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) Beim Zugriff auf Datei-Inhalte wird typischerweise zwischen sequentiellem und wahlfreiem Zugriff unterschieden. Welche Aussage ist **richtig**? 2 Punkte

- ❏ bei sequentiellem Zugriff dürfen verschiedene Benutzer nur hintereinander auf die Datei zugreifen
- ❏ wahlfreier Zugriff erfolgt nach beliebigem Muster und ist ideal bei allen denkbaren Speichermedien geeignet
- ❏ bei wahlfreiem Zugriff ist ein wohlgeordnetes Zugriffsmuster nicht erkennbar und das Verfahren ist z. B. bei Festplatten geeignet
- ❏ bei wahlfrei organisiertem Massenspeicher ist sequentieller Zugriff nicht möglich

b) Welche Aussage zum Thema "Aktives Warten" ist **richtig**? 2 Punkte

- ❏ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit
- ❏ Auf Mehrprozessorsystemen ist aktives Warten unproblematisch und deshalb dem passiven Warten immer vorzuziehen
- ❏ Aktives Warten darf bei nicht-verdrängenden Scheduling-Strategien auf einem Monoprozessorsystem nicht verwendet werden
- ❏ Bei verdrängenden Scheduling-Strategien verzögert aktives Warten nur den betroffenen Prozess, behindert aber nicht andere

c) 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

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

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

e) Welche Aussage zum Thema Prozesseinplanung ist **falsch**? 3 Punkte

- ❏ Bei Stapelbetrieb ist es ein Hauptziel, die Anzahl fertiggestellter Prozesse pro vorgegebener Zeiteinheit zu maximieren.
- ❏ Hohe Systemlast kann bei Echtzeitbetrieb zu einer langsameren Bearbeitung von Prozessen führen.
- ❏ Bei interaktivem Betrieb soll vor allem die Antwortzeit minimiert werden.
- ❏ In einem System mit einer Mischung aus Echtzeitbetrieb und interaktivem Betrieb müssen in jedem Fall alle Echtzeitaufgaben erledigt sein, bevor Prozesse des interaktiven Betriebs den Prozessor zugeteilt bekommen.

f) Sie kennen den Begriff Seitenflattern (Thrashing). Welche Aussage ist **richtig**? 3 Punkte

- ❏ Als Seitenflattern bezeichnet man das wiederholte löschen und neu laden des Translation-Look-Aside-Buffer (TLB), ausgelöst durch häufigen Prozesswechsel.
- ❏ Als Seitenflattern bezeichnet man das wiederholte Einlagern einer erst vor kurzem verdrängten Speicherseite. Die Prozesse verbringen als Folge die meiste Zeit mit dem Warten auf die Behebung von Seitenfehlern.
- ❏ Seitenflattern erkennt man an der starken Geräuschentwicklung der Festplatte, da auf Grund häufiger Seitenzugriffe der Lesekopf ständig neu positioniert wird. Bei Systemen ohne Festplatte (Thin-Clients) kann das Seitenflattern nicht auftreten.
- ❏ Durch die Verwendung von Semaphoren kann das unkontrollierte Flattern von Seiten synchronisiert werden.

g) Für lokale Variablen, Aufrufparameter, etc. einer Funktion wird bei vielen Prozessoren ein sog. Aktivierungsblock (activation record oder stack frame) auf dem Stack angelegt. Welche Aussage ist **richtig**? 3 Punkte

- ❏ Über Zeiger kann man alle Daten des Aktivierungsblocks der aufrufenden Funktion verändern.
- ❏ Nach dem Rücksprung aus einer Funktion sind Zeiger auf die Speicherzellen ihres Aktivierungsblocks nicht mehr gültig.
- ❏ Bei rekursiven Funktionsaufrufen kann der Aktivierungsblock in jedem Fall wiederverwendet werden, weil die gleiche Funktion aufgerufen wird.
- ❏ Der Compiler legt zur Übersetzungszeit fest, an welcher Position im Aktivierungsblock der main-Funktion die globalen Variablen angelegt werden.

h) 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
- ❏ Der Nachteil von hierarchischen Namensräumen besteht darin, dass das Dateisystem spezielle Funktionen zum Auflösen von Namenskonflikten implementieren muss
- ❏ In einem hierarchisch organisierten Namensraum dürfen gleiche Namen in unterschiedlichen Kontexten enthalten sein

i) Beim Blockieren in einem Monitor muss der Monitor freigegeben werden. Warum? 2 Punkte

