Aufgabe 1.1: Einfachauswahl-Fragen (22 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒)und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten!

a) Beim Einsatz von RAID-Systemen kann durch zusätzliche Festplatten ein fehlertolerierendes Verhalten erzielt werden. Welche Aussage dazu ist richtig? 2 Punkte

- ❏ Bei RAID-4-Systemen wird die Paritätsinformation gleichmäßig über alle beteiligten Platten verteilt.
- ❏ Der Lesezugriff auf ein gestreiftes Plattensystem, insbesondere auch auf ein RAID-5 System, ist schneller, da mehrere Platten gleichzeitig beauftragt werden können.
- ❏ Wenn bei einem RAID-4-System die Parity-Platte ausfällt, sind alle Daten verloren.
- ❏ Bei einem RAID-5-System kommen immer genau 5 Platten zum Einsatz. Die Parity-Information wird dabei auf alle 5 Platten gleichmäßig verteilt.

b) Welche Aussage zu den Eigenschaften eines Journaling-Filesystems ist richtig? 2 Punkte

- ❏ Alle Änderungen am Dateisystem werden in Form von Transaktionen in einer Log-Datei mitprotokolliert. Wird nach einem Systemabsturz festgestellt, dass eine Transaktion in der Log-Datei unvollständig ist, muss die betroffene Datei gelöscht werden.
- ❏ Es wird immer zuerst die Änderung im Dateisystem auf der Platte duchgeführt und anschließend zur Absicherung ein entsprechender Eintrag in die Log-Datei geschrieben.
- ❏ Je größer eine Platte ist, desto länger dauert der Reparaturvorgang eines Journaling-Filesystems nach einem Systemabsturz.
- ❏ Die Einträge in der Protokolldatei müssen immer auch Informationen zu einem *Undo* und *Redo* der Transaktion enthalten.

c) Wie werden im Rahmen von Seitenersetzungsverfahren Freiseitenpuffer genutzt? 2 Punkte

- ❏ Wenn zu wenige freie Seitenrahmen im System vorhanden sind (low water mark erreicht) werden prophylaktisch Seiten ausgelagert, selbst wenn kein akuter Bedarf besteht.
- ❏ Wird ein Seitenrahmen in den Freiseitenpuffer eingetragen, wird seine bisherige Zuordnung aus dem enstsprechenden Seitendeskriptor entfernt.
- ❏ Auf eine Seite im Freiseitenpuffer darf von dem bisherigen "Besitzer" (Prozess) noch weiterhin zum Lesen und Schreiben zugegriffen werden, bis der Seitenrahmen tatsächlich für eine neue Seite genutzt wird.
- ❏ Auf eine Seite im Freiseitenpuffer darf von dem bisherigen "Besitzer" (Prozess) noch weiterhin, allerdings nur zum Schreiben, zugegriffen werden, bis der Seitenrahmen tatsächlich für eine neue Seite genutzt wird.

d) Welche Aussage zum Thema Adressraumschutz ist **richtig**? 2 Punkte

- ❏ Beim Adressraumschutz durch Eingrenzung ist es prinzipbedingt nicht möglich, dass mehrere Prozesse auf ein Stück gemeinsamen Speichers zugreifen.
- ❏ Beim Adressraumschutz durch Abteilung wird der logische Adressraum in mehrere Segmente mit unterschiedlicher Semantik unterteilt.
- ❏ Bei allen Verfahren des Adressraumschutzes führt jeder Zugriff auf eine ungültige Speicheradresse zu einem Trap.
- ❏ In einem segmentierten Adressraum kann zur Laufzeit kein weiterer Speicher mehr dynamisch nachgefordert werden.

e) Man unterscheidet die Begriffe Programm und Prozess. Welche der folgenden Aussagen zu diesem Themengebiet ist richtig? 2 Punkte

- ❏ Der Prozess ist der statische Teil (Rechte, Speicher, etc.), das Programm der aktive Teil (Programmzähler, Register, Stack).
- ❏ Wenn ein Programm nur einen aktiven Ablauf enthält, nennt man diesen Prozess, enthält das Programm mehrere Abläufe, nennt man diese Threads.
- ❏ Ein Prozess ist ein Programm in Ausführung - ein Prozess kann aber auch mehrere verschiedene Programme ausführen
- ❏ Ein Prozess kann mit Hilfe von Threads mehrere Programme gleichzeitig ausführen.

