Aufgabe 1: Ankreuzfragen (30 Punkte)

1) Einfachauswahlfragen (22 Punkte)

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

Wollen Sie eine 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) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (Symbolic Links) und feste Links (Hard Links) auf Dateien. Welche Aussage ist richtig? 2 Punkte

- ☐ Für jede reguläre Datei existiert mindestens ein Hard-Link im selben Dateisystem.
- ☐ Wird der letzte Symbolic Link auf eine Datei gelöscht, so wird auch die Datei selbst gelöscht.
- ☐ Ein Symbolic Link kann nicht auf Dateien anderer Dateisysteme verweisen.
- ☐ Ein Hard Link kann nur auf Verzeichnisse verweisen, nicht jedoch auf Dateien.

b) Ausnahmesituationen bei einer Programmausführung werden in die beiden Kategorien Trap und Interrupt unterteilt. Welche der folgenden Aussagen ist zutreffend? 2 Punkte

- ☐ Ein Trap signalisiert einen schwerwiegenden Fehler und führt deshalb immer zur Beendigung des unterbrochenen Programms.
- ☐ Ein durch einen Interrupt unterbrochenes Programm darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.
- ☐ Obwohl Traps immer synchron auftreten, kann es im Rahmen ihrer Behandlung zu Wettlaufsituationen mit dem unterbrochenen Programm kommen.
- ☐ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.

c) Man unterscheidet kurz-, mittel- und langfristige Prozesseinplanung. Welche Aussage hierzu ist richtig? 2 Punkte

- ☐ Wenn ein Prozess auf einen Seitenfehler (page fault) trifft, wird er im Rahmen der kurzfristigen Einplanung in den Zustand „schwebend bereit“ überführt, weil er ja unmittelbar nach dem Einlagern der Seite wieder weiterlaufen kann.
- ☐ Wenn ein Prozess auf einen Seitenfehler (page fault) trifft, wird er im Rahmen der kurzfristigen Einplanung immer in den Zustand „blockiert“ überführt, bis die Seite eingelagert wurde.
- ☐ Wenn der Adressraum eines laufbereiten Prozesses aufgrund von Speichermangel ausgelagert wird („swap-out“), wird der Prozess im Rahmen der mittelfristigen Einplanung in den Zustand „blockiert“ überführt, bis die Daten wieder eingelagert werden.
- ☐ Wenn ein Prozess auf Daten von der Platte warten muss, wird er in den Zustand „blockiert“ versetzt.

d) Man unterscheidet zwischen privilegierten und nicht-privilegierten Maschinenbefehlen. Welche Aussage ist richtig?

2 Punkte

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

e) Was passiert, wenn Sie in einem C-Programm über einen ungültigen Zeiger versuchen auf Speicher zuzugreifen?

2 Punkte

- ☐ Das Betriebssystem erkennt die ungültige Adresse bei der Weitergabe des Befehls an die CPU (partielle Interpretation) und leitet eine Ausnahmebehandlung ein.
- ☐ Der Compiler erkennt die problematische Code-Stelle und generiert Code, der zur Laufzeit bei dem Zugriff einen entsprechenden Fehler auslöst.
- ☐ Beim Zugriff über den Zeiger muss die MMU die erforderliche Adressumsetzung vornehmen, erkennt 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.

f) Welche der folgenden Aussagen zum Thema Dateispeicherung sind richtig?

2 Punkte

- ☐ Bei indizierter Speicherung von Dateien ensteht externer Verschnitt auf der Platte.
- ☐ Festplatten eignen sich besser für sequentielle als für wahlfreie Zugriffsmuster.
- ☐ Eine Datei in einem Winows-NT-Dateisystem kann nur genau einen Dateinamen haben, da dieser in ihrem Master-File-Table-Eintrag gespeichert ist.
- ☐ Da SSDs ohne mechanische Komponenten auskommen, gibt es auch durch häufige Schreiboperationen keinen Verschleis.

g) Welche Aussage zum Thema „Aktives Warten“ ist richtig?