- ❏ weil der Faden sonst aktiv warten würde
- ❏ weil sonst die Monitordaten inkonsistent sind
- ❏ weil ein anderer Faden die Blockierungsbedingung nur aufheben kann, wenn er den Monitor betreten darf
- ❏ weil kritische Abschnitte immer nur kurz belegt sein dürfen

j) Man unterscheidet Traps und Interrupts. Welche Aussage ist **richtig**? 3 Punkte

- ❏ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Interrupts immer an den gleichen Stellen auf.
- ❏ Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
- ❏ Der Zugriff auf eine logische Adresse kann zu einem Trap führen.
- ❏ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.

k) Virtualisierung kann als Maßnahme gegen Verklemmungen genutzt werden. Warum? 2 Punkte

- ❏ Im Fall einer Verklemmung können zusätzliche virtuelle Betriebsmittel neu erzeugt werden. Diese können dann eingesetzt werden, um die fehlenden physikalischen Betriebsmittel zu ersetzen.
- ❏ Durch Virtualisierung kann man über Abbildungsvorgänge Zyklen, die auf der logischen Ebene vorhanden sind, auf der physikalischen Ebene auflösen.
- ❏ Eine Verklemmungsauflösung ist einfacher, weil virtuelle Betriebsmittel jederzeit ohne Schaden entzogen werden können.
- ❏ Durch Virtualisierung ist ein Entzug von physikalischen Betriebsmitteln möglich, obwohl dies auf der logischen Ebene unmöglich ist.

l) In einem Seitendeskriptor werden verschiedene Informationen über eine Seite eines virtuellen Adressraums gehalten. Was gehört sicher **nicht** dazu? 2 Punkte

- ❏ die Adresse der Seite im Hauptspeicher
- ❏ die Position der Seite im virtuellen Adressraum
- ❏ Zugriffsrechte (z. B. lesen, schreiben, ausführen)
- ❏ ein Zähler, der ein Maß für das Alter der Seite enthält

m) In welcher Situation tritt **kein** Seitenfehler auf

3 Punkte

- ❑ die Seite wurde im Rahmen einer copy-on-write-Datenübertragung einem anderen Prozess zur Verfügung gestellt und der empfangende Prozess greift modifizierend auf die Seite zu.
- ❑ die Seite wurde im Rahmen einer copy-on-write-Datenübertragung einem anderen Prozess zur Verfügung gestellt und der sendende Prozess greift lesend auf die Seite zu.
- ❑ die Seite wurde auf Hintergrundspeicher geschrieben, wird in einem Freiseitenpuffer gehalten, ist noch in der Seitentabelle des Prozesses eingetragen und der Prozess greift modifizierend zu.
- ❑ das Anwesenheits-Bit der Seite wurde vom Betriebssystem zurückgesetzt und der Prozess greift danach erneut auf die Seite zu.

Aufgabe 2: (60 Punkte)

a) Schreiben Sie ein Programm `msgd` (Message-Daemon), das einen Mitteilungstext bei Anfragen über eine Netzverbindung ausgibt. Der Text wird dem Programm über eine Konfigurationsdatei zur Verfügung gestellt.

Dem Programm `msgd` kann als Parameter der Name der Konfigurationsdatei übergeben werden. Bei Aufruf ohne Parameter wird die Datei `msgd.conf` verwendet.

Das Programm soll folgendermaßen arbeiten:

- `msgd` startet zuerst einen Sohnprozess, der die eigentliche Arbeit im Hintergrund übernimmt (deshalb auch Daemon-Prozess genannt), gibt dessen Prozess-Id aus (`"msgd startet, PID=..."`) und terminiert dann sofort wieder. Die eigentliche Arbeit wird in der Funktion `void msgserver()` erledigt.
- Funktion `msgserver`: der Text aus der Konfigurationsdatei wird unter Verwendung der Funktion `char *getconf(int *size)` eingelesen. Anschliessend werden TCP-Verbindungen von beliebigen IP-Adressen auf Port 9999 erwartet.
- Es wird jeweils eine Verbindung angenommen, der konfigurierte Text auf die Verbindung ausgegeben und anschliessend die Verbindung beendet. Es sollen bis zu 20 Verbindungsanforderungen warten dürfen.
- Funktion `getconf`: Die Funktion ermittelt die aktuelle Größe der Konfigurationsdatei, allokiert den dafür benötigten Speicher, liest die komplette Konfigurationsdatei in diesen Speicherbereich ein und gibt einen Zeiger auf den Speicherbereich als Ergebnis zurück. Im Fehlerfall wird NULL geliefert. Im Parameter `size` wird die Länge des Textes zurückgegeben.
- Änderungen der Konfigurationsdatei können dem Daemon-Prozess durch ein SIGHUP-Signal mitgeteilt werden. Der Empfang des Signals soll bewirken, dass ab der nächsten TCP-Verbindungsanfrage der neue Text ausgegeben wird. Die Textausgabe auf einer gerade bestehenden Verbindung darf durch die Neukonfiguration nicht zerstört werden. Es muss entweder noch der alte oder schon der neue Text ausgegeben werden.
  Bedenken Sie, dass ein neuer Text eine andere Länge haben kann. Eine Neukonfiguration des Textes soll während der Laufzeit des Daemon-Prozesses beliebig oft möglich sein.
- Sie können davon ausgehen, dass der Name der Konfigurationsdatei nicht länger als 255 Zeichen ist. Die Länge der Datei selbst kann beliebig sein.

