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) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (Symbolic Links). Welche Aussage ist **richtig**? 2 Punkte

- ❏ Ein symbolischer Verweis kann ausschließlich auf Verzeichnisse verweisen.
- ❏ Für jeden symbolischen Verweis auf eine Datei wird in deren Inode ein Zähler erhöht.
- ❏ Der Systemaufruf `lstat()` liefert im Gegensatz zum Systemaufruf `stat()` die Dateiattribute des symbolischen Verweises und nicht die Attribute vom Ziel des Verweises.
- ❏ Symbolic Links können nur existieren wenn die verwiesene Datei oder das verwiesene Verzeichnis nicht gelöscht wurde.

b) Für welchen Zweck wird der Systemaufruf `listen()` benutzt? 2 Punkte

- ❏ Mit `listen()` wird ein Socket für die Verbindungsannahme vorbereitet. Ein Parameter gibt an, wieviele Verbindungsanfragen vor deren Annahme gepuffert werden können.
- ❏ Damit das Betriebssysteme für einen Socket überhaupt Systemaufrufe annimmt, muss es erst mit `listen()` in einen Modus des „Zuhörens“ gebracht werden.
- ❏ Der Aufruf von `listen()` wartet solange an einem Socket, bis eine einkommende Verbindungsanfrage vorliegt.
- ❏ Mit `listen()` wird ein Socket für die Verbindungsannahme vorbereitet. Ein Parameter gibt an, wieviele laufende Verbindungen maximal möglich sind.

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

- ❏ `shutdown()`
- ❏ `wait()`
- ❏ `dup()`
- ❏ `close()`

d) Welche Aussage über das aktuelle Arbeitsverzeichnis (Current Working Directory) trifft zu? 2 Punkte

- ❏ Jedem UNIX-Prozess ist zu jeder Zeit ein aktuelles Verzeichnis zugeordnet.
- ❏ Besitzt ein UNIX-Prozess kein Current Working Directory, so beendet sich der Prozess mit einem Segmentation Fault.
- ❏ Mit dem Systemaufruf `chdir()` kann das aktuelle Arbeitsverzeichnis vom Vaterprozess verändert werden.
- ❏ Das aktuelle Arbeitsverzeichnis erbt der Prozess vom aktuellen Benutzer.

e) Was muss ein(e) Software-Entwickler(in) unbedingt beachten, damit seine/ihre Programme nicht Opfer eines Hacker-Angriffs werden? 2 Punkte

- ❏ Der Quellcode darf nicht herausgegeben werden, damit Schwachstellen nicht entdeckt werden können.
- ❏ Bei Verwendung der Programmiersprache C muss darauf geachtet werden, dass die Stringoperationen `strcpy()` und `strcat()` nur eingesetzt werden, wenn die Länge des zu übertragenden Strings noch in das Ziel-Array passt.
- ❏ Es dürfen keine vorgefertigten Bibliotheksfunktionen verwendet werden, weil deren Implementierung als nicht vertrauenswürdig eingestuft werden muss.
- ❏ Der Einsatz der Bibliotheksfunktion `system()` muss verboten werden, da diese Funktion nicht sicher ausgeführt werden kann.

f) Welche Aussage ist bezüglich der Seitenersetzungsstrategie Least Recently Used (LRU) richtig? 2 Punkte

- ❏ Die LRU Strategie benötigt ein Referenzbit in jedem Eintrag der Seitenkacheltabelle.
- ❏ Als Auswahlkriterium für die Ersetzung einer Seite wird die Zeit seit dem letzten Zugriff auf die Seite verwendet.
- ❏ Zur Implementierung von LRU benötigt man eine sehr genaue Systemuhr.
- ❏ Die LRU Strategie gewährleistet, dass immer die Seiten eingelagert sind, auf die in der Zukunft zugegriffen wird.

