**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) Was passiert, wenn Sie in einem C-Programm über einen ungültigen Zeiger versuchen auf Speicher zuzugreifen — 2 Punkte

- ❏ Der Compiler erkennt den ungültigen Zugriff und meldet beim Übersetzen einen Fehler.
- ❏ Beim Zugriff über den Zeiger erkennt die MMU, wenn sie die erforderliche Adressumsetzung vornimmt, die ungültige Adresse und löst einen Trap aus.
- ❏ Der Speicher schickt an die CPU einen Interrupt. Hierdurch wird das Betriebssystem angesprungen, das den gerade laufenden Prozess mit einem "Segmentation fault"-Signal unterbricht.
- ❏ Das Betriebssystem erkennt die ungültige Adresse bei der Weitergabe des Befehls an die CPU (partielle Interpretation) und leitet eine Ausnahmebehandlung ein.

b) Welche der folgenden Aussagen bzgl. Threads ist richtig? — 2 Punkte

- ❏ Die Einlastung (das Dispatching) eines Threads ist eine privilegierte Operation und kann deshalb grundsätzlich immer nur durch das Betriebssystem vorgenommen werden.
- ❏ Wenn ein User-Level Thread eine fehlerhaften Operation ausführt (z. B. Zugriff auf eine ungültige Adresse) wird er vom Betriebssystem abgebrochen. Andere User-Level-Threads der gleichen Anwendung können aber weiterarbeiten (Konzept der Fehlerisolation).
- ❏ Ein Anwendungsprogrammierer kann die Schedulingstrategie für seine User-Level Threads selbst programmieren.
- ❏ Die Erzeugung eines User-Level Threads ist teurer als die Erzeugung eines Kernel-Level Threads.





c) 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`
- ❏ Sperren der Interrupts

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

- ❏ Zugriffsrechte (z. B. lesen, schreiben, ausführen)
- ❏ die Adresse der Seite auf dem Hintergrundspeicher
- ❏ die Zahl der page-faults
- ❏ die Information, ob die Seite seit dem letzten Einlagern modifiziert wurde

e) Welche Aussage zum Thema Kooperatives Scheduling ist richtig? — 2 Punkte

- ❏ Kooperatives Scheduling ist in Mehrbenutzersystemen unproblematisch, so lange sich die Benutzer des Systems kennen und bereit sind, sich kooperativ zu verhalten.
- ❏ Bei kooperativem Scheduling kann ein fehlerhaftes Programm zur Blockade des Gesamtsystems führen.
- ❏ Kooperatives Scheduling wird vor allem in Echtzeitsystemen eingesetzt, da durch die explizite Abgabe des Prozessors in den verschiedenen Anwendungen die Umschaltzeitpunkte zur Laufzeit immer vorausberechnet werden können.
- ❏ Kooperatives Scheduling ist in Verbindung mit virtuellem Speicher gut einsetzbar, weil durch die immer wieder auftretenden Seitenfehler automatisch die CPU freigegeben wird.

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

- ❏ durch Interrupts
- ❏ durch Signale an einen UNIX-Prozess
- ❏ durch Koroutinen
- ❏ durch Threads und preemptives Scheduling auf einem Monoprozessorsystem

g) Welche Aussage zum Thema Monitor ist richtig? 3 Punkte

- ❏ bei einem Hoare’schen Monitor muss die Wartebedingung nach dem wait in jedem Fall neu ausgewertet werden, weil zwischenzeitlich ein anderer Prozess die Bedingung bereits wieder entkräftet haben kann.
- ❏ pthread_wait und pthread_broadcast realisieren die wait- und signal-Primitiven Hansen’scher Monitore
- ❏ bei Hansen’schen Monitoren kann das Anwendungsprogramm gezielt entscheiden, welcher wartende Prozess den Monitor nach der Freigabe betreten darf.
- ❏ wenn in einem Monitor die Wartebedinung nicht erfüllt ist, aber davon ausgegangen werden kann, dass die Wartezeit sehr kurz sein wird, kann auf die Freigabe des Monitors verzichtet werden und kurzzeitig aktiv gewartet werden.

