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) Welche Aussage zum Thema Prozesszuständen ist richtig? 2 Punkte

- ☐ Pro Prozessorkern kann es stets nur einen laufenden, jedoch mehr als einen bereiten Prozess geben.
- ☐ Ein Prozess kann nicht direkt vom Zustand laufend in den Zustand bereit überführt werden.
- ☐ Ein Prozess kann mit Hilfe von Spezialbefehlen selbst vom Zustand bereit in den Zustand laufend wechseln.
- ☐ Terminiert ein laufender Prozess, so wird er vom Betriebssystem in den Zustand blockiert überführt.

b) Welche der folgenden Aussagen zu statischem bzw. dynamischem Binden ist richtig? 2 Punkte

- ☐ Statisch gebundene Programmdateien sind kleiner als dynamisch gebundene, da mehrfach genutzte Funktionen in einer shared library abgelegt werden und nicht in die ausführbare Datei kopiert werden.
- ☐ Bei dynamischem Binden können Fehlerkorrekturen in Bibliotheken leichter übernommen werden, da nur die Bibliothek selbst neu erzeugt werden muss. Programme, die die Bibliothek verwenden, müssen nicht neu kompiliert und gebunden werden.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst
- ☐ Bei dynamischem Binden müssen zum Übersetzungszeitpunkt alle Adressbezüge vollständig aufgelöst werden

c) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux-Dateideskriptor zu? 2 Punkte

- ☐ Ein Dateideskriptor 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.
- ☐ Dateideskriptoren sind Zeiger auf Betriebssystemstrukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ☐ Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei, ein Gerät, einen Socket oder eine Pipe benutzen kann.
- ☐ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Dateideskriptor zum Zugriff zurück.

d) Welche Aussage zum Thema Adressraumverwaltung ist richtig? 2 Punkte

- ☐ Ein Speicherbereich, der mit Hilfe der Funktion free() freigegeben wurde, verbleibt im logischen Adressraum des zugehörigen Prozesses.
- ☐ Mit malloc() angeforderter Speicher, welcher vor Programmende nicht freigegeben wurde, kann vom Betriebssystem nicht mehr an andere Prozesse herausgegeben werden und ist damit bis zum Neustart des Systems verloren.
- ☐ Mit Hilfe des Systemaufrufes malloc() kann ein Programm zusätzliche Speicherblöcke von sehr feinkörniger Struktur vom Betriebssystem anfordern.
- ☐ Da das Laufzeitsystem auf die Betriebssystemschnittstelle zur Speicherverwaltung zurückgreift, ist die Granularität der von malloc() zurückgegebenen Speicherblöcke vom Betriebssystem vorgegeben.

e) Welche der folgenden Aussagen zum Thema Synchronisation sind richtig? 2 Punkte

- ☐ Auch nicht-blockierende Synchronisation kann zu Verklemmungen führen.
- ☐ Auch bei sehr kurzen Wartezeiten ist passives Warten immer besser geeignet als aktives Warten.
- ☐ Wenn gewöhnliche Anwendungsprozesse Interrupts sperren könnten, wäre kein effektiver Schutz vor CPU-Monopolisierung möglich.
- ☐ Die Verwendung nicht-blockierender Synchronisation benötigt besondere Unterstützung durch das Betriebssystem.

f) Wie wird erkannt, dass eine Seite eines virtuellen Adressraums, auf die ein Maschinenbefehl zugreift, gerade ausgelagert ist? 2 Punkte

- ☐ Im Seitendeskriptor wird ein spezielles Bit geführt, das der MMU zeigt, ob eine Seite eingelagert ist oder nicht. Falls die Seite nicht eingelagert ist, löst die MMU einen Trap aus.
- ☐ Im Seitendeskriptor steht bei ausgelagerten Seiten eine Adresse des Hintergrundspeichers und der Speichercontroller leitet den Zugriff auf den Hintergrundspeicher um.
- ☐ Das Betriebssystem erkennt die ungültige Adresse vor Ausführung des Maschinenbefehls und lagert die Seite zuerst ein bevor ein Trap passiert.
- ☐ Bei Programmen, die in virtuellen Adressräumen ausgeführt werden sollen, erzeugt der Compiler speziellen Code, der vor Betreten einer Seite die Anwesenheit überprüft und ggf. die Einlagerung veranlasst.

