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 Aussagen treffen für die folgenden Dateisystemimplementierungen zu? 3 Punkte

- ❏ Journalled-Dateisysteme benötigen keine zusätzliche Operation beim Hochfahren des Systems, da ihre Daten immer konsistent sind.
- ❏ Das Prüfen der Konsistenz eines Dateisystems beim Hochfahren kann unterbleiben, wenn das System vor dem Herunterfahren die Konsistenz erkannt und im Dateisystem markiert hat.
- ❏ Die Prüfoperation beim Hochfahren eines UNIX-Systems kann auf einem typischen UNIX-Dateisystem (z.B. UFS) inkonsistent Zustände erkennen und beseitigen. Dabei können keine Daten verloren gehen.
- ❏ Bei einem Journalled-Dateisysteme muss zusätzlich zur normalen Konsistenzprüfung auch noch das Journal eingespielt werden, dieses verhindert zwar Datenverluste, verlangsamt jedoch das Hochfahren des Systems.

b) Welche Aussage zur Verklemmungsverhinderung ist richtig? 2 Punkte

- ❏ Bei Verklemmungsverhinderung wird dafür gesorgt, dass eine der vier notwendigen Bedingungen für Verklemmungen nicht auftreten kann.
- ❏ Das System überprüft vor dem Freigeben von Betriebsmitteln, ob ein unsicherer Zustand eintreten würde.
- ❏ Mit Hilfe des Bankers-Algorithmus wird beim Belegen eines Betriebsmittels geprüft, ob mit erfolgreicher Belegung ein unsicherer Zustand eintreten würde.
- ❏ Bei einem Zyklus im erweiterten Betriebsmittelgraph liegt ein unsicherer Zustand vor.

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

- ❏ `close()`
- ❏ `accept()`
- ❏ `mmap()`
- ❏ `shutdown()`

d) Welche Aussage ist in einem Monoprozessor-Betriebssystem **falsch**? 3 Punkte

- ❏ Ein Prozess im Zustand *blockiert* muss warten, bis der laufende Prozess den Prozessor abgibt und kann dann in den Zustand *laufend* überführt werden.
- ❏ Es befindet sich maximal ein Prozess im Zustand *laufend* und damit in Ausführung auf dem Prozessor.
- ❏ Ist zu einem Zeitpunkt kein Prozess im Zustand *laufend*, so ist auch kein Prozess im Zustand *bereit*.
- ❏ In den Zustand *blockiert* gelangen Prozesse in der Regel nur, wenn sie vorher den Zustand *laufend* inne hatten.

e) Was versteht man unter Virtuellem Speicher? 2 Punkte

- ❏ Speicher, in dem sich keine Daten speichern lassen, weil er physikalisch nicht vorhanden ist.
- ❏ 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.

f) Inodes sind Verwaltungs-Datenstrukturen in einem UNIX-Dateisystem? Welche Aussage ist richtig? 2 Punkte

- ❏ Ein Inode ist ein Eintrag in einem Dateikatalog. Der Inode enthält den Dateinamen sowie eine laufende Nummer über die die Datei auf der Platte gefunden werden kann.
- ❏ Ein Inode ist eine Datenstruktur, die Dateiattribute und Informationen über den Speicherort des Dateiinhalts enthält. Sie enthält aber nicht den Dateinamen. Der Dateiname ist an anderer Stelle gespeichert - diese Stelle nennt man "symbolic link".
- ❏ Dateikataloge enthalten Einträge, die auf Inodes verweisen. Durch den Systemaufruf "unlink" wird solch ein Eintrag gelöscht. Das bedeutet aber nicht, dass auch der zugehörige Inode in jedem Fall mit gelöscht wird.
- ❏ Wenn ein symbolic Link gelöscht wird, wird auch der Inode der Datei, auf die dieser Link zeigte, gelöscht.

g) User-Level- und Kernel-Level-Threads unterscheiden sich in verschiedenen Eigenschaften. Welche Kombination ist richtig? 2 Punkte

