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 Programm über einen ungültigen Zeiger versuchen auf Speicher zuzugreifen? 2 Punkte

- ❏ Die MMU erkennt die ungültige Adresse bei der Adressumsetzung und löst einen Trap aus.
- ❏ Der Arbeitsspeicher erkennt, dass es sich um eine ungültige Adresse handelt, und leitet eine Ausnahmebehandlung ein.
- ❏ Die MMU schickt an die CPU einen Interrupt. Hierdurch wird das Betriebssystem angesprungen, das den gerade laufenden Prozess mit einem "Segmentation fault"-Signal unterbricht.
- ❏ Beim Binden des Programms wird die ungültige Adresse erkannt und ein Sprung auf eine Abbruchfunktion eingefügt. Diese Funktion beendet das Programm mit der Meldung "Segmentation fault".

b) 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 der möglichen Speicherabbildungen in einem sehr schnellen Pufferspeicher vorgehalten wird.
- ❏ Der TLB puffert Daten bei der Ein-/Ausgabebehandlung und beschleunigt diese damit.
- ❏ Der TLB ist eine schnelle Umsetzeinheit der MMU, die physikalische in logische Adressen umsetzt.
- ❏ Wird eine Speicherabbildung im TLB nicht gefunden, wird der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (Segmentation Fault) abgebrochen.

c) Wodurch kann es zu Seitenflattern kommen? 2 Punkte

- ❏ wenn bei einer nicht-verdrängenden Scheduling-Strategie die Zahl der von den Prozessen aktiv genutzten Seiten die Zahl der verfügbaren Seitenrahmen übersteigt
- ❏ durch Programme, die eine Defragmentierung auf der Platte durchführen
- ❏ wenn sich zu viele Prozesse im Zustand blockiert befinden
- ❏ wenn zu viele Prozesse im Rahmen der mittelfristigen Einplanung ausgelagert (swap-out) wurden

d) In einem Segmentdeskriptor werden verschiedene Informationen über ein Segment eines logischen Adressraums gehalten. Was gehört sicher **nicht** dazu? 2 Punkte

- ❏ die Benutzer-Ids, für die diese Zugriffsrechte gelten
- ❏ die physikalische Adresse des Segmentanfangs im Hauptspeicher
- ❏ Zugriffsrechte (z. B. lesen, schreiben, ausführen)
- ❏ die Länge des Segments

e) Welches der folgenden Verfahren trägt in der Praxis am besten dazu bei, Seitenfehler und ihre Auswirkungen zu minimieren? 2 Punkte

- ❏ man setzt eine Segmentierung in Kombination mit Seitenadressierung ein
- ❏ man ermittelt, welche der Seiten eines Prozesses in Zukunft am längsten nicht angesprochen wird und lagert genau diese aus (OPT-Strategie)
- ❏ man übergibt Prozesse, die einen Seitenfehler verursachen der mittelfristigen Prozesseinplanung, damit sie in nächster Zeit nicht wieder aktiv werden
- ❏ man lagert regelmäßig länger nicht genutzte Seiten aus und trägt sie in einem Freiseitenpuffer ein.

f) Wozu benötigt man Bedingungsvariablen (condition variables)? 2 Punkte

- ❏ Bei if-Abfragen in C-Programmen.
- ❏ Um in einem kritischen Abschnitt auf ein Ereignis zu warten und den kritischen Abschnitt während der Wartezeit freizugeben.
- ❏ Zur Implementierung von Spinlocks.
- ❏ Zur Erkennung von Verklemmungen.

g) Welche Aussage zu Dateisystemen ist **falsch**? 2 Punkte

- ❏ Die Prüfoperation beim Hochfahren eines UNIX-Systems kann auf einem typischen UNIX-Dateisystem (z.B. UFS) inkonsistente Zustände erkennen und beseitigen. Dabei gehen unter Umständen Daten verloren.
- ❏ 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. Vor der Veränderung eines so markierten Dateisystems muss die Marke entfernt werden.
- ❏ Bei Journalled-Dateisystemen wird die Konsistenz durch ein Transaktionskonzept mit Protokollierung der Änderungen sichergestellt.

