungsvariablen (condition variables)? 2 Punkte

- ❑ Bei if-Abfragen in C-Programmen.
- ❑ Um in einem kritischen Abschnitt auf ein Ereignis zu warten und den kritischen Abschnitt während der Wartezeit freizugeben.
- ❑ Zur Vermeidung von aktivem Warten in kritischen Abschnitten.
- ❑ Zur Erkennung von Verklemmungen.

n) Welche Problematik wird durch das Philosophenproblem beschrieben? 2 Punkte

- ❑ Ein Erzeuger und ein Verbraucher greifen gleichzeitig auf gemeinsame Datenstrukturen zu.
- ❑ Mehrere Prozesse greifen lesend und schreibend auf gemeinsame Datenstrukturen zu.
- ❑ Exklusive Bearbeitung durch mehrere Bearbeitungsstationen.
- ❑ Gleichzeitiges Belegen mehrerer Betriebsmittel durch einen Prozess.

Aufgabe 2: (60 Punkte)

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

a) Schreiben Sie ein Programm `ftpd` (File-Transfer-Protocol-Daemon), das als Server die Übertragung von Dateien und das Anzeigen von Directory-Inhalten über TCP/IP-Verbindungen unterstützt.

Wird `ftpd` ohne Parameter aufgerufen, werden beliebig viele Verbindungen gleichzeitig bearbeitet. Bei Aufruf `ftpd` *`n`* wird die Zahl der zu einem Zeitpunkt aktiven Verbindungen auf *`n`* beschränkt.

Das Programm soll folgendermaßen arbeiten:

- `ftpd` nimmt auf Port 21 Verbindungen von beliebigen Adressen entgegen. Es sollen maximal 7 Verbindungen anstehen können.
- Es wird jeweils eine Verbindung angenommen und in einem Kindprozess bearbeitet (die Bearbeitung erfolgt in der Funktion `void serve(int socket);`), der Vaterprozess steht sofort für die nächste Verbindung wieder bereit.
- Ist die Zahl der Verbindungen beschränkt, so nimmt `ftpd` nach Erreichen der maximalen Zahl zwar weiterhin Verbindungen an, erzeugt aber keinen Kindprozess, sondern gibt nur die Meldung "Server überlastet" auf die Verbindung aus und bricht sie sofort wieder ab.
- Ein Client sendet über die Verbindung ftpd-Kommandos. Die Funktion `serve` nimmt die Kommandos entgegen (maximale Länge eines Kommandos 1023 Zeichen) und delegiert die Bearbeitung an andere Funktionen. Die Ausgabe der Kommandos (und Fehlermeldungen) soll jeweils auf den Socket erfolgen.
- Implementieren Sie folgende Kommandos:
  `QUIT` beendet die Verbindung
  `RETR` *`datei`* liefert den Inhalt der Datei
  `LIST` *`directory`* liefert eine Liste der Dateien in dem angegebenen Directory (Datei- und Directorynamen enthalten keine Leerzeichen).
  Die Bearbeitung erfolgt in den Funktionen
  `void retr(FILE *socket, char *dateiname);` und
  `void list(FILE *socket, char *dirname);`
- Um die Zahl der Verbindungen beschränken zu können, müssen Sie die aktiven Verbindungen bzw. Kindprozesse in geeigneter Form mitzählen. Das Terminieren von Kindprozessen soll dabei jeweils unmittelbar bearbeitet werden.

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 ggf. auch weitere Funktionen benötigen.

