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) Mit logischen Adressräumen kann man mehrere Zwecke erreichen. Was gehört **nicht** dazu? 2 Punkte

- ❏ Schutzmechanismus: man kann die Auswirkungen von Berechnungsfehlern oder technischen Fehlern begrenzen.
- ❏ Sicherheit: man kann unbefugten Zugriff auf Daten verhindern.
- ❏ Virtualisierung: man kann in einem Programmlauf mehr Speicher nutzen, als physikalisch vorhanden ist.
- ❏ Bessere Verwaltung: Man kann Speicherbereiche mit unterschiedlicher Bedeutung voneinander abgrenzen.

b) Welche der Aussagen bzgl. eines logischen Adressraums, der auf dem Prinzip der Segmentierung aufgebaut wurde, ist richtig? 3 Punkte

- ❏ Alle Segmente eines Prozesses haben die gleiche Länge.
- ❏ Bei der Aus- und Einlagerung von Segmenten zwischen Hauptspeicher und Festplatte muss das Segment immer an die gleiche Hauptspeicherposition eingelagert werden, da sonst die Adressen nicht mehr gültig sind.
- ❏ Die Segmentierung erlaubt bei der Abbildung eines logischen Adressraums keinen Zugriff auf Speicherzellen, die auch Bestandteil von anderen logischen Adressräumen sind (Zugriffschutz).
- ❏ Die Segmentierung schränkt den logischen Adressraum derart ein, dass nur auf gültige Speicheradressen erfolgreich zugegriffen werden kann.

c) Welche Seitennummer und welcher Versatz gehören bei einer Seitengröße von 1024 Bytes zu folgender logischer Adresse: 0xcafe 3 Punkte

- ❏ Seitennummer 0x32, Versatz 0x2fe
- ❏ Seitennummer 0xc, Versatz 0xafe
- ❏ Seitennummer 0x19, Versatz 0x2fe
- ❏ Seitennummer 0xca, Versatz 0xfe

d) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux Filedeskriptors zu? 2 Punkte

- ❏ Ein Filedeskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei, ein Gerät, einen Socket oder eine Pipe benutzen kann.
- ❏ Filedeskriptoren sind Zeiger auf Betriebssystemstrukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ❏ Ein Filedeskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.
- ❏ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Filedeskriptor zum Zugriff zurück.

e) Gegeben sei folgendes Szenario: zwei Fäden werden auf einem Monoprozessorsystem mit der Strategie "First Come First Served" verwaltet. In jedem Faden wird die Anweisung "i++;" auf die gemeinsame, globale Variable i ausgeführt. Welche der folgenden Aussagen ist richtig: 2 Punkte

- ❏ In einem Monoprozessorsystem ohne Verdrängung ist keinerlei Synchronisation erforderlich.
- ❏ Während der Inkrementoperation müssen Interrupts vorübergehend unterbunden werden.
- ❏ Die Inkrementoperation muss mit einer CAS-Anweisung nicht-blockierend synchronisiert werden.
- ❏ Die Operation i++ ist auf einem Monoprozessorsystem immer atomar.

f) In welcher der folgenden Situationen wird ein *laufender* Prozess in den Zustand *blockiert* überführt? 3 Punkte

- ❏ Ein Kindprozess des Prozesses terminiert.
- ❏ Der Prozess hat einen Seitenfehler für eine Seite, die bereits in den Freiseitenpuffer eingetragen, aber noch im Hauptspeicher vorhanden ist.
- ❏ Der Prozess greift lesend auf eine Datei zu und der entsprechende Datenblock ist noch nicht im Hauptspeicher vorhanden.
- ❏ Der Prozess ruft eine V-Operation auf einen Semaphor auf und der Semaphor hat gerade den Wert 0.

g) Für lokale Variablen, Aufrufparameter, etc. einer Funktion wird bei vielen Prozessoren ein sog. Aktivierungsblock (activation record oder stack frame) auf dem Stack angelegt. Welche Aussage ist **richtig**?

