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

- ☐ Statisch gebundene Programme können zum Ladezeitpunkt an beliebige virtuelle Speicheradressen platziert werden.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst.
- ☐ Beim dynamischen Binden erfolgt die Adressauflösung beim laden des Programms, oder zur Laufzeit.
- ☐ Änderungen am Code einer dynamischen Bibliothek (z. B. Bugfixes) erfordern immer das erneute Binden aller Programme, die diese Bibliothek benutzen.

b) Bei Demand-Paging kann der Effekt des Seitenflatterns (Thrashing) auftreten. Welche Aussage ist richtig? 2 Punkte

- ☐ Wird eine eben ausgelagerte Seite gleich wieder angesprochen, so muss diese wieder eingelagert werden. Tritt dieser Effekt häufig auf, so spricht man von Seitenflattern.
- ☐ Seitenflattern tritt auf, wenn Seiten zur Defragementierung im Speicher verschoben werden.
- ☐ Bei der Ersetzungsstrategie Second Chance (SC) wird bei einem Zugriff auf eine Seite ein Referenzbit gesetzt. Wird die Seite längere Zeit nicht angesprochen, so wird dieses Bit gelöscht. Da dieses Bit ständig den Wert ändert, spricht man von Seitenflattern.
- ☐ Seitenflattern kann nur auftreten, wenn der dynamisch genutzte Speicher eines Prozesses größer ist, als der physikalisch vorhandene Speicher des Systems.

c) Beim Einsatz von RAID-Systemen kann durch zusätzliche Festplatten Fehlertoleranz erzielt werden. Welche Aussage dazu ist richtig? 2 Punkte

- ☐ Bei allen RAID-Systemen ist ein höherer Lese-Durchsatz als bei einer einzelnen Platte möglich, da mehrere Platten gleichzeitig beauftragt werden können.
- ☐ Bei RAID 4 Systemen wird Paritätsinformation gleichmäßig über alle beteiligten Platten verteilt.
- ☐ RAID 0 erzielt Fehlertoleranz durch das Verteilen der Daten auf mehrere Platten.
- ☐ Bei RAID 4 und 5 darf eine bestimmte Menge von Festplatten nicht überschritten werden, da es sonst nicht mehr möglich ist, die Paritätsinformation zu bilden.

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

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

e) Man unterscheidet Programmunterbrechungen in Traps und Interrupts. Welche Aussage ist richtig? 2 Punkte

- ☐ Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie Interrupt einzuordnen.
- ☐ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleichen Eingabedaten treten Interrupts immer an den gleichen Stellen auf.
- ☐ Ein gerade laufendes Maschinenprogramm kann bei Bedarf die Behandlung aller Programmunterbrechungen unterdrücken.
- ☐ Normale Rechenoperationen können zu einem Trap führen.

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

- ☐ 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 der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (Segmentation Fault) abgebrochen.
- ☐ Der TLB verkürzt die Zugriffszeit auf den physikalischen Speicher da ein Teil des möglichen Speichers in einem schnellen Pufferspeicher vorgehalten wird.
- ☐ Der TLB puffert Daten bei der Ein-/Ausgabebehandlung und beschleunigt diese damit.

g) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux Dateideskriptors zu? 2 Punkte

- ☐ Ein Dateideskriptor kann über gemeinsamen Speicher an einen anderen Prozess übergeben werden und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden.
- ☐ Dateideskriptoren sind Zeiger auf Betriebssystem-interne Strukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ☐ Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei benutzen kann.
- ☐ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Dateideskriptor zum Zugriff zurück.

h) Welche Aussage zum Thema Speicherzuteilung ist richtig?