2 Punkte

- ☐ Zur Implementierung einer Schlossvariable mit aktivem Warten ist keine Unterstützung durch das Betriebssystem notwendig.
- ☐ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit.
- ☐ Bei verdrängenden Scheduling-Strategien verzögert aktives Warten nur den betroffenen Prozess, behindert aber nicht andere.
- ☐ Aktives Warten sollte bei einer nicht-verdrängenden Scheduling-Strategie auf einem Monoprozessorsystem dem passiven Warten vorgezogen werden.

h) Welche Aussage zum Thema Seitenersetzungsstrategien ist richtig?

2 Punkte

- ☐ Die Ersetzungsstrategie FIFO eignet sich vor allem für Programme, die nicht-sequentielle Speicherzugriffsmuster aufweisen.
- ☐ Bei der Ersetzungsstrategie LFU ist der Zeitpunkt des letzten Zugriffes das ausschlaggebende Kritierum für die Ersetzung einer Seite.
- ☐ Die Ersetzungsstrategie OPT wird in der Praxis gerne angewendet, da Wissen über das zukünftige Verhalten des Gesamtsystems berücksichtigt wird.
- ☐ Bei der Ersetzungsstrategie LRU wird diejenige Seite ersetzt, seit deren letztem Zugriff die längste Zeit vergangen ist.

i) Welche der folgenden Aussagen über Einplanungsverfahren ist richtig?

2 Punkte

- ☐ Beim Einsatz präemptiver Einplanungsverfahren kann laufenden Prozessen die CPU nicht entzogen werden.
- ☐ Asymmetrische Einplanungsverfahren können ausschließlich auf asymmetrischen Multiprozessor-Systemen zum Einsatz kommen.
- ☐ Bei kooperativer Einplanung kann es zur Monopolisierung der CPU kommen.
- ☐ Probabilistische Einplanungsverfahren müssen die exakten CPU-Stoßlängen aller im System vorhandenen Prozesse kennen.

j) Welches Signal wird bei einer Speicherschutzverletzung versendet?

2 Punkte

- ☐ SIGKILL
- ☐ SIGSEGV
- ☐ SIGTERM
- ☐ SIGABORT

k) Welche Aussage zu den Eigenschaften eines Journaling-Filesystems ist richtig?

2 Punkte

- ☐ Es wird immer zuerst die Änderung im Dateisystem auf der Platte duchgeführt und anschließend zur Absicherung ein entsprechender Eintrag in die Log-Datei geschrieben.
- ☐ Die Einträge in der Protokolldatei müssen immer auch Informationen zu einem Undo und Redo der Transaktion enthalten.
- ☐ 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, wird die betroffene Datei gelöscht.
- ☐ Je größer eine Platte ist, desto länger dauert der Reparaturvorgang eines Journaling-Filesystems nach einem Systemabsturz.

2) Mehrfachauswahlfragen (8 Punkte)

Bei den Mehrfachauswahlfragen in dieser Aufgabe sind jeweils m Aussagen angegeben, davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie alle richtigen Aussagen an.
Jede korrekte Antwort in einer Teilaufgabe gibt einen Punkt, jede falsche Antwort einen 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 Aussage zum Thema RAID ist richtig?

4 Punkte

- ○ Bei RAID 4 enthält eine Platte die Paritätsinformationen, die anderen Platten enthalten Daten.
- ○ Bei RAID 4 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.
- ○ Bei RAID 0 führt der Ausfall einer der beteiligten Platten nicht zu Datenverlust.
- ○ Bei RAID 1 werden die Datenblöcke über mehrere Festplatten verteilt und repliziert gespeichert.
- ○ Bei RAID 1 wird beim Lesen ein Geschwindigkeitsvorteil erzielt.
- ○ Bei RAID 5 liegen die Paritätsinformationen auf einer dedizierten Platte.
- ○ Bei RAID 5 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.
- ○ Bei RAID 0 werden die Datenblöcke über mehrere Festplatten verteilt und repliziert gespeichert.