g) In welcher der folgenden Situationen wird ein Prozess vom Zustand laufend in den Zustand bereit übeführt? 2 Punkte

- ☐ Der Prozess ruft die Bibliotheksfunktion exit(3) auf.
- ☐ Der Scheduler bewirkt, dass der Prozess durch einen anderen Prozess verdrängt wird.
- ☐ Der Prozess greift lesend auf eine Datei zu und der entsprechende Datenblock ist noch nicht im Hauptspeicher vorhanden.
- ☐ Der Prozess ruft eine P-Operation auf einen Semaphor auf, welcher den Wert 0 hat.

h) In einem Unix/Linux-Dateisystem gibt es symbolische Verknüpfungen/Links (Symbolic Links) und harte Links (Hard Links) auf Dateien. Welche Aussage ist richtig. 2 Punkte

- ☐ Für jede reguläre Datei existiert mindestens ein Hard Link im selben Dateisystem.
- ☐ Ein Hard Link kann nur auf Verzeichnisse, nicht jedoch auf Dateien verweisen.
- ☐ 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.

i) Welche Aussage zu den verschiedenen Gewichtsklassen von Prozessen trifft zu? 2 Punkte

- ☐ Schwergewichtige Prozesse sind die einzige Klasse von Prozessen, die auf einem Multiprozessorsystem echt parallel ausgeführt werden kann, da nur hier jeder Benutzerfaden einem eigenen Kernfaden zugeordnet ist.
- ☐ Federgewichtige Prozesse (user-level threads) blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ☐ Beim Blockieren eines schwergewichtigen Prozesses werden alle anderen schwergewichtigen Prozesse, die das selbe Progamm ausführen, ebenfalls blockiert.
- ☐ Zu jedem leichtgewichtigen Prozess (kernel-level thread) gehört ein eigener, isolierter Adressraum.

j) Welche der folgenden Aussagen zum Thema Seitenfehler (Page Fault) ist richtig? 2 Punkte

- ☐ Wenn der gleiche Seitenrahmen in zwei verschiedenen Seitendeskriptoren eingetragen wird, löst dies einen Seitenfehler aus (Gefahr von Zugriffskonflikten!).
- ☐ Ein Seitenfehler wird ausgelöst, wenn der Offset in einer logischen Adresse größer als die Länge der Seite ist.
- ☐ Ein Seitenfehler zieht eine Ausnahmebehandlung nach sich. Diese wird dadurch ausgelöst, dass die MMU das Signal SIGSEGV an den aktuell laufenden Prozess schickt.
- ☐ Das Auftreten eines Seitenfehlers kann dazu führen, dass der aktuell laufende Prozess in den Zustand beendet überführt wird.

k) Sie kennen den Translation Lookaside Buffer (TLB). Welche Aussage ist richtig? 2 Punkte

- ☐ Der TLB puffert die Ergebnisse der Abbildung von physikalische auf logische Adressen, sodass eine erneute Anfrage sofort beantwortet werden kann.
- ☐ Verändert sich die Speicherabbildung von logischen auf physikalische Addressen aufgrund einer Adressraumumschaltung, so werden auch die Daten im TLB ungültig.
- ☐ Wird eine Speicherabbildung im TLB nicht gefunden, wird vom Prozessor immer eine Ausnahme (Trap) ausgelöst die vom Betriebssystem behandelt wird.
- ☐ Wird eine Speicherabbildung im TLB nicht gefunden, wird der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (Segmentation Fault) abgebrochen.

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) Man unterschiedet zwei Kategorien von Ausnahmesituationen bei einer Programmausführung: Traps und Interrupts. Welche der folgenden Aussagen sind zutreffend?