f) Welche Aussage zum Thema Prozesse/Threads ist **richtig**? 2 Punkte

- ❏ Die Umschaltung schwergewichtiger Prozesse kann nur mit Hilfe des Betriebssystems erfolgen.
- ❏ Eine gleichzeitige Ausführung von mehreren schwergewichtigen Prozessen auf unterschiedlichen Prozessoren ist nicht möglich.
- ❏ Die Einlastung eines federgewichtigen Prozesses ist eine privilegierte Operation und erfordert Unterstützung des Betriebssystems.
- ❏ Leichtgewichtige Prozesse bedingen den Einsatz von verdrängenden Schedulingverfahren.

g) Welche der folgenden Aussagen zum Thema Adressräume ist richtig? 2 Punkte

- ❏ Der logische Adressraum ist ebenso wie der physikalische Adressraum durch die gegebene Hardwarekonfiguration definiert.
- ❏ Der virtuelle Adressraum eines Prozesses kann nie größer sein als der physikalisch vorhandene Arbeitsspeicher.
- ❏ Der physikalische Adressraum ist durch die gegebene Hardwarekonfiguration definiert.
- ❏ Die maximale Größe des virtuellen Adressraums kann unabhängig von der verwendeten Hardware frei gewählt werden.

h) Welche der folgenden Aussagen zum Thema Seitenfehler (*page fault*) ist richtig? 2 Punkte

- ❏ Ein Seitenfehler zieht eine Ausnahmebehandlung nach sich. Diese wird dadurch ausgelöst, dass die MMU das Signal *SIGSEGV* an den aktuell laufenden Prozess schickt.
- ❏ Wenn der gleiche Seitenrahmen in zwei verschiedenen Seitendeskriptoren eingetragen wird, löst dies einen Seitenfehler aus (Gefahr von Zugriffskonflikten!).
- ❏ Seitenfehler können auch auftreten, obwohl die entsprechende Seite gerade im physikalischen Speicher vorhanden ist.
- ❏ Ein Seitenfehler wird ausgelöst, wenn der Offset in einer logischen Adresse größer als die Länge der Seite ist.

i) Bei der Behandlung von Ausnahmen (Traps oder Interrupts) unterscheidet man zwei Bearbeitungsmodelle. Welche Aussage hierzu ist richtig? 2 Punkte

- ❏ Das Wiederaufnahmemodell dient zur Behandlung von Interrupts (Fortführung des Programms nach einer zufällig eingetretenen Unterbrechung). Bei einem Trap ist das Modell nicht sinnvoll anwendbar, da ein Trap deterministisch auftritt und damit eine Wiederaufnahme des Programms sofort wieder den Trap verursachen würde.
- ❏ Nach dem Beendigungmodell werden Interrupts bearbeitet. Gibt man z. B. CTRL-C unter UNIX über die Tastatur ein, wird ein Interrupt-Signal an den gerade laufenden Prozess gesendet und dieser dadurch beendet.
- ❏ Interrupts dürfen auf keinen Fall nach dem Beendigungsmodell behandelt werden, weil überhaupt kein Zusammenhang zwischen dem unterbrochenen Prozess und dem Grund des Interrupts besteht.
- ❏ Das Betriebssystem kann Interrupts, die in ursächlichem Zusammenhang mit dem gerade laufenden Prozess stehen, nach dem Beendigungsmodell behandeln, wenn eine sinnvolle Fortführung des Prozesses nicht mehr möglich ist.

j) Welche Aussage zu nicht-blockierender Synchronisation ist richtig? 2 Punkte

- ❏ Bei allen nicht-blockierenden Verfahren tritt das ABA-Problem auf.
- ❏ Verfahren zur nicht-blockierenden Synchronisation benötigen keine spezielle Hardware-Unterstützung.
- ❏ In vielen Fällen sind die Algorithmen bei Verwendung nicht-blockierender Synchronisation einfacher zu beschreiben als bei blockierender Synchronisation.
- ❏ Bei nicht-blockierenden Verfahren können keine Verklemmungen auftreten.

k) Wodurch kann Nebenläufigkeit in einem System entstehen? 2 Punkte

- ❏ durch Seitenflattern
- ❏ durch langfristiges Scheduling
- ❏ durch Compiler-Optimierungen
- ❏ durch Interrupts

Aufgabe 1.2: Mehrfachauswahl-Fragen (8 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe sind jeweils m Aussagen angegeben, n ($0 \leq n \leq m$) Aussagen davon sind richtig. Kreuzen Sie **alle richtigen** Aussagen an. Jede korrekte Antwort in einer Teilaufgabe gibt einen halben Punkt, jede falsche Antwort einen halben Minuspunkt. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, d. h. falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten!

a) Welche der folgenden Aussagen zu UNIX-Dateisystemen sind richtig?