h) Welche Aussage über den Rückgabewert von `wait()` ist richtig? 1 Punkt

- ❏ Dem Sohn-Prozess wird die Prozess-ID des Vater-Prozesses zurückgeliefert.
- ❏ Der Systemaufruf wait() wartet auf die Eingabe eines beliebigen Zeichens von der Tastatur, der entsprechende ASCII-Wert wird als Rückgabewert geliefert.
- ❏ Im Fehlerfall wird eine Fehlernummer ungleich -1 zurückgeliefert.
- ❏ Beim Beenden eines Sohn-Prozesses wird dessen Prozess-ID zurückgeliefert.

i) Bei RAID 0 wird duch das Verteilen der Daten auf mehreren Platten ein sogenanntes gestreiftes Plattensystem erzeugt. 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.
- ❏ Der Lesezugriff auf ein gestreiftes Plattensystem ist schneller, da mehrere Platten gleichzeitig beauftragt werden können.
- ❏ Die Paritätsinformation wird gleichmäßig über alle beteiligten Platten verteilt.
- ❏ Die Daten auf einem RAID-0-Festplattensystem sind besonders sicher, da beim Ausfahl von nur einer Platte die verlorengegangenen Daten wieder hergestellt werden können.

j) User-Level- und Kernel-Level-Threads unterscheiden sich in verschiedenen Eigenschaften. Welche Kombination ist richtig? 3 Punkte

- ❏ Bei User-Level-Threads können anwendungsabhängig Schedulingstrategien eingesetzt werden; Kernel-Level-Threads können Multiprozessoren nicht ausnutzen.
- ❏ Kernel-Level-Threads werden am effizientesten 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 effizient umgeschaltet; blockierende Systemaufrufe von Kernel-Level-Threads blockieren keine anderen Threads.

k) 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 wird Prozessen die sich nicht kooperativ verhalten (z. B. in einer Endlosschleife rechnen) zwangsweise der Prozessor entzogen.
- ❏ Kooperatives Scheduling ist im Mehrprogrammbetrieb gut einsetzbar, weil das Betriebssystem im Rahmen der Systemaufrufe der beteiligten Prozesse Prozessumschaltungen vornehmen kann und damit eine Monopolisierung der CPU durch einen Prozess in jedem Fall automatisch verhindert wird.
- ❏ Programme, die unter kooperativem Scheduling zum Einsatz kommen, müssen auf diese Situation entsprechend vorbereitet sein (z. B. durch Einstreuen von regelmäßigen Aufrufen an das Betriebssystem), wenn eine Monopolisierung der CPU vermieden werden soll.

l) Wozu dient der Maschinenbefehl cas (compare-and-swap)? 2 Punkte

- ❏ Um bei Monoprozessorsystemen Interrupts zu sperren.
- ❏ Um auf einem Multiprozessorsystem einfache Modifikationen an Variablen ohne Sperren implementieren zu können.
- ❏ Um bei der Implementierung von Schlossvariablen (Locks) aktives Warten zu vermeiden.
- ❏ Zur Realisierung einer effizienten Verdrängungssteuerung bei einseitiger Symchronisation.

m) Was versteht man unter Verklemmungsvermeidung? 2 Punkte

- ❏ In einem System wird dafür gesorgt, dass eine der vier hinreichenden Bedingungen für Verklemmungen nicht eintreten kann.
- ❏ Das System überprüft vor dem Freigeben von Betriebsmitteln, ob ein unsicherer Zustand eintreten würde.
- ❏ Eine Betriebsmittelanforderung wird nicht gewährt, wenn dadurch ein sicherer Zustand verlassen würde.
- ❏ Bei einem Zyklus im Betriebsmittelgraph wird einer der Prozesse aus dem Zyklus terminiert.