h) Welche Aussage zu Speicherzuteilungsverfahren ist richtig? 2 Punkte

- ❏ die worst-fit-Strategie ist lediglich theoretisch interessant, da es in der Praxis nie sinnvoll ist, den am schlechtesten passenden Speicherplatz zuzuweisen.
- ❏ buddy-Verfahren sind nur bei sehr leistungsfähigen Rechnern und großem Speicher einsetzbar, weil sie aufwändig in der Berechnung sind und viel Verschnitt verursachen.
- ❏ best-fit ist in jedem Fall das beste Verfahren
- ❏ bei buddy-Verfahren gibt es keinen externen Verschnitt

i) Man unterscheidet zwischen privilegierten und nicht-privilegierten Maschinenbefehlen. Welche Aussage ist **richtig**? 2 Punkte

- ❏ Privilegierte Maschinenbefehle dürfen in Anwendungsprogrammen grundsätzlich nicht verwendet werden, der Compiler würde eine Fehlermeldung erzeugen.
- ❏ Die Benutzung eines privilegierten Maschinenbefehls in einem Anwendungsprogramm führt zu einer asynchronen Programmunterbrechung.
- ❏ Privilegierte Maschinenbefehle können durch Betriebssystemprogramme implementiert werden.
- ❏ Mit nicht-privilegierten Befehlen ist der Zugriff auf Geräteregister grundsätzlich nicht möglich.





j) 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

- ❏ Auf der zusätzlichen Festplatte wird Paritätsinformation über den Inhalt der anderen Festplatten gespeichert. Dies belastet die zusätzliche Platte besonders stark.
- ❏ 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.

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

- ❏ Einen Pufferspeicher des Compilers um den Übersetzungsvorgang zu beschleunigen (es werden die Codes der zuletzt übersetzen Statements vorgehalten).
- ❏ Einen speziellen Cache der CPU, der die zuletzt ausgeführten Maschinenbefehle zwischenspeichert (beschleunigt vor allem den Ablauf von Schleifen).
- ❏ 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.

l) Welche Aussage zum Aufbau einer Kommunikationsverbindung zwischen einem Client und Server über eine Socket-Schnittstelle ist richtig? 3 Punkte

- ❏ Der Server erzeugt einen Socket und ruft anschließend listen auf - der Client kann daraufhin mit connect eine Verbindung herstellen und Daten übertragen.
- ❏ Der Server richtet am Socket eine Warteschlange für ankommende Verbindungen ein und kann dann mit accept eine konkrete Verbindung annehmen. accept blockiert so lange die Warteschlange leer ist bzw. bis der Client seinerseits eine Verbindung mit connect aufbaut.
- ❏ Der Client kann erst connect aufrufen nachdem der Server accept aufgerufen hat - vorher würde der connect-Versuch mit "connection rejected" abgewiesen werden.
- ❏ Der Server signalisiert durch den Aufruf von connect, dass er zur Annahme von Verbindungen bereit ist, ein Client kann dies durch accept annehmen.

m) 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 `fgets()` muss verboten werden, da diese Funktion nicht sicher ausgeführt werden kann.

n) Wofür wird ein Freiseitenpuffer bei der Speicherverwaltung eingesetzt? 2 Punkte

- ❏ Um bei einem Seitenfehler möglichst schnell einen freien Seitenrahmen (page frame) zum Einlagern der fehlenden Seite zur Verfügung zu haben.
- ❏ Um die Zahl der Auslagerungen von Seiten zu minimieren.
- ❏ Um den virtuellen Speicher schnell vergrößern zu können.
- ❏ Um Seiten schneller auslagern zu können.





## Aufgabe 2: (60 Punkte)

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

a) Implementieren Sie eine vereinfachte Variante eines parallel arbeitenden make-Programms.

Aufruf: **`pmake` *`Zieldatei n`***

Das Programm startet bis zu **`n`** Compiler-Läufe zur Erzeugung von .o-Dateien parallel, wartet auf deren Fertigstellung und bindet schließlich die .o-Dateien zu einem ausführbaren Programm in der Zieldatei zusammen. Wird der Parameter **`n`** weggelassen, so wird nur jeweils ein Compiler gestartet.