- ❍ Dateien (Name, Attribute und Inhalt) werden in Dateikatalogen abgespeichert.
- ❍ Beim Anlegen einer Datei wird die maximale Größe festgelegt. Wird sie bei einer Schreiboperation überschritten, wird ein Fehler gemeldet.
- ❍ Ein Dateikopf ist eine Verwaltungsstruktur, die vorne in der Datei gespeichert wird.
- ❍ Ein *hard link* ist ein Verweis aus einem Katalog auf eine Dateiverwaltungsstruktur (*Inode*).
- ❍ Auf einen Katalog darf immer nur ein *hard link* existieren.
- ❍ Man darf einen *hard link* auf eine Datei erzeugen, auch wenn man keine Zugriffsrechte auf diese Datei hat.
- ❍ In einem Verzeichnis darf es nicht mehrere Einträge mit gleichem Namen geben, selbst wenn diese auf verschiedene *Inodes* verweisen würden.
- ❍ Obwohl eine Datei gelöscht wurde, kann es *symbolic links* geben, die noch auf sie verweisen.

b) Welche der folgenden Aussagen zum Thema Einplanungsverfahren sind richtig?

4 Punkte

- ❍ *First-Come-First-Served* ist nur im Stapelbetrieb bei lang laufenden Aufträgen sinnvoll einsetzbar.
- ❍ *Round-Robin* benachteiligt E/A-intensive Prozesse zu Gunsten von rechenintensiven Prozessen
- ❍ *Virtual-Round-Robin* benachteiligt E/A-intensive Prozesse zu Gunsten von rechenintensiven Prozessen
- ❍ Prioritäten-basierte Verfahren sind auch für interaktiven Betrieb gut geeignet.
- ❍ Zur Realisierung von verdrängenden Einplanungsverfahren wird Hardwareunterstützung durch eine MMU benötigt.
- ❍ *Shortest-Process-Next* ist nur theoretisch interessant, weil die Länge der nächsten Rechenphase (CPU-Stoß) in der Praxis nicht abgeschätzt werden kann.
- ❍ *Feedback*-Strategien sind eine Erweiterung von *Round-Robin* um Prioritätsebenen.
- ❍ Echtzeitverarbeitung ist nur mit geeigneten Prioritäten-setzenden Verfahren realisierbar.

Aufgabe 2: videostreamer (60 Punkte)

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

Schreiben Sie ein mehrfädiges Programm **videostreamer**, das auf Anfrage über eine TCP-Verbindung transcodierte Videodateien ausliefert beziehungsweise die vorhandenen Videodateien auflistet. Die Videodateien liegen in unkomprimierter Form im aktuellen Arbeitsverzeichnis.

Beim Start des Programms wird ein Thread-Pool mit 16 Arbeiter-Threads erzeugt, die die Funktion **doTranscoding()** ausführen. Diese Threads erhalten ihre Aufträge in **VideoFrame**-Strukturen über einen Ringpuffer (Implementierung vorgegeben) mit einer Kapazität von 32 Elementen.

Das Programm nimmt Verbindungen auf Port 7083 entgegen. Jede angenommene Verbindung wird durch einen neu zu startenden Thread bedient (Thread-Funktion **void *serve(FILE *client)**). Der Thread liest von der Verbindung eine Zeile mit dem Kommando des Clients (Maximallänge: 2048 Zeichen), bearbeitet es und schließt die Verbindung.

Wird das Kommando **"LIST"** empfangen, ruft der Thread die Funktion **int listDir(FILE *client)** auf, die die Namen aller Einträge im aktuellen Arbeitsverzeichnis zeilenweise auf den Socket ausgibt. Einträge, deren Name mit einem Punkt beginnt, werden ignoriert. Im Erfolgsfall liefert **listDir()** 0 zurück, im Fehlerfall -1.

Wird das Kommando **"PLAY *Dateiname*"** empfangen, ruft der Thread die Funktion **streamVideo()** mit dem angeforderten Dateinamen auf. Jedes andere Kommando ist ungültig und wird mit der Zeile **"INVALID"** beantwortet. Tritt bei der Anfrageverarbeitung ein Fehler auf, erhält der Client die Antwortzeile **"FAILED"** und die Verbindung wird getrennt.