2 Punkte

- ☐ Bei allen listenbasierten Zuteilungsverfahren (First-, Next-, Best-, Worst-Fit) kann externer Verschnitt auftreten.
- ☐ Die Worst-Fit-Strategie ist lediglich theoretisch interessant, da es in der Praxis nie sinnvoll ist, den am schlechtesten passenden Speicherplatz zuzuweisen.
- ☐ Beim Next-Fit-Verfahren muss entstehender Verschnitt immer am Ende der Freispeicherliste einsortiert werden.
- ☐ Beim First-Fit-Verfahren ist die Liste der freien Speicherbereiche aufsteigend nach der Größe der jeweiligen Bereiche sortiert.

i) Gegeben seien die folgenden Präprozessor-Makros:

2 Punkte

```
#define ADD(a, b) a + b
#define DIV(a, b) a / b
```

Was ist das Ergebnis des folgenden Ausdrucks?

```
3 * DIV(ADD(4, 8), 2)
```

- ☐ 18
- ☐ 10
- ☐ 24
- ☐ 16

j) Welche Aussage über Einplanungsverfahren ist richtig?

2 Punkte

- ☐ Im Round-Robin-Verfahren nutzen E/A-intensive Prozesse die ihnen zugeteilte Zeitscheibe immer voll aus.
- ☐ Der Konvoieffekt kann bei kooperativen Einplanungsverfahren wie First-Come-First-Served nicht auftreten.
- ☐ Bei kooperativen Verfahren können Prozesse die CPU nicht monopolisieren.
- ☐ In einem asymmetrischen Multiprozessorsystem ist der Einsatz von asymmetrischen Verfahren zur Planung obligatorisch.

k) Welche Aussage zu virtuellem Speicher ist richtig?

2 Punkte

- ☐ Virtueller Speicher sind die nicht vorhandenen Bereiche des physikalischen Adressraums.
- ☐ Virtueller Speicher kann dynamisch zur Laufzeit von einem Programm mit der Funktion `malloc(3p)` erzeugt werden.
- ☐ Unter einem Virtuellen Speicher versteht man einen physikalischen Adressraum, dessen Adressen durch eine MMU vor dem Zugriff auf logische Adressen umgesetzt werden.
- ☐ Virtueller Speicher kann größer sein als der physikalisch vorhandene Arbeitsspeicher. Gerade nicht benötigte Speicherbereiche können auf Hintergrundspeicher ausgelagert werden.

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) Konzeptionell ist der Speicher eines UNIX-Prozesses in Text-, Daten- und Stack-Segment untergliedert. Welche der folgenden Aussagen treffen zu?

4 Punkte

- ○ Lokale „automatic“ Variablen einer Funktion werden im Stack-Segment abgelegt.
- ○ Das Text-Segment enthält sowohl den Programmcode als auch konstante Zeichenketten.
- ○ Dynamisch allozierte Zeichenketten werden in das Text-Segment gelegt.
- ○ Variablen der Speicherklasse „automatic“ werden durch den Übersetzer mit dem Wert 0 initialisiert.
- ○ Zeigervariablen können auf Daten aus allen Segmenten verweisen.
- ○ Vor Ausführung einer Funktion wird das Daten-Segment vergrößert, um den Speicher für lokale Variablen zu reservieren.
- ○ Variablen der Speicherklasse „static“ liegen im Daten-Segment.
- ○ Lokale Variablen der Speicherklasse „static“ werden beim Betreten der zugehörigen Funktion neu initialisiert.

b) Man unterscheidet die Begriffe Programm und Prozess. Welche der folgenden Aussagen zu diesem Themengebiet sind richtig?

4 Punkte

- ○ Der Prozess ist der statische Teil (Rechte, Speicher, etc.), das Programm der aktive Teil (Programmzähler, Register, Stack).
- ○ Der Systemaufruf `exec(3)` ersetzt das bestehende Programm im aktuell laufenden Prozess.
- ○ Der Übersetzer (Compiler) erzeugt aus mehreren Programmen (Modulen) einen Prozess.
- ○ Ein Programm kann immer nur einen Prozess ausführen.
- ○ Der Systemaufruf `fork(3)` erstellt einen neuen Prozess, der das gleiche Programm ausführt.
- ○ Der Binder erzeugt aus einer oder mehreren Objekt-Dateien ein Programm.
- ○ Ein Prozess kann mit Hilfe von Threads mehrere Programme gleichzeitig ausführen.
- ○ Ein Programm kann durch mehrere Prozesse gleichzeitig ausgeführt werden.