Das Programm soll im Detail folgendermaßen arbeiten:

- pmake sucht zunächst im aktuellen Verzeichnis nach Dateien, die auf ".c" enden. Mit einer Funktion **`int needcompile(char *cfile, char*ofile);`** wird für jede .c-Datei überprüft, ob es eine entsprechende .o-Datei gibt und die .c-Datei nach Erstellung der .o-Datei nicht mehr verändert wurde (Rückgabewert 0). Falls Neuübersetzung erforderlich ist (Rückgabewert 1), ruft main zur Erzeugung der .o=-Datei die Funktion **`void pjob(char *cfile);`** auf.
- Funktion **`pjob`**: Falls die maximale Zahl parallel zu startender Compiler bereits läuft wird gewartet, bis wieder ein Compiler-Lauf fertig wird. Sonst wird der Auftrag durch Aufruf von **`/usr/bin/cc`**, Argumente **`cc -c` *`.c-Datei`*** abgewickelt.
- Sollte ein Compiler-Lauf mit einem Fehler terminieren (exit-Status != 0), so soll pmake diesen Fehler erkennen und abbrechen. Gerade laufende Compiler arbeiten dabei aber noch zu Ende, es werden aber keine mehr nachgestartet.
- Falls alle Compiler-Läufe fehlerfrei fertig geworden sind, bindet pmake am Ende alle .o-Dateien, die erzeugt oder zu einer .c-Datei vorgefunden wurden.
  pmake ruft hierzu **`cc -o` *`Zieldatei .o-Dateien`*** auf.
  Hinweis: am besten bauen Sie sich das hierzu benötige Argumentfeld bereits auf, während Sie das Verzeichnis nach .c-Dateien durchsuchen. Sie können davon ausgehen, dass es maximal 100 .c-Dateien zu einem Programm gibt.
- Zur Koordinierung der maximalen Prozesszahl haben Sie zwei **Alternativen**:
  **entweder:** 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`.
  **oder:** Sie nehmen an, es steht Ihnen eine Datei `pv.o` mit einer passenden 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 das Ergebnis 1.

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 Leitlinie zu geben. Es ist überall großzügig Platz gelassen, damit Sie auch weitere notwendige Anweisungen 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.

```
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <errno.h>
#include <dirent.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>

/* weitere Includes, Makros,
   Funktionsdeklarationen, globale Variablen */

/* Funktion main */

    /* lokale Variablen und was man sonst am
       Anfang so vorbereitet */
```

A:





```
    /* Aufrufargumente auswerten */

    /* Argumentfeld fuer Bindeaufruf initialisieren */

    /* Signalbehandlung initialisieren */
```

O:
A:
S:

```
/* .c-Dateien suchen und bearbeiten */
```

D:





```
/* auf Beendigung aller Compiler-Laeufe warten und
   .o-Dateien binden */
```

```
/* Ende main-Funktion */
```

S:
P:

```
/* Auftragsbearbeitung: Funktion pjob */

    /* Koordinierung mit der Signalbehandlung */

    /* Prozess erzeugen und Compiler starten */

/* Ende Funktion pjob */
```

S:
P:





```
/* Feststellen, ob Uebersetzung erforderlich ist:
   Funktion needcompile */

/* Signalbehandlung  fuer SIGCHLD  */
```

T:
C:

b) Schreiben Sie ein Makefile zum Erzeugen des pmake-Programms.
Ein Aufruf von

```
make pmake
```

soll das Programm erzeugen, ein Aufruf von

```
make clean
```

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





**Aufgabe 3: (11 Punkte)**

Festplatten sind in der Regel in Blöcke unterteilt. Um die Daten einer Datei zu speichern werden meist mehrere Blöcke benötigt. Beschreiben Sie drei Verfahren, wie das Betriebssystem dabei vorgehen und die Blöcke verwalten kann. Nennen Sie kurz (in Stichpunkten) die wesentlichen Eigenschaften und ggf. Vor- und Nachteile. Geben Sie zwei Beispiele an, wo solche Verfahren in der Praxis eingesetzt werden bzw. wurden.

**Aufgabe 4: (19 Punkte)**

a) Bei der Prozesseinplanung (Scheduling) unterscheidet man kurz-, mittel- und langfristige Einplanung.
Skizzieren Sie die dabei auftretenden Prozesszustände und die möglichen Übergänge. Schreiben Sie an die Übergänge jeweils ein kurzes Stichwort, das die Ursache für so einen Übergang beschreibt.

Das Problem des Seitenflatterns ist eng mit den beschriebenen Prozesszuständen verbunden.

b) Was versteht man unter Seitenflattern und wodurch wird es verursacht? Zu welchen Zustandsübergängen kommt es dabei bei den betroffenen Prozessen?

c) Wie kann man durch geeignete Maßnahmen der Prozesseinplanung dagegen vorgehen - welche Zustandsübergänge werden hierfür genutzt.

**M:**