4 Punkte

- ○ Ein Programm darf im Rahmen einer Trapbehandlung abgebrochen werden.
- ○ Ein durch einen Interrupt unterbrochenes Programm darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.
- ○ Bei einem Trap wird der gerade in Bearbeitung befindliche Maschinenbefehl immer noch vollständig zu Ende bearbeitet, bevor mit der Trapbehandlung begonnen wird.
- ○ Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.
- ○ Die Ausführung einer Ganzzahl-Rechenoperation (z. B. Addition, Division) kann zu einem Trap führen.
- ○ Da Traps immer synchron auftreten, kann es im Rahmen ihrer Behandlung nicht zu Wettlaufsituationen mit dem unterbrochenen Programm kommen.
- ○ Wenn ein Interrupt ein schwerwiegendes Ereignis signalisiert, wird das unterbrochene Programm im Rahmen der Interruptbearbeitung immer abgebrochen.
- ○ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.

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

4 Punkte

- ○ Zur Implementierung von logischen Adressräumen ist spezielle Hardwareunterstützung nötig.
- ○ Ein Seitenfehler (Page Fault) führt abhängig von der Ursache zur Fortsetzung oder Beendigung des aktuellen Prozeses.
- ○ Ein Adressraumwechsel ist eine privilegierte Operation und kann daher nur durch das Betriebssystem vorgenommen werden.
- ○ Nach einem erfolgreichen Aufruf von exec() kann weiterhin auf Datenstrukturen des Aufrufenden Programms zugegriffen werden.
- ○ Da virtuelle Adressräume direkt durch die MMU unterstützt werden, können zur Laufzeit keine zusätzlichen Kosten gegenüber logischen Adressräumen auftreten.
- ○ Speicherbereiche, die im logischen Adressraum zusammenhängend sind, müssen auch im physikalischen Hauptspeicher zusammenhängend sein.
- ○ Bei Adressraumschutz durch Abgrenzung werden die Betriebssystemdaten vor Zugriffen aus dem Anwendungsprogramm geschützt.
- ○ Bei einer seitenorientierten Adressraumorganistaion muss die Größe jeder Seite in der Seitentabelle gespeichert werden.

Aufgabe 2: SINGLE-LINE PARALLEL PALAVER (60 Punkte)

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

Schreiben Sie das Server-Programm `sp2` (**s**ingle-line **p**arallel **p**alaver), um die virtuelle Kommunikation zwischen Tutoren und Studierenden zu ermöglichen. Die Unterscheidung der Rollen erfolgt über den maximal 20 Zeichen langen Nutzernamen, der bei Verbindungsaufbau des Klienten auf IPv6-Port 7370 zu Beginn in einer Zeile geschickt wird. Je eine Tutoren- und eine Studierendenverbindung werden zusammengeschaltet. Darüber erfolgt dann der abwechselnde, zeilenbasierte Nachrichtenaustausch. Tutoren senden dabei immer die erste Zeile.
Die Nutzernamen der Tutoren werden dem Programm `sp2` als Aufrufargumente übergeben.

Implementieren Sie die folgenden Funktionen:

`int main(int argc, char *argv[])`
: Prüft die Argumente und startet den Pool an Arbeiterfäden (`worker`) mit der `NULL`-terminierten Liste der Tutorennamen als Parameter. Angenommene Verbindungen werden als `client_t`-Struktur (`client_create()`) über die `fifo_accept`-Struktur an die Arbeiterfäden weitergegeben. Hierbei ist darauf zu achten, dass selbst wenn sich alle Tutoren verbunden haben, noch ein Arbeiterfaden existiert, der eingehende Verbindungen bearbeitet.

`client_t *client_create(int fd)`
: Übernimmt den in `fd` übergebenen Dateideskriptor und gibt eine daraus erstellte `client_t`-Struktur zurück. Im Fehlerfall werden belegte Ressourcen freigegeben, eine Fehlermeldung ausgegeben und der Rückgabewert `NULL` an den Aufrufer gemeldet.