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

Aufgabe 2: `meet` (60 Punkte)

Schreiben Sie das Programm `meet` (**me**morize **e**encoun**t**er), das als Backend einer Kontaktverfolgungs-Anwendung genutzt werden kann. Dazu speichert `meet` die Kombination aus pseudonymisierten Nutzernamen und Zeitstempel. Es können neue Einträge hinzugefügt und Anfragen über den Kontakt eines Nutzers ab einem bestimmten Zeitpunkt gestartet werden.

Der Server lauscht dafür auf IPv6-Port 49177 auf eingehende Verbindungen und verteilt diese an einzelne Threads zur Bearbeitung. Eine Anfrage besteht aus einer einzelnen Zeile und ist laut Protokoll wie folgt definiert: `<user>␣<task>␣<arg>\n`

Dabei ist `<user>` der aus **16** Hexdezimalziffern bestehende pseudonymisierte Nutzername und `<task>` gibt die gewünschte Operation an (`'a'` für „add“, `'l'` für „list“). Bei „add“ wird das Pseudonym des Kontaktes in `<arg>` erwartet. Für „list“ enthält `<arg>` den Zeitstempel, ab dem die Kontakte gelistet werden sollen. Zeitstempel werden in **8** Hexadezimalstellen codiert. Im Folgenden ist je ein Beispiel für „list“ und „add“ abgedruckt:

Anfrage:
`e1549aa85a2a1e91␣l␣603623a0`
Antwort:
`24cf13a1dfe835a8`
`98d90ccc8610db89`

Anfrage:
`e1549aa85a2a1e91␣a␣24cf13a1dfe835a8`
Antwort:

Implementieren Sie die folgenden Funktionen:

`void main(void)` Initialisiert das Programm und startet **16** Arbeiterfäden. Anschließend werden eingehende Verbindungen zur Verarbeitung an `handle_connection` weitergegeben.

`void handle_connection(int s)` Liest die Anfragezeile und ruft `handle_request` mit einem schreibbaren Dateistrom auf die Verbindung (`tx`) sowie relevanten Bestandteilen der Anfragezeile auf. Im Fehlerfall muss `tx` aufgeräumt werden.

`int handle_request(FILE *tx, char *user, char task, char *arg)` Validiert die Anfrageparameter und fügt diese der Schlange von Aufträgen des für `user` zuständigen Arbeiterfadens (`threadid_for_pseudonym`) hinzu. Treten dabei Fehler auf, gibt die Funktion -1 zurück.

`void job_enqueue(struct queue *q, struct job *j)` Fügt `j` in $\mathcal{O}(1)$ an `q`s Ende ein.

`struct job *job_dequeue(struct queue *q)` Entnimmt das erste Element in $\mathcal{O}(1)$ aus `q`. Blockiert solange kein Element vorhanden ist.

`void *worker(struct queue *q)` Die Thread-Routine entnimmt Aufträge aus der Schlange `q` und führt die entsprechende `task`-Funktion aus. Die Verbindung zum Klienten sowie die belegten Ressourcen sollen freigegeben werden, bevor der nächste Auftrag entnommen wird.

`void list(struct job *job)` Sendet zeilenweise alle Pseudonyme, deren letzter Kontakt mit `job->user` nach dem Zeitpunkt `job->timestamp` war an den Klienten (`job->tx`).

Die Funktion für die „add“-Anfrage ist bereits als **`void`** `add(`**`struct`** `job *)` gegeben und muss nicht implementiert werden. Falls notwendig, erstellt diese im Ausführungsverzeichnis ein nach dem Nutzer benanntes Unterverzeichnis und legt darin für den angegeben Kontakt (`job->other`) eine neue Datei an. Der Zeitpunkt des Kontakts wird automatisch als Modifikationszeit der Datei vermerkt.

