

**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) Wie funktioniert Adressraumschutz durch Eingrenzung? 2 Punkte
- Der Lader positioniert Programme immer so im Arbeitsspeicher, dass unerlaubte Adressen mit nicht-existierenden physikalischen Speicherbereichen zusammenfallen.
  - Begrenzungsregister legen einen Adressbereich im logischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.
  - Begrenzungsregister legen einen Adressbereich im physikalischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.
  - Jedes Programm bekommt zur Ladezeit mehrere Wertepaare aus Basis- und Längenregistern zugeordnet, die die Größe aller Segmente des darin laufenden Prozesses festlegen.
- b) 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.
  - Beim Laden des Programms wird die ungültige Adresse erkannt und der Speicherzugriff durch einen Sprung auf eine Abbruchfunktion ersetzt. Diese Funktion beendet das Programm mit der Meldung "Segmentation fault".
  - Die MMU erkennt die ungültige Adresse bei der Adressumsetzung und löst einen Trap aus.
  - Der Compiler erkennt die problematische Code-Stelle und generiert Code, der zur Laufzeit bei dem Zugriff einen entsprechenden Fehler auslöst.

- c) Ein Prozess wird vom Zustand *blockiert* in den Zustand *bereit* überführt. Welche Aussage passt zu diesem Vorgang? 2 Punkte
- Der Prozess hat auf das Einlesen von Daten von der Festplatte gewartet, die nun verfügbar sind.
  - Ein Prozess, der zu einem früheren Zeitpunkt aufgrund von Speichermangel auf den Hintergrundspeicher ausgelagert wurde, ist nun wieder eingelagert und kann weiterlaufen.
  - Ein anderer Prozess wurde vom Betriebssystem verdrängt und der erstgenannte Prozess wird nun auf der CPU eingelastet.
  - Es ist kein direkter Übergang von *blockiert* nach *bereit* möglich.
- d) Wodurch kann es zu Seitenflattern kommen? 2 Punkte
- Wenn die Zahl der residenten Seiten die Größe des physikalischen Speichers überschreitet.
  - Durch Programme, die eine Defragmentierung auf der Platte durchführen.
  - Wenn ein Prozess zum Weiterarbeiten immer gerade die Seiten benötigt, die durch das Betriebssystem im Rahmen einer globalen Ersetzungsstrategie gerade erst ausgelagert wurden.
  - Wenn zu viele Prozesse im Rahmen der mittelfristigen Einplanung auf den Hintergrundspeicher ausgelagert wurden (swap-out).
- e) Man unterscheidet bei Programmunterbrechungen zwischen Traps und Interrupts. Welche Aussage dazu ist richtig? 2 Punkte
- Die Behandlung eines Traps führt immer zur Beendigung des unterbrochenen Programms, da Traps nur durch schwerwiegende Fehler ausgelöst werden.
  - Da das Betriebssystem nicht vorhersagen kann, wann ein Benutzerprogramm einen Systemaufruf absetzt, sind Systemaufrufe als Interrupts zu klassifizieren.
  - Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Traps immer an den gleichen Stellen auf.
  - Da Interrupts in keinem Zusammenhang mit dem unterbrochenen Programm stehen, muss der Prozessorstatus des unterbrochenen Programms während der Behandlung nicht speziell gesichert werden.

- f) Namensräume dienen u. a. der Organisation von Dateisystemen. Welche Aussage ist richtig? 2 Punkte
- Das Arbeitsverzeichnis eines Prozesses definiert die Wurzel des hierarchisch organisierten Namensraums in einem Dateisystem.
  - In einem hierarchisch organisierten Namensraum dürfen gleiche Namen in unterschiedlichen Kontexten enthalten sein.
  - Flache Namensräume erlauben pro Benutzer nur einen Kontext.
  - Hierarchische Namensräume werden erzeugt, indem man in einem Kontext symbolische Verweise auf Dateien einträgt.
- g) Welche Seitennummer und welcher Versatz gehören bei einstufiger Seitennummerierung und einer Seitengröße von 2048 Bytes zu folgender logischer Adresse: 0xba1d 2 Punkte
- Seitennummer 0xb, Versatz 0xa1d
  - Seitennummer 0x17, Versatz 0x21d
  - Seitennummer 0xba, Versatz 0x1d
  - Seitennummer 0x2e, Versatz 0x21d
- h) Welche Aussage zu Programmbibliotheken ist richtig? 2 Punkte
- Statische Bibliotheken können in C nicht implementiert werden.
  - Eine Änderung am Code einer statischen Bibliothek (z. B. Bugfixes) erfordert kein erneutes Binden der Programme, die diese Bibliothek benutzen.
  - Eine statische Bibliothek, mit der ein Programm gebunden wurde, muss zum Ladezeitpunkt des Programms nicht mehr als eigenständige Datei im Dateisystem vorhanden sein.
  - Beim Binden mit einer statischen Bibliothek werden in einem Programm nur Verweise auf verwendete Symbole der Bibliothek angelegt.