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

4 Punkte

- ○ Verdrängende Prozesseinplanung bedeutet, dass das Eintreten des erwarteten Ereignisses unmittelbar die Einlastung des wartenden Prozesses bewirkt.
- ○ Ein Prozess kann sich in realen Systemen nie im Zustand beendet befinden, da bei seiner Terminierung sämtliche Betriebsmittel freigegeben werden und damit auch der Prozess selbst verschwindet.
- ○ Prozesse im Zustand blockiert oder bereit können unmittelbar in den Zustand gestoppt überführt werden.
- ○ Einplanungsverfahren lassen sich in drei Kategorien einteilen: federgewichtig, leichtgewichtig und schwergewichtig.
- ○ Ein Prozess, der sich im Zustand laufend befindet, kann nicht direkt in den Zustand schwebend blockiert überführt werden.
- ○ Prozesse im Zustand gestoppt sind der langfristigen Einplanung zuzuordnen.
- ○ Für die mittelfristige Einplanung muss das Betriebssystem die Umlagerung (engl. swapping) von kompletten Programmen bzw. logischen Adressräumen unterstützen.
- ○ Ein Prozess im Zustand erzeugt kann sich selbst durch die Ausführung des Systemaufrufes exec() in den Zustand bereit überführen.

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

Aufgabe 2: SitzPlatz Planer-Backend (60 Punkte)

Ihre Aufgabe ist es, das Backend einer Anwendung zur Platzreservierung zu schreiben. Jeder Sitzplatz ist dabei eine reguläre Datei, deren Name auf `.slot` endet. Ein Sitzplatz wird über den Dateipfad exklusive dieser spezifischen Endung identifiziert. Ist der Raum belegt, so enthält diese Textdatei den Namen des Belegenden, anderenfalls ist die Datei leer. Um die Performanz zu steigern, wird der Zustand im Arbeitsspeicher zwischengehalten und nur auf expliziten Befehl hin in das Dateissystem zurückgeschrieben.

Das Programm erwartet als Parameter Pfade, die rekursiv nach Reservierungsdateien durchsucht werden. Anschließend werden eingehende Verbindungsanfragen auf TCP-Port 1416 (IPv6) über einen Ringpuffer auf 10 Arbeiterfäden verteilt. Jeder Faden wartet auf neue Verbindungen im Ringpuffer und bearbeitet die dadurch empfangenen Befehle zeilenweise. Eine Zeile ist dabei maximal 1024 Zeichen lang und besteht aus bis zu zwei durch Leerzeichen oder Tabulatoren getrennten Teilen. Der erste Teil identifiziert dabei den gewünschten Sitzplatz, während der zweite den Namen des Nutzers enthält, für den die Reservierung durchgeführt werden soll. Ist kein Nutzer angegeben, so soll der entsprechende Raum freigegeben werden. Zeilen, die weder Sitzplatz noch Nutzer enthalten, sorgen dafür, dass der Reservierungs-Zustand in das Dateisystem gesichert wird.

Implementieren Sie folgende Funktionen:

`int main(int argc, char *argv[])` Sucht in allen als Parameter übergebenen Pfaden nach reservierbaren Plätzen, um diese im Arbeitsspeicher zu verwalten (`load()`). Anschließend werden die 10 Arbeiterfäden (`worker()`) gestartet, sowie ein Socket zum Warten auf eingehende Verbindungen geöffnet. Eingehende Verbindungen werden über einen Ringpuffer zur Bearbeitung an die Arbeiterfäden gegeben.

