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 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 besonders aufwändig.
- ❏ Je kleiner die Speichereinheiten sind, desto länger ist die Bitliste.

b) Was versteht man unter einem Translation Lookaside Buffer (TLB)? 2 Punkte

- ❏ Einen speziellen Cache der MMU, der den Inhalt der zuletzt angesprochenen Speicherzellen vorhält.
- ❏ Einen speziellen Cache der MMU, der Informationen aus den zuletzt genutzten Seitendeskriptoren vorhält.
- ❏ Einen Pufferspeicher des Compilers, um den Übersetzungsvorgang zu beschleunigen (es werden die Codes der zuletzt übersetzten Statements vorgehalten).
- ❏ Der TLB ist eine schnelle Umsetzeinheit der MMU, die physikalische in logische Adressen umsetzt.

c) Wie groß ist die Seitentabelle zur Adressabbildung von logischen auf physikalische Adressen in einem System mit Seitenadressierung wenn ein Eintrag in der Seitentabelle 32 Bit groß ist, der virtuelle Adressraum 4 GiB umfasst und eine Speicherseite 8192 Byte groß ist. 2 Punkte

- ❏ 512 bis 8.192 Byte
- ❏ 2.097.152 Byte
- ❏ 4.194.304 Byte
- ❏ 16.777.216 Byte

d) 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.
- ❏ Unter einem Virtuellen Speicher versteht man einen physikalischen Adressraum, dessen Adressen durch eine MMU vor dem Zugriff auf logische Adressen umgesetzt werden.
- ❏ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.

e) 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.

f) 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.

g) 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 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.
- ❑ Es dürfen nicht mehr als 5 Festplatten beteiligt sein, da sonst die Paritätsinformation nicht mehr gebildet werden kann.

h) Welches Attribut ist **nicht** im Inode eines UNIX-Dateisystems verzeichnet? 1 Punkt

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

i) 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.

j) Was versteht man unter einem Interrupt? 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 abzubrechen.

k) Welche der folgenden Aussagen bezüglich Threads ist **falsch**? 2 Punkte

- ❑ Die Einlastung (das Dispatching) eines Threads ist eine privilegierte Operation und kann deshalb grundsätzlich immer nur durch das Betriebssystem vorgenommen werden.
- ❑ Das Betriebssystem ist nicht in der Lage, einen einzelnen User-Level-Thread bei einer fehlerhaften Operation (z. B. Segmentation fault) gezielt abzubrechen.
- ❑ Ein Anwendungsprogrammierer kann die Schedulingstrategie für seine User-Level-Threads selbst programmieren.
- ❑ Wenn ein User-Level-Thread im Rahmen einer read-Operation warten muss (blockiert wird), kann das Betriebssystem nicht auf einen anderen User-Level-Thread des gleichen Prozesses umzuschalten.

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

- ❑ Der Prozess wartet auf eine Tastatureingabe.
- ❑ Der Prozess wurde von einem Prozess mit einer höherer Priorität verdrängt.
- ❑ Der Prozess hat auf Daten von der Festplatte gewartet und die Daten stehen nun zur Weiterbearbeitung bereit.
- ❑ Der Prozess ist grundsätzlich lauffähig und wird im Rahmen der mittelfristigen Prozesseinplanung eingelagert.

m) Es gibt verschiedene Ursachen, wie Nebenläufigkeit in einem System entstehen kann (gewollt oder auch ungewollt). Was gehört **nicht** dazu? 2 Punkte

- ❑ durch Interrupts
- ❑ durch preemptives Scheduling
- ❑ durch Traps
- ❑ durch Threads auf einem Multiprozessorsystem

n) Welches der folgenden Verfahren ist zur Synchronisation des Zugriffs auf gemeinsame Daten in einem Multiprozessorsystem **nicht geeignet**? 2 Punkte

- ❑ Aktives Warten bis die Sperre aufgehoben wird
- ❑ Binäre Semaphore
- ❑ Spezialbefehle wie `cas`
- ❑ Sperre der Interrupts

o) Was versteht man unter Verklemmungsvorbeugung? 2 Punkte

- ❑ Das System überprüft vor dem Belegen von Betriebsmitteln, ob ein unsicherer Zustand eintreten würde.
- ❑ Bei einem Zyklus im Betriebsmittelbelegungsgraph wird einer der Prozesse aus dem Zyklus terminiert.
- ❑ In einem System wird dafür gesorgt, dass eine der vier Voraussetzungen für Verklemmungen nicht eintreten kann.
- ❑ Wurde ein unsicherer Zustand im System erkannt, wird vorbeugend einer der beteiligten Prozesse abgebrochen, bevor die Verklemmung eintritt.

Aufgabe 2: harsh (60 Punkte)

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

a) Schreiben Sie ein Programm `harsh` (Hopefully Airtight Remote SHell), das als Server entfernten Benutzern einen authentifizierten Shell-Zugriff ermöglicht.
Im Konfigurationsverzeichnis */etc/harsh* befindet sich für jeden gültigen Benutzer eine reguläre Datei, deren Dateiname dem Benutzernamen entspricht und die Passwort und Shellprogramm des Benutzers beinhaltet (Konfigurationsdatei des Benutzers). Bei jeder angenommenen Verbindung werden die vom Client übertragenen Daten mit der Konfigurationsdatei abgeglichen (Authentifizierung). Nach erfolgreicher Authentifizierung wird die konfigurierte Shell gestartet.
Die Bearbeitung einer angenommenen Verbindung erfolgt in einem Kindprozess. Die maximale Zahl der zu einem Zeitpunkt existierenden Kindprozesse ist beschränkt.