- i) Welche Aussage über Einplanungsverfahren ist richtig? 2 Punkte
- Bei kooperativen Verfahren können Prozesse die CPU nicht monopolisieren.
  - In einem asymmetrischen Multiprozessorsystem ist der Einsatz von asymmetrischen Verfahren zur Planung obligatorisch.
  - Im Round-Robin-Verfahren nutzen E/A-intensive Prozesse die ihnen zugeteilte Zeitscheibe immer voll aus.
  - Probabilistische Verfahren eignen sich besonders für den Einsatz in strikten Echtzeitsystemen, da sie die Einhaltung von Zeitgarantien sicherstellen.
- j) Man unterscheidet die Begriffe Programm und Prozess. Welche der folgenden Aussagen zu diesem Themengebiet ist richtig? 2 Punkte
- Ein Programm kann immer nur von einem Prozess gleichzeitig ausgeführt werden.
  - Das Programm ist der statische Teil (Rechte, Speicher, etc.), der Prozess 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 während seiner Lebenszeit auch mehrere verschiedene Programme ausführen.
- k) Welche der folgenden Aussagen über UNIX-Dateisysteme ist richtig? 2 Punkte
- Auf ein Verzeichnis darf immer nur genau ein *hard-link* verweisen.
  - Hard-links* auf Dateien können nur innerhalb des Dateisystems angelegt werden, in dem auch die Datei selbst liegt.
  - In einem Verzeichnis darf es mehrere Einträge mit dem selben Namen geben, falls diese Einträge auf unterschiedliche Dateiköpfe (*inodes*) verweisen.
  - Wenn der letzte *symbolic link*, der auf eine Datei verweist, gelöscht wird, wird auch der zugehörige Dateikopf (*inode*) gelöscht.

**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 zum Thema Seiteneinlagerungs- und Seiteneretzungsstrategien ist richtig? 4 Punkte
- Die Ersetzungsstrategie MIN ist in der Praxis nur schwer realisierbar, weil Wissen über das zukünftige Verhalten des Gesamtsystems notwendig ist.
  - Bei der Verwendung von globalen Seiteneretzungsstrategien sind Seitenfehler vorhersagbar bzw. reproduzierbar.
  - Mit dem Systemaufruf `free()` kann eine Speicherseite in den Freiseitenpuffer eingefügt werden.
  - Die Ersetzungsstrategie LRU ersetzt die am längsten nicht mehr referenzierte Seite.
  - Bei der Verwendung von lokalen Seiteneretzungsstrategien sind Seitenfehler vorhersagbar bzw. reproduzierbar.
  - Die Ersetzungsstrategie LRU benötigt im Vergleich zu FIFO immer weniger Versuche, bis eine zu ersetzende Seite gefunden werden kann.
  - Lokale Seiteneretzungsstrategien wählen die zu ersetzende Seite immer aus der Menge aller im System verfügbaren Seitenrahmen aus.
  - Bei der Ersetzungsstrategie LFU wird die am seltensten referenzierte Seite aus dem Speicher verdrängt.

- b) Welche der folgenden Aussagen zum Thema Synchronisation sind richtig? 4 Punkte
- Ein Mutex kann ausschließlich für einseitige Synchronisation verwendet werden.
  - Der Einsatz von nicht-blockierenden Synchronisationsmechanismen kann zu Verklemmungen (*dead-locks*) führen.
  - Die V-Operation kann auf einem Semaphor auch von einem Faden aufgerufen werden, der zuvor keine P-Operation auf dem selben Semaphor ausgeführt hat.
  - Ein Anwendungsprozess muss bei der Verwendung von Semaphoren Interrupts sperren, um Probleme durch Nebenläufigkeit zu verhindern.
  - Für nichtblockierende Synchronisation werden spezielle Befehle der Hardware genutzt, die wechselseitigen Ausschluss garantieren.
  - Semaphore können sowohl für einseitige als auch für mehrseitige Synchronisation verwendet werden.
  - Zur Synchronisation eines kritischen Abschnitts ist passives Warten immer besser geeignet als aktives Warten.
  - Gibt ein Faden einen Mutex frei, den er selbst zuvor nicht angefordert hatte, stellt dies einen Programmierfehler dar; der fehlerhafte Prozess sollte dann abgebrochen werden.

**Aufgabe 2: timebox (60 Punkte)**

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

Schreiben Sie ein Programm `timebox`, das auf dem TCP/IPv6-Port 2016 (`LISTEN_PORT`) einen Dienst anbietet, um die Modifikationszeit von auf dem Server abgelegten Dateien abzufragen. Die Dateien liegen im Verzeichnis `/proj/timebox` (`BASEDIR`). Die Abarbeitung parallel eintreffender Anfragen soll von einem Arbeiter-Thread-Pool übernommen werden; die Threads werden über einen entsprechend synchronisierten Ringpuffer mit Verbindungen versorgt.