`void client_destroy(client_t *c)`
: Gibt die von `c` belegten Ressourcen frei.

`void fifo_put(struct fifo *f, client_t *c)`
: Hängt `c` an das Ende der *FIFO* `f` an.

`client_t *fifo_get(struct fifo *f)`
: Wartet ggf. passiv bis eine `client_t`-Struktur am Kopf von `f` vorliegt und entnimmt diese.

`void *worker(void *tutors)`
: Entnimmt fortlaufend je eine Verbindung aus `fifo_accept` zur Bearbeitung. Diese hängt von der mit `role()` ermittelten Rolle der Verbindung ab. Kann sie nicht ermittelt werden, so wird die Verbindung geschlossen und ignoriert. Bei `ROLE_STUDENT` wird die Verbindung zur späteren Verarbeitung in `fifo_students` eingereiht. Für Tutoren wird auf eine Verbindung aus `fifo_students` gewartet. Liegen Verbindungen beider Rollen vor, so wird abwechselnd – beginnend mit `ROLE_TUTOR` – je eine Zeile übermittelt (`exchange_line`) bis ein Fehler oder Verbindungsende auftritt.

`int role(FILE *rx, const char *tutors[])`
: Liest eine Zeile von `rx` als Nutzernamen ein und sucht diesen im `NULL`-terminierten Array `tutors`. Bei einer Übereinstimmung ist der Rückgabewert `ROLE_TUTOR`, sonst `ROLE_STUDENT`. Fehler oder überlange Zeilen führen zu einer passenden Meldung auf `stderr` und der Rückgabewert ist `ROLE_UNKNOWN`.

`int exchange_line(FILE *rx, FILE *tx)`
: Leitet die auf `rx` eingehenden Zeichen einer Zeile an `tx` weiter. Wurde eine gesamte Zeile erfolgreich übermittelt wird `0` zurückgegeben. Fehler werden auf `stderr` ausgegeben und - wie auch geschlossene Verbindungen - mit dem Rückgabewert `-1` gemeldet.

Hinweise:

- Der `argv`-Vektor ist durch `NULL` terminiert.
- Die **`struct`** `fifo` Instanzen sind bereits initialisiert.
- Nutzernamen sind maximal 20 Zeichen lang und enthalten kein `'\n'` Zeichen.
- Kein Tutor wird sich jemals mehrfach/parallel verbinden.

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

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

#define PORT         7370
#define MAX_USER_NAME 20

#define ROLE_TUTOR    0
#define ROLE_STUDENT  1
#define ROLE_UNKNOWN -1

// vorgegeben Strukturen:
typedef struct client {
  FILE *rx, *tx;
  struct client *next;
} client_t;

struct fifo {
  client_t       *head;
  pthread_mutex_t mutex;
  pthread_cond_t  condition;
};

// vorgegeben Funktionen:
static void die(const char *msg) {
  perror(msg);
  exit(EXIT_FAILURE);
}
static void usage(const char *name) {
  fprintf(stderr, "USAGE: %s <tutor-names...>\n", name);
}

// Globale Variablen:
static struct fifo fifo_accept = {
  .head = NULL,
  .mutex = PTHREAD_MUTEX_INITIALIZER,
  .condition = PTHREAD_COND_INITIALIZER,
};
static struct fifo fifo_students = {
  .head = NULL,
  .mutex = PTHREAD_MUTEX_INITIALIZER,
  .condition = PTHREAD_COND_INITIALIZER,
};
```

```
int main(int argc, char *argv[]) {
  // Argumente prüfen und Prozesszustand initialisieren

  // Arbeiterfäden starten

  // Socket öffnen
```



```
  // Verbindungen annehmen

}
```

M:

```
static client_t * client_create(int fd) {
```

```
}

static void client_destroy(client_t *c) {
}
```

C:


```
static void fifo_put(struct fifo *f, client_t *c) {
}

static client_t *fifo_get(struct fifo *f) {
}
```

F:

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

    // Rolle der Verbindung prüfen

    // Kommunikation ausführen

}
```