3 Punkte

- ❏ Über Zeiger kann man alle Daten des Aktivierungsblocks der aufrufenden Funktion verändern.
- ❏ Nach dem Rücksprung aus einer Funktion sind Zeiger auf die Speicherzellen ihres Aktivierungsblocks nicht mehr gültig. Ein Zugriff über solch einen Zeiger führt dann zu einem Segmentation fault.
- ❏ Bei rekursiven Funktionsaufrufen kann der Aktivierungsblock in jedem Fall wiederverwendet werden weil die gleiche Funktion aufgerufen wird.
- ❏ Der Compiler legt zur Übersetzungszeit fest, an welcher Position im Aktivierungsblock der main-Funktion die globalen Variablen angelegt werden.

h) Was versteht man unter der Second-Chance- (oder Clock-) Policy?

2 Punkte

- ❏ Eine Seitenersetzungsstrategie, bei der jeweils die älteste Seite ausgelagert wird.
- ❏ Eine Seitenersetzungsstrategie, die mit Hilfe eines Referenz-Bits eine einfacher zu implementierende Annäherung an LRU realisiert.
- ❏ Eine Scheduling-Strategie, bei der Prozesse vor der Verdrängung eine zweite Chance erhalten.
- ❏ Eine Speicherallokationsstrategie, bei der im Fehlerfall ein zweiter Allokationsversuch stattfindet.

i) Was versteht man unter RAID 0?

2 Punkte

- ❏ Ein auf Flash-Speicher basierendes, extrem schnelles Speicherverfahren.
- ❏ Datenblöcke werden über mehrere Platten verteilt und repliziert gespeichert.
- ❏ Auf Platte 0 wird Parity-Information der Datenblöcke der Platten 1 - 4 gespeichert.
- ❏ Datenblöcke eines Dateisystems werden über mehrere Platten verteilt gespeichert.

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 Teilaufgabse 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 Threads sind richtig?

4 Punkte

- ❍ Bei User-Threads ist die Schedulingstrategie keine Funktion des Betriebssystemkerns.
- ❍ User-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ❍ Kernel-Threads können Multiprozessoren nicht ausnutzen.
- ❍ Zur Umschaltung von Kernel-Threads ist kein Adressraumwechsel erforderlich.
- ❍ Die Umschaltung von User-Threads ist sehr effizient.
- ❍ Zu jedem Kern-Thread gehört ein eigener Adressraum.
- ❍ Bei Kernel-Threads ist die Schedulingstrategie meist durch das Betriebssystem vorgegeben.
- ❍ Die Umschaltung von Threads muss immer im Systemkern erfolgen (privilegierter Maschinenbefehl).

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

4 Punkte

- ❍ Semaphore sind für einseitige Synchronisation geeignet.
- ❍ Semaphore sind für mehrseitige Synchronisation geeignet.
- ❍ Auch nicht-blockierende Synchronisation kann zu Verklemmungen führen.
- ❍ Monitore sind Datentypen mit impliziten Synchronisationseigenschaften.
- ❍ Einseitige Synchronisation erfordert immer Betriebssystem-Unterstützung.
- ❍ Schlossvariablen sind teuer, da lock() immer zu einem Prozesswechsel führt.
- ❍ Die Synchronisation mit einem Signal-Handler unter LINUX erfolgt vorzugsweise über mehrseitige Synchronisationsverfahren.
- ❍ Sperren von Unterbrechungen ist das beste Verfahren zur Synchronisation mit Unterbrechungsbehandlungsfunktionen.

Aufgabe 2: expload (60 Punkte)

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

Schreiben Sie ein Programm `expload` (extended parallel web loader), das Dateien über http (TCP/IP, Port 80) lädt und lokal abspeichert. Die URLs der zu ladenden Dateien sind in einer Reihe von regulären Dateien (im aktuellen Verzeichnis) vorgegeben, deren Name mit einem *u* beginnt. Für jede *u*-Datei soll ein Unterverzeichnis erzeugt werden (der Name des Verzeichnisses entspricht dem Namen der *u*-Datei, das führende *u* wird durch ein *d* ersetzt), Anschließend werden alle in der *u*-Datei aufgeführten URLs geladen und im zugehörigen Verzeichnis abgelegt.