- ❏ Bei User-Level-Threads können anwendungsabhängig Schedulingstrategien eingesetzt werden; blockierende Systemaufrufe von Kernel-Level-Threads blockieren keine anderen Threads.
- ❏ Kernel-Level-Threads werden sehr effizient umgeschaltet; User-Level-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ❏ Bei Kernel-Level-Threads ist die Schedulingstrategie meist vorgegeben; User-Level-Threads können Multiprozessoren ausnutzen.
- ❏ User-Level-Threads werden nicht effizient umgeschaltet; blockierende Systemaufrufe von Kernel-Level-Threads blockieren keine anderen Threads.

h) Wie gross ist die Seitenkacheltabelle zur Adressabbildung von logischen auf physikalische Adressen in einem System mit Seitenadressierung wenn ein Eintrag in der Seitenkacheltabelle 32 Bit groß ist, der virtuelle Adressraum 4 GigaByte umfasst und eine Speicherseite 4096 Byte groß ist. 3 Punkte

- ❏ 32 Byte
- ❏ 512 bis 8.192 Byte
- ❏ 1.048.576 Byte
- ❏ 4.194.304 Byte

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

- ❏ Der Prozess wurde von einem Prozess mit einer höherer Priorität verdrängt.
- ❏ Der Prozess möchte Daten von der Festplatte lesen und die Daten stehen noch nicht zur Weiterbearbeitung bereit.
- ❏ Der Prozess hat einen Seitenfehler für eine Seite, die noch nicht im Hauptspeicher vorhanden ist.
- ❏ Der Prozess hat auf Daten von der Tastatureingabe gewartet und diese stehen nun zur Weiterbearbeitung bereit.

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 andere Prozess die Zeit sinnvoll nutzen könnte.
- ❏ Der Algorithmus von Peterson ist ein Koordinierungsmittel, welches aktiv auf die Freigabe von Betriebsmitteln wartet.
- ❏ Aktiv wartende Koordinierungsmittel sind nur sehr aufwendig zu realisieren, da eine zusätzliche Warteschlange für alle Prozesse benötig wird.
- ❏ Auf Systemen mit mehreren Prozessoren sind Programme mit aktiv wartenden Koordinierungsmittel häufig schneller.

k) Bei einer prioritätengesteuerten Prozess-Auswahlstrategie kann es zu Problemen kommen. Welches der folgenden Probleme kann auftreten? 2 Punkte

- ❏ Die Anzahl der Prioritäten reicht nicht aus, wenn nur wenige Prozesse vorhanden sind.
- ❏ Prozesse können ausgehungert werden.
- ❏ Das Phänomen der Prioritätsumkehr hungert niedrig-priore Prozesse aus.
- ❏ Die Auswahlstrategie arbeitet ineffizient, wenn viele Prozesse im Zustand bereit sind.

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

- ❏ Der Systemaufruf nimmt eine Verbindung entgegen 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.

m) Beim Einsatz von RAID 5 wird durch eine zusätzliche Festplatte Datensicherheit erzielt, so dass der Ausfall einer Festplatte den laufenden Betrieb nicht stören kann. Welche Aussage dazu ist richtig? 2 Punkte

- ❏ Es dürfen nicht mehr als 5 Festplatten beteiligt sein, da sonst die Paritätsinformation nicht mehr gebildet werden kann.
- ❏ Es sind mindestens 5 Festplatten nötig.
- ❏ Die Paritätsinformation wird gleichmäßig über alle beteiligten Platten verteilt.
- ❏ Der Lesezugriff auf ein Plattensystem mit RAID 5 ist langsamer als bei normalen Plattenzugriffen, da der Zugriff auf die Platten komplexer ist.

n) Ein Betriebssystem setzt logische Adressräume auf der Basis von Segmentierung ein. Welche Aussage ist **falsch**? 2 Punkte

- ❏ Die Segmentierung schränkt den logischen Adressraum derart ein, dass nur auf gültige Speicheradressen erfolgreich zugegriffen werden kann.
- ❏ Über gemeinsame Segmente kann innerhalb von zwei verschiedenen logischen Adressräumen auf die selben Speicherzellen zugegriffen werden.
- ❏ Mit der Segmentierung kann einem Prozess mehr Speicher zugeordnet werden, als physikalisch vorhanden ist, da ein Segment teilweise ausgelagert werden kann.
- ❏ Segmente können verschiedene Länge haben. Eine Längenbegrenzung wird üblicherweise bei der Speicherabbildung geprüft.