- Funktion **int streamVideo(const char fileName[], FILE *client)**:
  Erstellt und initialisiert zunächst eine Struktur vom Typ **VideoJob**, die die Anfrage repräsentiert. Anschließend wird die Videodatei geöffnet und nacheinander die Video-Frames (Einzelbilder) eingelesen, wobei alle Frames gleich groß sind (Größe: **VID_FRAME_SIZE**). Eventuelle unvollständige Frames am Dateiende werden verworfen. Jeder eingelesene Video-Frame erhält eine laufende Nummer und wird in eine Kontrollstruktur vom Typ **VideoFrame** eingehängt. Die Kontrollstrukturen werden zur Weiterverarbeitung über den Ringpuffer an den Thread-Pool übergeben. Nachdem alle Frames eingelesen und übergeben wurden, wird gewartet, bis der letzte Frame transcodiert und ausgegeben ist, und danach die **VideoJob**-Struktur zerstört. Der Rückgabewert 0 signalisiert Erfolg, -1 einen Fehler.
- In der Funktion **void *doTranscoding(void *arg)** entnimmt der Arbeiter-Thread eine **VideoFrame**-Kontrollstruktur aus dem Ringpuffer und ruft die im Modul **transcode.o** vorgegebene Funktion **size_t transcode(void *outData, const void *inData, size_t inSize)** auf. Diese Funktion führt das Transcodieren der übergebenen Frame-Daten durch, schreibt das Ergebnis in den Puffer **outData** und gibt die tatsächliche Größe des Ausgabe-Frames zurück. Es ist garantiert, dass die Ausgabe-Größe die Eingabe-Größe nicht übersteigt.
  Der Ausgabe-Frame wird nun auf der Socketverbindung ausgegeben. Dabei ist darauf zu achten, dass die Frames in der richtigen Reihenfolge ausgegeben werden.

Für die Reihenfolge-Synchronisation soll ein Modul **atomiccounter** benutzt und implementiert werden, das die folgende Schnittstelle bereitstellt:

```
int  acInit(AtomicCounter *c, int i); - initialisiert den Zähler; liefert 0 bzw. bei Fehler -1
void acDestroy(AtomicCounter *c);     - zerstört den atomaren Zähler
void acIncrement(AtomicCounter *c);   - inkrementiert den Zählerwert atomar um 1
void acWait(AtomicCounter *c, int v); - blockiert den aufrufenden Thread so lange, bis der
                                        Zähler den Wert v erreicht hat.
```

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 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 gegebenenfalls auch weitere Funktionen benötigen.


```
#include "atomiccounter.h"
#include "transcode.h"
#include <dirent.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>

// Konstanten
#define MAX_REQUEST_LEN 2048
#define VID_FRAME_SIZE   256

// Struktur-Deklarationen
typedef struct VideoJob {
    FILE *out;                      // Ausgabe-Stream
    AtomicCounter processedFrames; // Schon verarbeitete Frames
} VideoJob;

typedef struct VideoFrame {
    int  id;                    // Fortlaufende Frame-Nummer
    char data[VID_FRAME_SIZE]; // Frame-Daten
    VideoJob *job;              // Job, zu dem der Frame gehoert
} VideoFrame;

// Externe Funktionen des Ringpuffer-Moduls
BNDBUF *bbCreate(size_t size);
void bbDestroy(BNDBUF *bb);
void bbPut(BNDBUF *bb, VideoFrame *frame);
VideoFrame *bbGet(BNDBUF *bb);

// Hilfsfunktion
static int die(const char message[]) {
    perror(message);
    exit(EXIT_FAILURE);
}

// Globale Variablen, Funktionsdeklarationen usw.
```

```
// Funktion main()

    // Allgemeine Vorbereitungen

    // Socket erstellen und auf Verbindungsannahme vorbereiten
```

I:


```
    // Verbindungen annehmen und in neuem Thread abarbeiten

} // Ende Funktion main()
```

O:

```
// Thread-Funktion serve()
```

S:


```
// Funktion listDir()
```

D:

```
// Funktion streamVideo()
```

V:


```
// Thread-Pool-Funktion doTranscoding()
```

T:

```
/*************************************************************
 * Modul atomiccounter
 *************************************************************/

typedef struct AtomicCounter {
    volatile int    value;
    pthread_mutex_t mutex;
    pthread_cond_t  cond;
} AtomicCounter;
```

// Funktion acInit()

// Funktion acDestroy()

// Funktion acIncrement()

// Funktion acWait()

A:

Aufgabe 3: (18 Punkte)

Zur Verwaltung von freiem Speicher (z. B. bei der Verwaltung von Segmenten im physikalischen Hauptspeicher oder auch in Funktionen wie `malloc()` und `free()` zur feingranularen Verwaltung von Speicherbereichen) gibt es verschiedene Strategien bei der Speicherzuteilung.