Das Programm arbeitet mit einem Pool von 20 POSIX-Threads, die parallel Dateien laden. Hierzu übergibt der Hauptthread jeweils die URL (maximale Länge 4096 Zeichen) und den Namen des Zielverzeichnisses (max. Länge 256 Zeichen) über einen Ringpuffer an einen bereiten Thread. Die Synchronisation zwischen Hauptthread und den Pool-Threads erfolgt über die Synchronisation des Ringpuffers. Der Ringpuffer soll statisch als Feld von 20 geeignet zu deklarierenden Strukturen (`struct jobDesc`) angelegt werden. Zur Koordination stehen Ihnen Semaphore mit den Funktionen `sem_init`, `P` und `V` zur Verfügung, Sie müssen diese Funktionen nicht selbst implementieren. Die Schnittstellen sind auf der folgenden Seite am Anfang des Programms deklariert.

Das Programm soll folgendermaßen arbeiten:

- In der main-Funktion wird zunächst die Synchronisation vorbereitet und der Thread-Pool angelegt. Anschließend wird in einer Endlosschleife das aktuelle Verzeichnis nach *u*-Dateien durchsucht, für jede *u*-Datei wird die Funktion `processFile` aufgerufen und die *u*-Datei anschließend gelöscht (unlink(2)). Sie dürfen davon ausgehen, dass in dem Verzeichnis allle mit *u* beginnenden Einträge reguläre Dateien sind. Nachdem das Ende des Verzeichnisses erreicht wurde, wird das Verzeichnis geschlossen und von vorne begonnen.
- Funktion `void processFile(char *fileName)`: für die übergebene Datei *uirgendwas* wird das zugehörende Verzeichnis *dirgendwas* angelegt (mkdir(2)). Anschließend wird die Datei zeilenweise gelesen - in jeder Zeile steht genau eine URL. Jede URL wird zusammen mit dem Namen des Verzeichnisses mit Hilfe der Funktion `void bb_add(char *dirName, char *url)` in den Ringpuffer geschrieben. Die Funktion `bb_add` ist von Ihnen zu implementieren - sie fügt die Übergabeparameter in die nächste freie `jobDesc`-Struktur des Ringpuffers ein!
- Jeder Thread des Thread-Pools führt die Thread-Startfunktion `void *tstart()` aus.
  • `tstart` entnimmt mit Hilfe der Funktion void `bb_get(struct jobDesc *out)` den nächsten verfügbaren Auftrag aus dem Ringpuffer. Die Funktion `bb_get` ist von Ihnen zu implementieren! Die URL kann mit Hilfe der im Codegerüst auf Seite 12 vorgegebenen Funktion `splitURL` zerlegt werden.
  • Anschließend wird eine Verbindung zu dem in der URL angegebenen Server mit Hilfe der Funktion `FILE *connectTo(char *hostname)` (diese Funktion ist teilweise im Codegerüst vorgegeben und zu ergänzen) aufgebaut und es wird in dem in der `jobDesc`-Struktur angegebenen Verzeichnis eine Datei mit dem Namen aus der URL (`urlData.file`) angelegt.
  • Schließlich wird das http-Kommando "`GET` *`path`*`\n`" an den Server geschickt (für *`path`* ist die Pfad-Komponente aus der URL einzusetzen), der Server antwortet mit dem Datei-Inhalt, der in die Zieldatei zu schreiben ist.

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.