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


/* Funktionsdeklarationen, globale Variablen */
```



```
/* Funktion main */

{
  /* Argumente prüfen */

   /* Serverprozess starten */
   {
   }
} /* Ende Funktion main */
```

A:
P:

```
/* msgserver-Funktion */
void msgserver()
{
    /* Variablendefinitionen */

    char *message;       /* Text der auszugebenden Meldung */
    int msg_size;        /* Laenge der Meldung */
    int sock_accept;     /* Socketdeskriptor der angenommenen Verb. */

    /* Socket oeffnen, binden, usw. */
```

S:


```
    /* Signal-Handler für SIGHUP einrichten */
    {
    }

    /* Server-Schleife */
```

tg

I:
G:

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

s

```
        }

        /* Text ausgeben */
        {

          FILE *conn;
          int i;

          if ((conn = fdopen(sock_accept, "w")) == NULL ) {
            perror("fdopen socket");
            exit(EXIT_FAILURE);
          }
          for(i=0; i<msg_size; i++) putc(message[i], conn);
```

s

```
        }
    } /* Ende Server-Schleife */
} /* Ende msgserver-Funktion */
```

S:


```
/* Konfigurationsdatei einlesen: Funktion getconf */
char *getconf(int *s)
{
  /* Variablendefinitionen */
```

```
  /* Groesse der Datei ermitteln */
```

```
  /* Speicher allokieren */
```

```
  /* Datei einlesen */
```

```
  /* Ergebnis liefern */
```

```
} /* Ende Funktion getconf */
```

G:

```
/* signal-handler für SIGHUP */
```

i

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

Ein Aufruf von

```
make msgd
```

soll - falls erforderlich - die aktuelle Version der Quelldatei aus der RCS-Datei "auschecken" und das Programm erzeugen, ein Aufruf von

```
make clean
```

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

I:
M:

Aufgabe 3: (18 Punkte)

a) Skizzieren Sie (graphisch) die Abbildung von einer logischen Adresse in eine physikalische Adresse in einem System mit Segmentierung.

b) Was muss das Betriebssystem tun, wenn aufgrund von Hauptspeichermangel ein Segment ausgelagert werden soll? Was passiert, wenn der Prozess nach dem Auslagern auf die Daten des Segments zugreift?

c) Der Inhalt eines Segments soll als Nachricht von einem Prozess an einen anderen mittels copy-on-write-Technik übertragen werden. Was muss das Betriebssystem tun (z.B. was ist in welchen Datenstrukturen zu ändern) wenn:
1. die Nachricht von Prozess 1 abgeschickt wird (send)
2. die Nachricht von Prozess 2 empfangen wird (receive)
3. Prozess 2 nach dem Empfang den Segmentinhalt modifiziert
4. Prozess 2 den Segmentinhalt nur liest, dann terminiert und anschliessend modifiziert Prozess 1 den Segmentinhalt

Aufgabe 4: (12 Punkte)

Skizzieren Sie in einer programmiersprachen-ähnlichen Form die Programmierung der zwei Funktionen *put* und *get* (entspricht *store* und *fetch*), die in der üblichen Art und Weise als Erzeuger und Verbraucher auf einem Puffer fester Größe (Bounded Buffer) operieren.

Koordinieren Sie die Funktionen mit Hilfe von Semaphoren.

Beschreiben Sie kurz die Bedeutung der von Ihnen eingesetzten Semaphore und welche Werte sie initial haben müssen.

Gehen Sie davon aus, dass auch jeweils mehrere Erzeuger und Verbraucher gleichzeitig einen Zugriff versuchen könnten.