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

- ❏ Mit dem Systemaufruf `shmget()` kann ein Prozess einen Teil seines Speichers exportieren.
- ❏ Der Systemaufruf `shmget()` blendet ein bestehendes Shared-Memory-Segment in den logischen Adressraum des aufrufenden Prozesses ein.
- ❏ Der Aufruf von `shmget()` erzeugt ein Shared-Memory-Segment, 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 `shmget()` können die Prozesse anschließend auf den gemeinsamen Speicher zugreifen.

h) 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 schnellen Pufferspeicher vorgehalten wird.
- ❏ Der TLB puffert Daten bei der Ein-/Ausgabebehandlung und beschleunigt diese damit.
- ❏ Verändert sich die Speicherabbildung von logische auf physikalische Adressen 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.

i) Welche Aussage über die Koordinierung von kritischen Abschnitten unter Unix ist richtig? 2 Punkte

- ❏ In einem Unix Prozess kann es keinen kritischen Abschnitt geben, da ein Prozess immer nur einen Aktivitätsträger (Thread) besitzt.
- ❏ Kritische Abschnitte können unter Unix nur mit Semaphoren synchronisiert werden.
- ❏ Ein Unix Prozess kann durch das Sperren von Unterbrechungen (Interrupts) den Speicherzugriff in einem kritische Abschnitte synchronisieren.
- ❏ Zum Synchronisiseren von Prozessen bietet Unix den Systemaufruf `semop()`.

j) Ein Betriebssystem setzt logische Adressräume auf der Basis von Segmentierung ein. Welche Aussage ist **falsch**? 3 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.

k) Was versteht man unter einer Unterbrechung bei der Ausführung von Instruktionen durch einen Prozessor? 2 Punkte

- ❏ Eine Signalleitung teilt dem Prozessor mit, dass er den aktuellen Prozess anhalten und auf das Ende der Unterbrechung warten soll.
- ❏ Mit einer Signalleitung wird dem Prozessor eine Unterbrechung angezeigt. Der Prozessor sichert den aktuellen Zustand bestimmter Register, insbesondere des Programmzählers, und springt eine vordefinierte Behandlungsfunktion an.
- ❏ Der Prozessor wird veranlasst eine Unterbrechungsbehandlung durchzuführen. Der gerade laufende Prozess kann die Unterbrechungsbehandlung ignorieren.
- ❏ Durch eine Signalleitung wird der Prozessor veranlasst, die gerade bearbeitete Maschineninstruktion zu unterbrechen und in den Benutzermodus umzuschalten.

l) In einem UNIX- oder Linux-System gibt es Spezialdateien zur Repräsentation von Geräten. Welche Aussage ist richtig? 1 Punkt

- ❏ Spezialdateien von Festplatten können nicht von einem Prozess angesprochen werden.
- ❏ Die Spezialdateien ermöglichen Anwendungsprozessen den Zugriff auf Geräte über das Dateisystem. Die Spezialdateien können wie reguläre Dateien mit `open()` und `close()` geöffnen bzw. geschlossen werden.
- ❏ Spezialdateien müssen immer in dem Verzeichnis „`/dev`“ stehen.
- ❏ Spezialdateien dürfen nur von Prozessen mit Administrator-(Root-)Rechten geöffnet werden.

m) Was versteht man unter der copy-on-write Technik? 3 Punkte

- ❑ Das Dateisystem ist besonders sicher organisiert. Immer wenn eine Datei modifiziert wird, wird zuerst eine Kopie angelegt.
- ❑ Ein spezieller Parameter eines Laserdrucker-Treibers mit dem man die Kopienanzahl einstellen kann.
- ❑ Für eine Seite, die von einem Prozess an einen anderen übertragen wird, wird erst dann eine Kopie angefertigt, wenn einer der beiden Prozesse schreibend auf eine Seite zugreift.
- ❑ Zum Schutz vor Buffer-Overflows wird vor Änderungen im Stack immer eine Kopie erzeugt.