Aufgabe 2: (60 Punkte)

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

Das **`fileattrd`**-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 (die Bearbeitung erfolgt in der Funktion `serve(int socket)`), der Vaterprozess steht sofort für die nächste Verbindung wieder bereit.
- Ein Client sendet über die Verbindung die Namen der Dateien (komplette Pfadnamen), deren Attribute er abfragen möchte. Es wird jeweils ein Dateiname in einer Zeile gesendet. Ein Dateiname kann maximal 128 Zeichen lang sein.
- Der **`fileattrd-`**Server ruft zu jedem angefragten Dateinamen die Funktion `prattr(char *filename)` auf. Die Funktion gibt den Dateinamen, eine Information über den Dateityp (D für Directory, F für "normale Datei" (*regular file*) und X für sonstige Dateitypen) sowie bei normalen Dateien die Dateigröße aus.
  Beispiel für eine Ausgabe:
  ```
  /etc/passwd F 2067
  /usr/bin D
  /dev/tty X
  ```
- Werden die Attribute einer nicht-existierenden oder nicht-zugreifbaren Datei abgefragt, so wird der Dateiname, gefolgt von der Fehlermeldung ausgegeben.
- Der bearbeitende Sohnprozess terminiert, wenn er alle Daten des Client abgearbeitet hat und der Client die Verbindung geschlosssen hat.
- Der Sohnprozess geht beim Terminieren in den Zustand "ZOMBIE" über. Sorgen Sie in Ihrem Programm dafür, dass sich solche terminierten Sohnprozesse nicht ansammeln, sondern dass sie geeignet aufgeräumt werden. Hierfür gibt es verschiedene Möglichkeiten - erläutern Sie die von Ihnen gewählte Lösung durch einen kurzen Kommentar im Programmtext.

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 <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 vorbereitet */

    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
```

A:


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

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

    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
.....

    /* Warteschlange ankommender Verbindungen
        auf 5 einstellen */

    .....
    .....
```

S:

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

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

S:


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

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

```
   /* Sohnprozess */
```

```
   /* Vaterprozess */
```

P:

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

```
/* serve-Funktion */
```

a

w

l

p

V:


```
/* prattr-Funktion */
```

a

s

tg

f

```
/* weitere Funktionen, falls benoetigt */
```

T:

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

Ein Aufruf von

```
make fileattrd
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

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

Aufgabe 3: (20 Punkte)

a) Bei Virtuellem Speicher kann es zu Seitenfehlern (page fault) kommen. In welchen Situationen tritt ein Seitenfehler auf und was muss das Betriebssystem jeweils zur Behebung tun (beschreiben Sie drei solche Situationen)?

b) Falls keine freie Kachel im Hauptspeicher verfügbar ist, muss eine Seite ausgelagert werden. Wie würde die optimale Seitenersetzungsstrategie funktionieren und warum ist sie nicht realisierbar?

c) Eine mögliche Approximation der optimalen Strategie wäre LRU. Beschreiben Sie die Funktionsweise von LRU. Wie könnte man LRU implementieren? Ist LRU eine gut oder problematisch implementierbare Strategie? Warum?

d) Clock (oder Second Chance) ist eine weitere Strategie, die eine möglichst optimale Seitenersetzung anstrebt. Beschreiben Sie die grundsätzliche Funktionalität dieser Strategie. Hat sie gegenüber LRU Vor- bzw. Nachteile? Warum?

Aufgabe 4: (10 Punkte)

Es gibt unterschiedliche Möglichkeiten zur Realisierung von Aktivitätsträgern. Beschrieben Sie die verschiedenen, in der Vorlesung vorgestellten Konzepte, die Unterschiede sowie die Vor- und Nachteile.