n) Welche Aussage zu Fäden (Threads) ist **falsch**? 2 Punkte

- ❏ Kernfäden können als virtuelle Prozessoren zur Ausführung von Benutzerfäden eingesetzt werden.
- ❏ Der Synchronisationsbedarf im Anwendungsprogramm kann von der Ablaufplanung der Kernfäden abhängen.
- ❏ Das Betriebssystem kann bei einem durch einen Benutzerfaden ausgelösten Seitenfehler nicht auf einen anderen Benutzerfaden umschalten.
- ❏ Das Betriebssystem führt Buch über Kernfäden und Benutzerfäden

o) Welche Angaben enthält die Implementierung eines Verzeichniseintrags in einem Standard-UNIX-Dateisystem (z.B. UFS, EXT2)? 2 Punkte

- ❏ Blocknummer des Inode-Plattenblocks und Dateiname
- ❏ Inode-Nummer und Dateiname
- ❏ Dateiname, Blocknummer des ersten Datenblocks, Dateigröße, Eigentümer und Zugriffsrechte
- ❏ nur Inode-Nummer

Aufgabe 2: prplex (60 Punkte)

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

a) Schreiben Sie ein Programm `prplex` (PRinter multiPLEXer), das als Server mehrere Warteschlangen für einen Drucker anbietet.

`prplex` liest beim Start aus dem Konfigurationsverzeichnis */etc/prplex* die verfügbaren Warteschlangen aus jeweils einer Konfigurationsdatei und startet für jede Warteschlange einen Thread, der auf einem TCP-Socket Verbindungen für diese Warteschlange entgegennimmt. Über einen Ringpuffer werden angenommene Verbindungen an den Hauptthread übergeben, der dann den Auftrag an den Drucker übermittelt.

Das Programm soll folgendermaßen arbeiten:

- Aus dem Konfigurationsverzeichnis werden beim Programmstart alle Dateien mit Anfangsbuchstaben 'c' gelesen. Jede dieser Dateien enthält in der 1. Zeile eine Portnummer und in der 2. Zeile einen Header-String für die Warteschlange (max. Zeilenlänge 1023 Zeichen). Die beiden Werte werden in eine von Ihnen zu definierende `pqconf`-Struktur eingelesen. Dann wird ein Thread zur Bearbeitung der Warteschlange gestartet, dem als Parameter ein Zeiger auf die Struktur übergeben wird.
- Jeder dieser Threads nimmt nun Verbindungen auf dem zu der Warteschlange gehörigen Port an. Für jede angenommene Verbindung schreibt der Thread den Filedeskriptor der Verbindung und die Headerzeile seiner Warteschlange in den Ringpuffer.
- Der Ringpuffer besteht aus 1024 `cconn`-Strukturen, die jeweils einen Filedeskriptor und einen String aufnehmen können.
- Nach Erzeugen der Threads öffnet der Hauptthread das Druckerdevice (=Datei) unter dem Pfad */dev/lp0*. Im weiteren Verlauf entnimmt der Thread dann jeweils mit der Funktion *nextJob* eine wartende Verbindung aus dem Ringpuffer und ruft zu deren Bearbeitung die Funktion *handleJob* auf.
- Funktion `struct cconn nextJob()`: Die Funktion entnimmt eine `cconn`-Struktur aus dem Ringpuffer und liefert diese zurück. Wenn der Ringpuffer leer ist, blockiert die Funktion solange, bis wieder ein Auftrag eintrifft.
- Funktion `void handleJob(FILE *printer, struct cconn *conn)`: Die Funktion erhält als Parameter das Filehandle des geöffneten Drucker-Devices und einen Zeiger auf die von *nextJob* gelieferte `cconn`-Struktur. Dann wird zunächst die Headerzeile der Warteschlange, über die der Job eingeliefert wurde, und dann alle über die Verbindung übertragenen Daten auf das Druckerdevice ausgegeben. Danach wird die Verbindung beendet womit die Bearbeitung des Jobs abgeschlossen ist.
- Zur Synchronisation auf dem Ringpuffer können Sie eine **gegebene** Implementierung von zählenden Semaphoren verwenden, welche die Funktionen

```
SEM *sem_init(int initial_value); (schlägt nie fehl!)
void P(SEM *sem);
void V(SEM *sem);
```

zur Verfügung stellt.