W:

```
static int role(FILE *rx, const char *tutors[]) {
  // Nutzername einlesen

  // Nutzername prüfen

}
```

R:

```
static int exchange_line(FILE *rx, FILE *tx) {

}
```

Z:

Aufgabe 3: Synchronisation (11 Punkte)

1) Was versteht man unter einer Verklemmung und was sind die (hinreichenden und notwendigen) Bedingungen, damit eine Verklemmung auftreten kann? (6 Punkte)

2) Erläutern Sie das Konzept Semaphor. Welche Operationen sind auf Semaphoren definiert und was tun diese Operationen? (5 Punkte)

Aufgabe 4: Adressräume & Freispeicherverwaltung (6 Punkte)

1) Man unterscheidet bei Adressraumkonzepten und bei Zuteilungsverfahren zwischen externer und interner Fragmentierung. Erläutern Sie den Unterschied. Was kann man jeweils gegen die beiden Arten der Fragmentierung tun? (4 Punkte)

2) Im Hinblick auf Adressraumkonzepte gibt es bei interner Fragmentierung einen Nebeneffekt in Bezug auf Programmfehler (vor allem im Zusammenhang mit Zeigern). Beschreiben Sie diesen Effekt. (2 Punkte)

Aufgabe 5: Dateisystem (13 Punkte)

1) Beschreiben Sie kurz (in Stichworten) die grundsätzliche Funktionsweise eines *Journaling-File-Systems.* (3 Punkte)

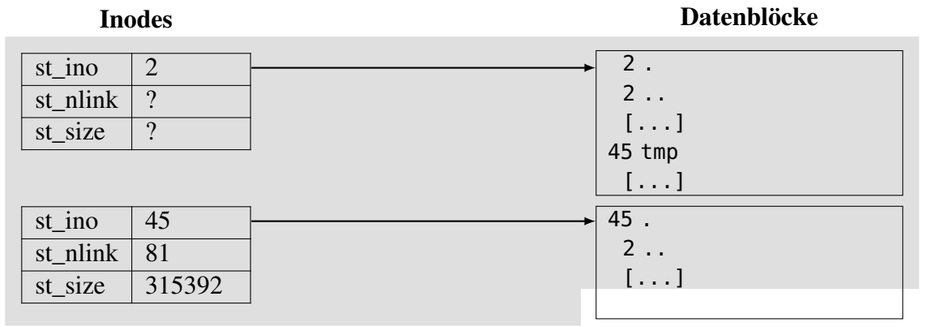
2) Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/sp2` auf einem Linux-System; eine rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/sp2` mit Angabe der Inode-Nummer (erste Spalte), des Referenzzählers (dritte Spalte) und der Dateigröße (fünfte Spalte). (10 Punkte)

```
gerhorst@fauiXX:~$ ls -aoRi /tmp/sp2
/tmp/sp2:
total 60
48 drwx------  3 gerhorst   4096 Feb 14 15:12 .
45 drwxrwxrwt 81 root     315392 Feb 14 15:14 ..
61 drwxr-xr-x  3 gerhorst   4096 Feb 14 15:13 stud

/tmp/sp2/stud:
total 16
61 drwxr-xr-x 3 gerhorst 4096 Feb 14 15:13 .
48 drwx------ 3 gerhorst 4096 Feb 14 15:12 ..
59 lrwxrwxrwx 1 gerhorst   16 Feb 14 15:12 gerhorst -> /tmp/sp2/stud/53
53 drwxr-xr-x 2 gerhorst 4096 Feb 14 15:14 luis
64 -rw-r--r-- 2 gerhorst    1 Feb 14 15:06 whatami

/tmp/sp2/stud/luis:
total 12
53 drwxr-xr-x 2 gerhorst 4096 Feb 14 15:14 .
61 drwxr-xr-x 3 gerhorst 4096 Feb 14 15:13 ..
64 -rw-r--r-- 2 gerhorst    1 Feb 14 15:06 file
```

Ergänzen Sie im weißen Bereich die auf der folgenden Seite im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	