n) Welche Aussage zur verketteten Speicherung von Datenblöcken auf einer Festplatte ist falsch? 2 Punkte

- ❑ Bei der verketteten Speicherung geht von den Nutzdaten eines Blocks der Platz für die Verzeigerung ab.
- ❑ Die verkettete Speicherung ist besonders fehleranfällig, da die Daten nicht mehr restauriert werden können, falls eine Verzeigerung einmal fehlt.
- ❑ Es kommt zur häufigen Positionierung des Schreib-/Lesekopfs bei verstreuten Datenblöcken.
- ❑ Das sequenzielle Lesen ist bei einer verketteten Speicherung besonders schnell möglich, da alle Datenblöcke verkettet sind.

o) Welche Aussage über Prozesszustände ist in einem Monoprozessor-Betriebssystem mit blockierenden Ein-/Ausgabeoperationen richtig? 2 Punkte

- ❑ Wenn gerade keine Prozessumschaltung stattfindet und kein Prozess im Zustand *laufend* ist, so ist auch kein Prozess im Zustand *blockiert*.
- ❑ Es befinden sich bis zu zwei Prozesse im Zustand *laufend* und damit in Ausführung auf dem Prozessor (Vordergrund- und Hintergrundprozess).
- ❑ Wenn kein Prozess im Zustand *laufend* ist, so gibt es auch keinen Prozess im Zustand *bereit*.
- ❑ Ein Prozess im Zustand *bereit* wird in den Zustand *blockiert* überführt, wenn seine Ein-/Ausgabeoperation nicht sofort abgeschlossen werden kann.

Aufgabe 2: (60 Punkte)

a) Schreiben Sie ein Programm `mailq` (Mail-Queue-Bearbeiter), das Mail aus einem Warteschlangendirectory an entfernte Rechner zustellt.

Wird `mailq` ohne Parameter aufgerufen, werden maximal 5 Aufträge gleichzeitig bearbeitet. Bei Aufruf `mailq` *`n`* werden bis zu n Aufträge gleichzeitig bearbeitet.

Das Programm soll folgendermaßen arbeiten:

- `mailq` sucht in dem Directory `/var/spool/mailq` nach Dateien, deren Name mit dem Zeichen `m` beginnt und bearbeitet den Auftrag durch Aufruf der Funktion `void pjob(char *mfile)`. Andere Dateien werden ignoriert.
- Funktion `pjob`: Falls die maximale Zahl gleichzeitig zu bearbeitender Aufträge bereits erreicht ist, wird gewartet, bis ein laufender Auftrag fertig wird. Sonst wird für den Auftrag ein Prozess erzeugt und zur eigentlichen Auftragsbearbeitung die Funktion `void send_mail(char *mfile)` aufgerufen.
- Zur Koordinierung der maximalen Prozesszahl haben Sie zwei **Alternativen**: **entweder:** nehmen Sie an, es stehen Ihnen eine Datei `pv.c` mit der Implementierung von zwei Funktionen mit der aus der Vorlesung bekannten Semantik von Operationen auf einem zählenden Semaphor `S` zur Verfügung:
  `int P(int *S)` und `void V(int *S)`
  Ein blockiertes P wird durch Signale unterbrochen und liefert dann immer Ergebnis -1 und setzt `errno` auf EINTR, erfolgreiches P liefert es als Ergebnis 1.
  oder: Sie zählen die Anzahl der erzeugten Prozesse in einer normalen Variablen, beachten die potentielle Nebenläufigkeit der Signalbehandlung und warten, falls die maximale Prozesszahl erreicht ist, mit `sigsuspend`.
- Funktion `send_mail`: In einer `mfile`-Datei steht in der ersten Zeile der Empfänger in der Form *`user@hostname`* und ab der zweiten Zeile bis zum Dateiende der Inhalt der Mail. Es wird eine Verbindung zu *`hostname`*, Port 25, aufgebaut und folgendes übertragen:
  `RCPT TO:` *`user@hostname`*
  `DATA`
  `Mailinhalt`
  `.`
  Abschliessend wird die `mfile`-Datei gelöscht.
- Sie können davon ausgehen, dass alle benutzten Dateinamen sowie die Zeilen im Mailinhalt jeweils nie länger als 1024 Zeichen sein werden und dass der Inhalt der `mfile`-Datei in jedem Fall korrekt aufgebaut ist.