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

SEM * sem_init(int initial_value);
void P(SEM *sem);
void V(SEM *sem);

#define CFGDIR "/etc/prplex"
#define PRDEV  "/dev/lp0"

struct cconn {
  const char *header;
  int fd;
};
```

/* Funktionsdeklarationen, globale Variablen, Strukturen */

A:

/* Funktion main */

```
{
```

/* Variablendefinitionen, Initialisierungen */

3A

/* Konfigurationsverzeichnis öffnen */

1,5D

A:
D:

```
/* Verzeichnis durchsuchen, Konfigurationsdateien lesen,
   Threads erzeugen */
```

```
/* Konfigurationsdatei einlesen */
```

D:


```
/* Thread erzeugen */
```

```
/* Ende Verzeichnis durchsuchen,
   Fehlerbehandlung readdir */
```

```
/* Drucker öffnen */
```

```
/* Aufträge bearbeiten */
```

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

D:
P:

```
/* Funktion nextJob */
```

```
/* Funktion handleJob */
```

N:
H:


```
/* Thread-Funktion */
void *tstart(struct pqconf *pqcs) {
```

```
/* Verbindungen annehmen und in Puffer eintragen */
```

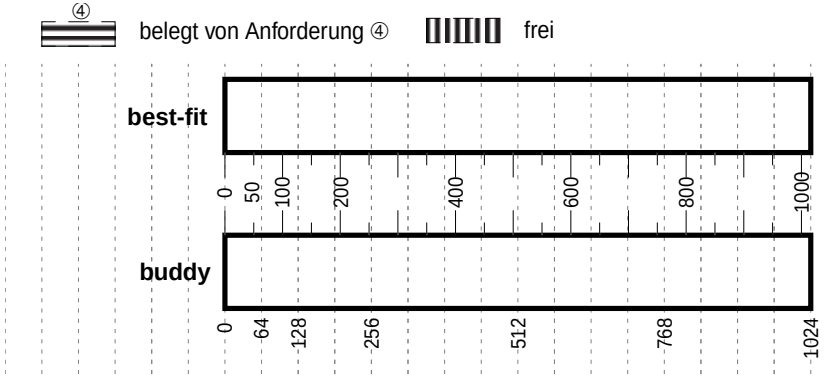
T:

Aufgabe 3: (20 Punkte)

Zur Verwaltung von freiem Speicher (z. B. in Funktionen wie malloc und free) gibt es verschiedene Strategien bei der Speicherzuteilung.

a) Nehmen Sie einen Speicher von 1024 Byte an, initial ist ein Datenblock der Größe 50 Byte an Adresse 0 vergeben. Ein Programm führt die im folgenden Bild angegeben malloc-Aufrufe aus.
Geben Sie für das das *best-fit*- und für das *Buddy*-Verfahren an, welches Ergebnis diese Aufrufe jeweils zurückliefern, und skizzieren Sie in der Grafik, wie der Speicher danach aussieht (kennzeichnen Sie die vergebenen Speicherbereiche jeweils mit der Nummer der malloc-Aufrufe). Auch der initial vergebene Block wird mit dem jeweilgen Verfahren verwaltet und ist in der Skizze zu berücksichtigen.

	best-fit	buddy	
① `malloc(100);`	300		
② `malloc(50);`			
③ `malloc(100);`			
④ `malloc(100);`			
⑤ `malloc(150);`			



b) nun werden der Reihe nach in den Schritten A, B, C und D die Bereiche
A: ①, B: ③, C: ⑤, D: ④
freigegeben.
Geben Sie den initialen Aufbau der Freispeicherstrukturen (nach malloc ⑤) und jeweils nach jedem Schritt an.

best-fit

init.

A

B

C

D

Buddy

	init.	A	B	C	D
2^{10}					
2^{9}					
2^{8}					
2^{7}					
2^{6}					
2^{5}					
2^{4}					
2^{3}					
2^{2}					
2^{1}					

Aufgabe 4: (10 Punkte)

Beschreiben Sie die Prozesszustände bei kurz- und mittelfristiger Einplanung sowie die Ereignisse, die jeweils zu den Zustandsübergängen führen (Skizze mit kurzer Erläuterung der Zustände und Übergänge).