Nehmen Sie einen Speicher von 1024 Bytes an und gehen Sie davon aus, dass die Freispeicher-Verwaltungsstrukturen separat liegen. Initial seien zwei Datenblöcke (60 und 40 Bytes groß) vergeben. Ein Programm führt nun die im Folgenden angegebenen `malloc()`-Aufrufe aus.

a) Geben Sie für das *worst-fit*-Verfahren an, welches Ergebnis diese Aufrufe jeweils zurückliefern, und skizzieren Sie, wie die Freispeicher-Verwaltungsstrukturen initial und nach jedem Schritt aussehen.
Die initial vergebenen Blöcke liegen an den Adressen 0 (60 Bytes) und 360 (40 Bytes). Sie werden ebenfalls mit dem *worst-fit*-Verfahren verwaltet und sind in den Skizzen zu berücksichtigen.

worst-fit

initial

① `malloc(300);`

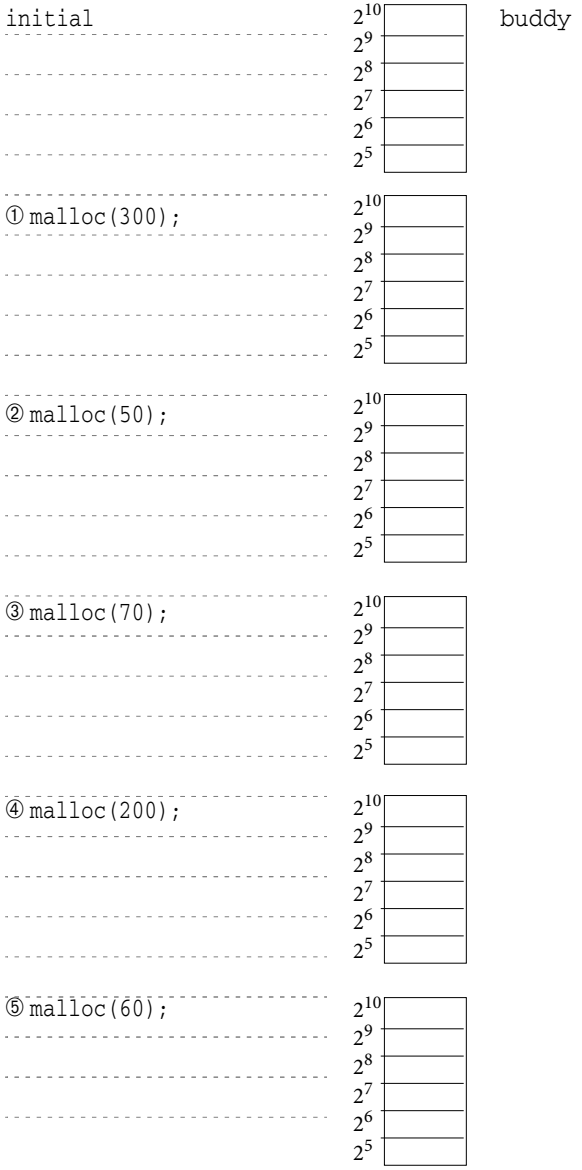
② `malloc(50);`

③ `malloc(70);`

④ `malloc(200);`

⑤ `malloc(60);`

b) Geben Sie nun entsprechend zu Teilaufgabe a) für das *Buddy*-Vefahren die Ergebnisse der Aufrufe und die Freispeicher-Verwaltungsstrukturen (beschränken Sie sich auf den wesentlichen Ausschnitt der Strukturen!) an.
Die initial vergebenen Blöcke liegen an den Adressen 0 und 64. Auch sie werden mit dem *Buddy*-Verfahren verwaltet und sind in der Skizze zu berücksichtigen.



Aufgabe 4: (12 Punkte)

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