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

/* Funktionsdeklarationen, globale Variablen */
void childhandler(int);
void list(FILE *, const char *);
void retr(FILE *, const char *);
void serve(int);
```

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

```
{
   /* Variablendefinitionen */
```

A:


```
/* Argumente prüfen */
```

```
/* Signale vorbereiten */
```

A:
S:

```
/* Server-Socket oeffnen
   und Verbindungsannahme vorbereiten */
```

O:


```
/* Server-Schleife:
   Verbindungen annehmen und bearbeiten */
```

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

O:

```
/* Auftragsbearbeitung: Funktion serve */
void serve(int socket)
{
    /* Variablendefinitionen, Vorbereitungen */
```

```
    /* Kommandos lesen und bearbeiten */
```

V:

```
} /* Ende Funktion serve */
```



```
/* Kommandobearbeitung: Funktion retr */
void retr(FILE *sockfp, const char *filename)
{
```

R:

```
} /* Ende Funktion retr */
```

```
/* Kommandobearbeitung: Funktion list */
void list(FILE *sockfp, const char *dirname)
{
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
} /* Ende Funktion list */

/* Signalbehandlung für SIGCHLD */
void childhandler(int s)
{
```

.....

.....

.....

.....

.....

.....

.....

.....

```
} /* Ende Funktion childhandler */
```

L:
S:

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

Ein Aufruf von

```
make ftpd
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

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

.....

.....

.....

.....

.....

.....

.....

.....

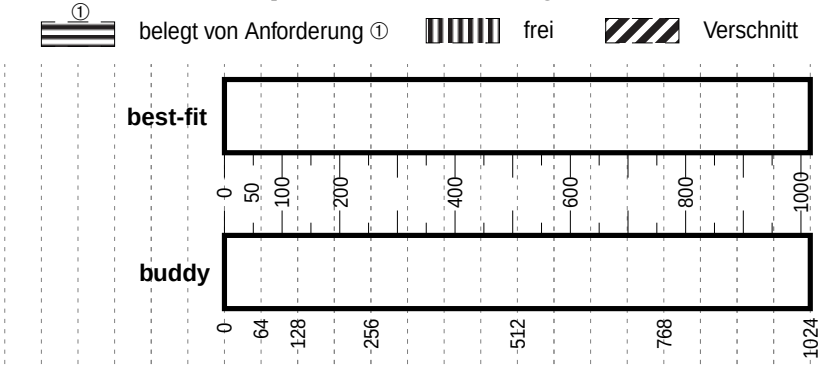
M:

Aufgabe 3: (23 Punkte)

Zur Verwaltung von freiem Speicher (z. B. in Funktionen wie malloc und free) gibt es verschiedenen Strategien bei der Speicherzuteilung. Beschreiben Sie für die beiden Verfahren *best-fit* und *buddy* jeweils

a) welche Datenstrukturen benötigen Sie zur Verwaltung des freien Speichers (der Löcher) und wie organisieren Sie darin die Löcher.

b) Nehmen Sie einen Speicher von 1024 Byte an und skizzieren Sie für beide Verfahren, wie der Speicher nach Anforderungen von
① 100 Byte, ② 200, ③ 50, ④ 50 , ⑤ 200 , ⑥ 50 , ⑦ 100 Byte
aussieht. Welche Bereiche wurden für welche Anforderung vergeben, welche Bereiche sind frei, welche Bereiche sind Verschnitt?
Skizzieren Sie nur den Speicher, nicht die Verwaltungsstrukturen!



c) Skizzieren Sie den Aufbau der Strukturen zur Verwaltung der Löcher nachdem anschließend die Bereiche ②, ⑥, ④, ① freigegeben wurden.

d) Beschreiben Sie kurz (in Stichworten) die Unterschiede zwischen best-fit, worst-fit und first-fit bezüglich Suchaufwand und Aufwand beim Freigeben.

Aufgabe 4: (7 Punkte)

Skizzieren Sie (graphisch) die Abbildung von einer logischen Adresse in eine physikalische Adresse in einem seitennummerierten Adressraum.

Tragen Sie konkret die Abbildung für die logische Adresse 0x1267 ein, wobei die Basisadresse des Seitenrahmens der ersten Seite 0x01502000 sein soll (Seitengröße 4096 Byte).