Hinweise:

- Fehler die während der Initialisierung auftreten oder nicht sinnvoll behandelt werden können dürfen zur Terminierung des Programms führen.
- Fehler die kein Protokollverstoß darstellen müssen mit einer Fehlermeldung beschrieben werden
- Sie können davon ausgehen, dass die Bestandteile einer Anfrage durch ein einzelnes Leerzeichen getrennt sind
- Das Feld `.st_mtime.tv_sec` von **`struct`** `stat` gibt den Zeitpunkt der letzten Modifikation der Datei an

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

```c
#include <dirent.h>
#include <errno.h>
#include <netinet/in.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>

#define HASHLEN 16
#define UTSLEN 8
#define NTHREADS 16
#define PORT 49177

struct job {
  FILE *tx;
  struct job *next;
  void (*task)(struct job *);
  int timestamp;
  char user[HASHLEN +1];
  char other[HASHLEN +1];
};

struct queue {
  struct job *head;
  struct job **tail;
  pthread_mutex_t guard;
  pthread_cond_t cond;
} queue[NTHREADS];

static void handle_connection(int);
static int handle_request(FILE *, char *, char, char *);
static void * worker(struct queue *);
static void list(struct job *);
static void job_enqueue(struct queue *, struct job *);
static struct job *job_dequeue(struct queue *);

/* vorgegeben */
static void die(const char *const msg);
static int is_valid_pseudonym(const char * const); // return 1, on valid hashes
static int parse_uhex(char *); // parse positive hex-number, return -1 on error
static void add(struct job *job); // creates the appropriate contact file
static int threadid_for_pseudonym(char * ); // determine index from pseudonym
```

```
int main(void) {
  // Zustand initialisieren

  // Threads initialisieren

  // Verbindungen akzeptieren
```



```
}
```

M:

```
void handle_connection(int s) {

  // Anfrage einlesen

  // Anfrage parsen
```



```
}

int handle_request(FILE *tx, char *user, char task, char *arg) {
  // Validierung der Eingabe
```

```
  // Erstellung struct job
}
```

D:


```
void * worker(struct queue *q) {
}
```

```
void list(struct job *job) {

  // Verzeichniseinträge lesen

}
```

C:


```
void job_enqueue(struct queue *q, struct job *j) {

}

struct job * job_dequeue(struct queue *q) {

}
```

Q:

Aufgabe 3: Synchronisation (14 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)

3) Skizzieren Sie in Programmiersprachen-ähnlicher Form, wie mit Hilfe eines zählenden Semaphors das folgende Szenario korrekt synchronisiert werden kann: Zu jedem Zeitpunkt müssen so viele Threads wie möglich, maximal jedoch 4, die Funktion `threadFunc` ausführen. Ihnen stehen dabei folgende Semaphor-Funktionen zur Verfügung: (3 Punkte)

- `SEM * semCreate(int);`
- `void P(SEM *);`
- `void V(SEM *);`

Beachten Sie, dass nicht unbedingt alle freien Zeilen für eine korrekte Lösung nötig sind. Kennzeichnen Sie durch `/`, wenn Ihre Lösung in einer freie Zeile keine Operation benötigt.

Hauptthread:

```
static SEM *s;
int main(void){

-------------------------------------

  while(1) {

-------------------------------------

  startWorkerThread(threadFunc);

-------------------------------------

  }

-------------------------------------

}
```

Arbeiterthread:

```
void threadFunc(void) {

-------------------------------------

  doWork();

-------------------------------------

}
```

Aufgabe 4: Prozesse (16 Punkte)

1) Beschreiben Sie die Prozesszustände **bei kurz- und mittelfristiger** Einplanung sowie die Ereignisse, die jeweils zu den Zustandsübergängen führen (Skizze mit kurzer Erläuterung der Zustände und Übergänge). (10 Punkte)

2) Es gibt Prozesse, Kernel Threads und User Threads. Beschreiben Sie in Stichworten wie sich diese voneinander unterscheiden. Nennen sie dazu für jede Art je zwei disjunkte Eigenschaften oder Vor-/Nachteile. (6 Punkte)