Das Programm soll folgendermaßen arbeiten:

- Das Programm erhält als Parameter die maximale Zahl der zu einem Zeitpunkt existierenden Kindprozesse sowie die Portnummer, auf der Verbindungen entgegengenommen werden sollen. Wird das Programm mit der falschen Zahl an Argumenten aufgerufen, soll es sich mit einer Fehlermeldung beenden.
- Der Server führt Buch über die aktuell laufende Zahl an Kindprozessen. Richten Sie eine Signalbehandlung ein, um das Terminieren von Kindprozessen zu bearbeiten.
- Der Server nimmt Verbindungen auf dem angegebenen Port entgegen. Nach Annahme einer Verbindung wird überprüft, ob die maximale Zahl an Kindprozessen bereits erreicht ist. In diesem Fall sendet der Server die Meldung "Server ueberlastet" an den Client und schließt die Verbindung. Sonst wird ein neuer Prozess zur Bearbeitung der Verbindung erzeugt, während der Vaterprozess weitere Verbindungen annimmt.
- Der Kindprozess ruft zur Authentifizierung des Clients die Funktion `authenticate` auf. Bei erfolgreicher Authentifizierung liefert die Funktion den Rückgabewert 1 und speichert den Pfad zur Shell des Benutzers im übergebenen Puffer. Zur weiteren Bearbeitung wird die Funktion `spawnShell` aufgerufen. Beim Fehlschlagen der Authentifizierung liefert `authenticate` den Rückgabewert 0, woraufhin die Meldung "access denied" an den Client übertragen und die Verbindung beendet wird.
- Funktion `int authenticate(FILE *conn, char *buf, size_t bufsize)`: Die Funktion erhält ein `FILE`-Handle auf die Client-Verbindung sowie die Adresse und Länge eines Puffers zur Ablage des Pfades der Benutzershell. Die Funktion liest zunächst Benutzername und Passwort in jeweils einer Zeile von der Verbindung und liest dann das hinterlegte Passwort und die Shell des Benutzers aus den ersten beiden Zeilen der Konfigurationsdatei des Benutzers (Länge der Strings jeweils max. 64 Zeichen). Die Authentifizierung ist erfolgreich, wenn das hinterlegte Passwort mit dem übertragenen Passwort übereinstimmt.
- Funktion `void spawnShell(FILE *conn, const char *shell)`: Die Funktion leitet die Standard-E/A-Kanäle (*stdin*, *stdout*, *stderr*) des Prozesses auf die Client-Verbindung um und lädt dann die als Parameter übergebene Shell ohne weitere Parameter in den Kindprozess.

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

/* Hilfsfunktion readline zum einfacheren Einlesen von Zeilen*
static char *readline(char *buf, int bufsize, FILE *cl) {
    char *res = fgets(buf, bufsize, cl);

    if(res != NULL && buf[strlen(buf)-1] == '\n') {
        buf[strlen(buf)-1] = '\0';
    }

    return res;
}

/* Makros, Funktionsdeklarationen, globale Variablen */
```



```
/* Funktion main */

{
   /* Variablendefinitionen */

   /* Argumente pruefen und weitere Vorbereitungen */

   /*  Signalbehandlung installieren */
```

A:

```
/* Socket oeffnen (wahlweise IPv4 oder IPv6)*/
```


```
/* Verbindungen annehmen */
```

```
/* prüfen, ob weiterer Prozess erzeugt werden darf */
```

```
/* Prozess erzeugen -> nächste Seite */
```

O:

```
/* Prozess erzeugen und Verbindung bearbeiten */
```

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

P:


```
/* Funktion authenticate */
```

T:

```
/* Funktion spawnShell */
```

```
/* Signalbehandlung für terminierende Kindprozesse */
```

S:
C:

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

Ein Aufruf von

```
make harsh
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

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

Sie dürfen bei Ihrer Lösung keine builtin-Regeln von make voraussetzen!

M:

Aufgabe 3: (21 Punkte)

a) 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 einen Ringpuffer fester Größe (Bounded Buffer) füllen bzw. leeren.

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.
(12 Punkte)

b) Die Verwendung blockierender Synchronisationsverfahren ist nicht immer unproblematisch. Beschreiben Sie drei Problembereiche, die blockierende Synchronisation mit sich bringen kann. (3 Punkte)

c) An welcher Stelle des in Teilaufgabe a beschriebenen Puffers könnte man einfach auch nicht-blockierende Synchronisation einsetzen? (2 Punkte)

d) Beschreiben Sie, wie nicht-blockierende Synchronisation prinzipiell (z.B. mit CAS) in einem einfachen Fall funktioniert. (4 Punkte)

Aufgabe 4: (9 Punkte)

Beschreiben Sie die unterschiedlichen Arten ("Gewichtsklassen") von Prozessen/ Threads.