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 <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <dirent.h>
#include <stdio.h>
#include <signal.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>

/* Funktionsdeklarationen, globale Variablen */
```

2A

```
/* Funktion main */

{
  /* Argumente prüfen */
```

A:


```
  /* Signalbehandlung fuer SIGCHLD aufsetzen */
  {
  }

  /* Auftraege suchen und bearbeiten */
  {
  }
} /* Ende Funktion main */
```

S:
D:

```
/* Auftragsbearbeitung: Funktion pjob */

void pjob(char *mfile) {
    /* Variablendefinitionen */

    /* Koordinierung mit der Signalbehandlung */

    /* Prozess erzeugen und Auftrag ausfuehren */

}/* Ende Funktion pjob */
```

K:P:


```
/* Mailuebertragung: Funktion send_mail */

void send_mail(char *mfile) {

  /* Variablendefinitionen */
  char user[1024], hostname[1024];

  /* m-Datei oeffnen */

  /* Empfaenger ermitteln */
   fscanf(mf, "%s@%s\n", user, hostname);
```

```
/* Adress-Struktur fuer hostname aufbauen */
{
   struct sockaddr_in addr;
   struct hostent *h;
   if ((h=gethostbyname(hostname)) == NULL) {
      perror(hostname);
      exit(EXIT_FAILURE);
   }

   memcpy(&addr.sin_addr, h->h_addr_list[0], sizeof(addr.sin_addr));
   addr.sin_port = htons(25);
   addr.sin_family = AF_INET;

   /* Verbindung aufbauen */

}
/* Mailinhalt uebertragen */
```

U:


```
   /* Auftrag abschliessen */

}
/* Ende Funktion send_mail */


/* Signalhandler fuer SIGCHLD */
void chld_handler() {

}
```

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

Ein Aufruf von

```
make mailq
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

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

K:
M:

Aufgabe 3: (15 Punkte)

a) Geben Sie einen kurzen Überblick über die Ihnen bekannten Schedulingstrategien (bis zu 5 Stück). Skizzieren Sie die Funktionsweise sowie die Vor- und Nachteile.

b) Welche Strategie würden Sie in einem Mehrbenutzer-Betriebssystem verwenden, in dem gleichzeitig mehrere interaktive Anwendungen und sehr rechenzeitintensive Hintergrundaufgaben zu bearbeiten sind?
Die Gesamteffizienz der Strategie hängt sicherlich von der Wahl geeigneter Parameter ab. Welche Parameter sind bei der von Ihnen gewählten Strategie einstellbar und wie wirkt sich welche Art der Einstellung auf das Systemverhalten aus?

Aufgabe 4: (15 Punkte)

a) Bei vielen Dateisystemen ist es von Zeit zu Zeit erforderlich, eine Defragmentierung vorzunehmen. Nennen Sie Beispiele. Warum tritt dieses Problem überhaupt auf.

b) Wie kann man das Problem der Fragmentierung durch die Art der Dateisystemorganisation vermeiden? Skizzieren Sie die Funktionsweise solcher Dateisysteme und beschrieben Sie, warum das Problem nicht auftritt.

c) Mit Hilfe von RAID-Techniken kann man die Geschwindigkeit und/oder die Sicherheit von Plattensystemen verbessern. Beschreiben Sie die Ihnen bekannten RAID-Techniken (inkl. Vor- und Nachteile).