`void load(const char *path)` Prüft, ob `path` selbst eine Platzreservierung ist (reguläre Datei, deren Name auf `".slot"` endet), oder durchsucht `path` rekursiv, falls es sich um ein Verzeichnis handelt. Platzreservierungen werden durch `add_slot()` behandelt. Fehler in dieser Funktion führen mit einer aussagekräftigen Meldung zur Beendigung des Programms.

`int add_slot(const char *path)` Erstellt eine Datenstruktur (`slot_alloc()`) für die Platzreservierung, die zur Datei `path` gehört, und fügt sie dem Key-Value-Store (`kvs_add()` hinzu. Im Fehlerfall wird `-1` zurückgegeben und der Grund in `errno` gesetzt, anderenfalls `0`.

`void *worker(void *arg)` Erwartet als `arg` ein **`struct`** `params` und entnimmt Verbindungen aus dem enthaltenen Ringpuffer. Anschließend werden die Befehle der Verbindung zeilenweise eingelesen, in die Bestandteile zerteilt und an `command` übergeben. Die zurückgegebe Zeichenkette wird dem Klienten als Antwort geschickt, bevor die nächste Zeile bearbeitet wird.

`const char *command(struct params *p, const char *slot, const char *owner)` Abhängig von den Werten von `slot` und `owner` wird der entsprechende Befehl ausgeführt. Sind beide `NULL`, so soll der Arbeitsspeicherzustand durch `kvs_iter` in die jeweiligen Dateien geschrieben werden. Hierbei muss sichergestellt sein, dass andere Fäden mit Änderungen an den Reservierungen warten, bis alles zurückgeschrieben wurde. Anderenfalls wird der entsprechende `slot` im Key-Value-Store gesucht (`kvs_get()`) und die Reservierung für `owner` ausgeführt (`slot_own()`). Im Erfolgsfall ist der Rückgabewert `"success"`, andern falls eine aussagekräftige, statische Zeichenkette.

`void save(const char *key, void *value)` Sichert die entsprechende Platzreservierung, kann daher als Callback-Funktion für `kvs_iter()` verwendet werden. Dabei ist `value` vom Typ **`struct`** `slot`. Fehler werden durch eine passende Meldung auf dem Fehlerausgabekanal behandelt, sollen aber nicht zur Beendigung des Programms führen.

Hinweise: Da es sich um das Backend handelt, können Sie vereinfachend davon ausgehen, dass keine überlangen oder ungültigen Zeilen empfangen werden.

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

```
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <dirent.h>
#include <unistd.h>
#include <netinet/in.h>
#include <fnmatch.h>
#define CMD_MAX_LEN  1024
#define THREADS         10
#define PORT            1416
void die(const char *msg) { ... }
struct slot { ... };
struct params { SEM *active, *sync; BNDBUF *clients; };

// stores @value in the store associated with @key. Never fails!
void kvs_add(const char *key, void *value);
// returns the value associated with @key if it is in the store or NULL.
void *kvs_get(const char *key);
// calls @callback for every item in the key value store
void kvs_iter(void (*callback)(const char *key, void *value));

// creates and returns a new ‘struct slot‘ reserved by @owner (see slot_own).
// On error, errno is set and NULL returned.
struct slot *slot_alloc(const char *owner);
// atomically reserves @slot for @owner, or releases it if @owner is NULL.
// On success, 0 is returned, otherwise -1 and errno is set appropriately.
// This may be if the slot is already reserved and @owner is not NULL, or
// already free if @owner is NULL, as well as related to required memory
// allocations (@owner is dublicated).
int slot_own(struct slot *slot, const char *owner);

// creates a new semaphore with initial value of @count.
// returns NULL on error and sets errno.
SEM *semCreate(int count);
void P(SEM *);
void V(SEM *);

// create a new bounded buffer of given @size.
// returns NULL on error and sets errno
BNDBUF *bbCreate(size_t size);
// adds @value into @bb. Blocks if necessary and never fails.
void bbPut(BNDBUF *bb, int value);
// retrieves next value from @bb. Blocks if necessary and never fails.
int bbGet(BNDBUF *bb);
```

```
int main(int argc, char *argv[]) {
  if (argc < 2) {
    fprintf(stderr, "USAGE: %s [paths...]\n", argv[0]);
    return EXIT_FAILURE;
  }
  // Parameter verarbeiten


  // Netzwerkkommunikation aufsetzen
```

```
  // Thread-Pool aufsetzen

  // Anfragen annehmen

}
```