Ein Client sendet pro Verbindung eine Befehlszeile mit einer Maximallänge von 1024 Zeichen (`MAX_LINE_LEN`) an den Server. Der angebotene Dienst unterstützt genau einen Befehl:

- `TIME relPath` liefert das Änderungsdatum (*modification date*) der Datei am übergebenen Pfad (relativ zum Verzeichnis `BASEDIR`) an den Client zurück.

Der Server soll eine dynamische Aktualisierung der Anzahl aktiver Arbeiter-Threads erlauben. Dazu wird eine Konfigurationsdatei (`CONFIG`) eingelesen, die als einzigen Inhalt die Anzahl der Threads enthält. Nach einer Änderung signalisiert der Server-Administrator dem Serverprozess durch ein `SIGUSR1`-Signal, dass die Konfigurationsdatei einen neuen Wert enthält. Spätestens nach der Annahme der nächsten Verbindung soll die Datei neu eingelesen und die Anzahl der laufenden Threads entsprechend angepasst werden.

Alle Threads des Servers sollen außerdem das Signal `SIGPIPE` ignorieren, damit eine fehlerhafte Verbindung zu einem Client nicht zum Beenden des Servers führt.

Das Programm soll folgendermaßen strukturiert sein:

- Das Hauptprogramm initialisiert zunächst alle benötigten Datenstrukturen und nimmt auf einem Socket Verbindungen an. Wird festgestellt, dass in der Zwischenzeit ein `SIGUSR1`-Signal verarbeitet wurde, soll die Konfigurationsdatei neu eingelesen werden. Eine erfolgreich angenommene Verbindung soll zur weiteren Verarbeitung in den Ringpuffer eingefügt werden.
- Funktion `void parse_config(void)`: Liest die Zahl der Threads aus der Konfigurationsdatei ein. Anschließend soll die Anzahl der laufenden Threads entsprechend der Differenz zum aktuellen Wert angepasst werden. Zum Beenden von Threads sollen dazu spezielle Werte (`DEAD_PILL`) in den Ringpuffer gelegt werden. Die Funktion soll auch zum initialen Starten der Arbeiter-Threads verwendet werden. Tritt beim Einlesen ein Fehler auf oder ist der gelesene Wert kleiner oder gleich 0, soll stattdessen der Standardwert (`DEFAULT_THREADS`) verwendet werden.
- Funktion `void* thread_handler(void *)`: Hauptfunktion der Arbeiter-Threads. Entnimmt dem Ringpuffer eine Verbindung. Falls der Wert `DEAD_PILL` gelesen wird, soll sich der aktuelle Thread beenden. Ansonsten wird zur weiteren Verarbeitung die Funktion `handleCommand` aufgerufen.
- Funktion `void handleCommand(FILE *rx, FILE *tx)`: Liest die Anfragezeile vom Client aus und überprüft, ob sie dem unterstützten Kommando entspricht. Ist dies der Fall, wird der Zeitstempel der Datei ausgelesen und an den Client gesendet (Formatstring: `%ld`). Bei auftretenden Fehlern wird eine kurze Fehlermeldung an den Client gesendet (Beispiel: "Error: unknown command\n").

Zusätzlich sollen Sie die Funktionen `bbPut(int value)` und `bbGet()` implementieren. Der Ringpuffer soll dabei statisch als Feld der Länge 16 (`BUFFER_SIZE`) angelegt werden. Zur Koordination stehen Ihnen Semaphore mit den Funktionen `semCreate`, `P` und `V` zur Verfügung. Diese Funktionen müssen Sie nicht selbst implementieren. Die Schnittstellen entsprechen dabei dem Modul, das sie aus den Übungen kennen, und sind auf der folgenden Seite am Anfang des Programms deklariert.

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 <arpa/inet.h>
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

```
SEM *semCreate(int initVal);
void P(SEM *sem);
void V(SEM *sem);
```

```
#define BUFFER_SIZE 16
#define MAX_LINE_LEN 1024
static const char BASEDIR[] = "/proj/timebox";
static const char CONFIG[] = "/etc/timebox/config";
static const unsigned int DEFAULT_THREADS = 2;
static const int LISTEN_PORT = 2016;
static const int DEAD_PILL = 0xdeaddead;
```

```
static void die(const char msg[]){
    fprintf(stderr, "Error: %s\n", msg); exit(EXIT_FAILURE);
}
// Funktionsdeklarationen, globale Variablen usw.
```

**// Funktion main()**



**// Socket erstellen und auf Verbindungsannahme vorbereiten**



**// Verbindungen annehmen und bearbeiten**



**// Ende Funktion main**

**// Signalbehandlung für SIGUSR1**



**// Ende Signalbehandlung**



**M:**

// Funktion parse\_config()

// Ende Funktion parse\_config()

// Funktion thread\_handler()

// Ende Funktion thread\_handler()

**// Funktion handleCommand()**

**// Ende Funktion handleCommand()**

**// Funktion bbPut()**

**// Ende Funktion bbPut()**

**// Funktion bbGet()**

**// Ende Funktion bbGet()**