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <pthread.h>

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

void die(char *message) {perror(message); exit(EXIT_FAILURE);}

typedef struct urlData {
        char hostname[256];
        char path[4096];
        char file[256];
} urlData;

/* Funktions- und Strukturdeklarationen, globale Variablen */
```

```
/* Funktion main */
```

```
    /* Initialisierungen */
```

```
  /*  Thread-Pool erzeugen */
```

A:


```
  /*  u-Dateien suchen und Auftraege erzeugen */
```

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

D:

```
/* Funktion processFile */
```

F:


```
/* Funktion bb_add */
```

```
/* Funktion bb_get*/
```

```
// Funktion splitUrl unterteilt URL http://host.de/a/b/file.c
// in Komponenten:
// file = file.c, path = /a/b/file.c, hostname = host.de

void splitUrl(char* url, urlData* data) {
    strcpy(data->file, strrchr(url, '/'));
    strcpy(data->path, strchr(url+7,'/'));
    strncpy(data->hostname, url+7, 256);
    *(strchr(data->hostname,'/')) = '\0';
}
```

B:

```
/* Funktion tstart*/
```

T:


```
/* Funktion connectTo*/
 FILE* connectTo(urlData *ud) {
    int sock;
    int gai_ret;
    struct addrinfo *srv_addrs;
    struct addrinfo flags;

    memset(&flags, 0, sizeof(struct addrinfo));
    flags.ai_family = AF_UNSPEC;
    flags.ai_socktype = SOCK_STREAM;
    flags.ai_flags = AI_ADDRCONFIG;

    gai_ret = getaddrinfo(ud->hostname, "80", &flags,  &srv_addrs);
    if(gai_ret != 0) {
        fprintf(stderr, "%s: %s\n", ud->hostname,
                                    gai_strerror(gai_ret));
        exit(EXIT_FAILURE);
    }

/* Verbindung zu erster erreichbarer Addr. aufbauen,
   FILE-Pointer zu Socket erzeugen und zurueckgeben */
```

C:

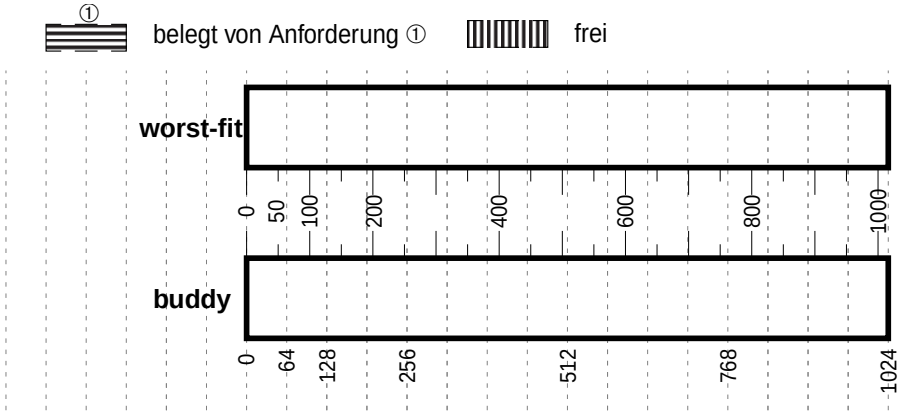
Aufgabe 3: (18 Punkte)

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 70 Byte an Adresse 0 vergeben. Ein Programm führt die im folgenden Bild angegebenen malloc-Aufrufe aus.
Geben Sie für das das *worst-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.

worst-fit **buddy**

① `malloc(300);`
② `malloc(50);`
③ `malloc(70);`
④ `malloc(200);`
⑤ `malloc(60);`



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

worst-fit

init.

A

B

C

Buddy

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

Aufgabe 4: (12 Punkte)

Bei virtuellen Adressräumen können Teile des Speichers auf Hintergrundspeicher ausgelagert sein.

a) Wie wird im System bei einem Speicherzugriff erkannt, dass der entsprechende Speicher ausgelagert ist?

b) Was läuft im System nach dieser Erkennung ab? Beschreiben Sie die einzelnen Schritte, die im Betriebssystem abgewickelt werden, um den Speicher verfügbar zu machen und geben Sie an, welche Prozesszustände der auslösende Prozess dabei jeweils einnimmt.

c) Was versteht man unter Seitenflattern?