M:

```
static void load(const char *path) {
  // Pfad-Type prüfen
```

```
}
```


L:

```
static int add_slot(const char *path) {
```

```
}
```

A:

```
static void save(const char *key, void *value) {
```

```
}
```

S:

```
static void *worker(void *arg) {
```

```
    // Befehle auslesen und bearbeiten
```

```
}
```

W:

```
const char *command(struct params *p, const char *slot,
                    const char *owner) {
  if (slot != NULL) {
    // Reservierung verwalten
```

```
  } else { // slot == NULL
    // Zustand synchronisieren
  }
}
```

C:

Aufgabe 3: Ausnahmen (17 Punkte)

1) Nennen Sie die zwei Modelle der Ausnahmebehandlung. Wie unterscheiden sich diese Modelle? Nennen Sie für jedes der Modelle je einen Auslösungsgrund. (5 Punkte)

2) Sie haben in der Vorlesung zwei Arten von Ausnahmen von der normalen Programmausführung kennengelernt. Wie lauten diese verschiedenen Arten von Ausnahmen? Nennen sie zwei Eigenschaften, durch die sie sich voneinander unterscheiden. (3 Punkte)

3) Untenstehender Code wird auf einem x86-Linux-System ausgeführt. (6 Punkte)

```
char *ptr = malloc(4294967296);
*ptr = 'c';
```

a) Unter welchen Umständen kann hier ein Fehler auftreten? (1 Punkt)

b) Was für ein Fehler tritt auf? Warum tritt dieser Fehler auf? (2 Punkte)

c) Welche Hardwarekomponente entdeckt diesen Fehler? Wie signalisiert diese Komponente den Fehler an das Betriebssystem? (2 Punkte)

d) Was macht das Betriebssystem mit dem Prozess, der gerade das Programmstück ausführt? (1 Punkt)

4) Zur Behandlung von Unterbrechungen werden Mantelprozeduren (*wrapper*) angesprungen. Welche Arbeitsschritte sind in diesen Prozeduren notwendig, um die Unterbrechung korrekt zu behandeln? (3 Punkte)

Aufgabe 4: Einplanung von Prozessen (13 Punkte)

1) Bei der Einplanung von Prozessen werden Gütemerkmale bzw. Kriterien zur Aufstellung der konkreten Einlastungsreihenfolge von Prozessen unterschieden.

Die Kriterien lassen sich übergreifend in zwei Kategorien einteilen. Beschreiben Sie im folgenden kurz den jeweiligen Fokus der Kategorie. Nennen Sie darüber hinaus je einen Vertreter der entsprechenden Kategorie und beschreiben Sie, welches Verhalten durch dieses Merkmal erreicht werden soll. (6 Punkte)

a) Kategorie der **benutzerorientierten** Kriterien (3 Punkte)

b) Kategorie der **systemorientierten** Kriterien (3 Punkte)

2) Welche Kriterien sind beim **Echtzeitbetrieb** eines System besonders zu bevorzugen? Zu welchen Konflikten zwischen Kritieren kann es dabei kommen? (4 Punkte)

3) Einplanungsverfahren lassen sich ebenfalls verschiedenen Kategorien zuschreiben. Beschreiben Sie kurz die Eigenschaften von **kooperativen** Einplanungsstrategien. Geben Sie darüber hinaus ein Verfahren aus dieser Kategorie an. In welcher Anwendungssituation kann dieses sinnvoll eingesetzt werden? (3 Punkte)