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 zur Seitenumlagerung in virtuellen Adressräumen ist richtig? 2 Punkte

- ☐ Beim Auslagern einer Speicherseite muss der zugehörige Seitendeskriptor angepasst werden. Beim Einlagern ist dies nicht nötig.
- ☐ Unter Seitenflattern versteht man das ständige Ein- und Auslagern von Speicherseiten, wenn der physisch vorhandene Hauptspeicher nicht ausreicht.
- ☐ Bei der Seitenersetzungsstrategie LRU wird diejenige Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ☐ Die Seitenersetzungsstrategie Second Chance (Clock) ist nur in der Theorie interessant, weil ihre Implementierung komplexe Datenstrukturen erfordert.

b) Beim Blockieren in einem Monitor muss der Monitor freigegeben werden. Warum? 2 Punkte

- ☐ Weil kritische Abschnitte immer nur kurz belegt sein dürfen.
- ☐ Weil ein anderer Thread die Blockierungsbedingung nur aufheben kann, wenn er den Monitor vorher betreten kann.
- ☐ Weil sonst die Monitordaten inkonsistent sind.
- ☐ Weil der Thread sonst aktiv warten würde.

c) Welche Aussage ist in einem Monoprozessor-Betriebssystem richtig? 2 Punkte

- ☐ Es befindet sich zu einem Zeitpunkt maximal ein Prozess im Zustand laufend.
- ☐ Ist zu einem Zeitpunkt kein Prozess im Zustand bereit, so ist auch kein Prozess im Zustand laufend.
- ☐ In den Zustand blockiert gelangen Prozesse nur aus dem Zustand bereit.
- ☐ Ein Prozess im Zustand blockiert muss warten, bis der laufende Prozess den Prozessor abgibt und kann dann in den Zustand laufend überführt werden.

d) Wozu benötigt man Bedingungsvariablen (*condition variables*)? 2 Punkte

- ☐ Bei if-Abfragen in C-Programmen.
- ☐ Zur Erkennung von Verklemmungen.
- ☐ Zum aktiven Warten in kritischen Abschnitten.
- ☐ Um in einem kritischen Abschnitt auf ein Ereignis zu warten und den kritischen Abschnitt während der Wartezeit freizugeben.

e) Was versteht man unter einer Unterbrechung bei der Ausführung von Instruktionen durch einen Prozessor?

2 Punkte

- ☐ Mit einer Signalleitung wird dem Prozessor eine Unterbrechung angezeigt. Der Prozessor sichert den aktuellen Zustand bestimmter Register, insbesondere des Programmzählers, und springt eine vordefinierte Behandlungsfunktion an.
- ☐ Der Prozessor wird veranlasst eine Unterbrechungsbehandlung durchzuführen. Der gerade laufende Prozess kann die Unterbrechungsbehandlung ignorieren.
- ☐ Durch eine Signalleitung wird der Prozessor veranlasst, die gerade bearbeitete Maschineninstruktion zu unterbrechen und in den Benutzermodus umzuschalten.
- ☐ Eine Signalleitung teilt dem Prozessor mit, dass er den aktuellen Prozess anhalten und auf das Ende der Unterbrechung warten soll.

f) Welche Aussage zum Thema Systemaufrufe ist richtig?

2 Punkte

- ☐ Mit Hilfe von Systemaufrufen kann ein Benutzerprogramm privilegierte Operationen durch das Betriebssystem ausführen lassen, die es im normalen Ablauf nicht selbst ausführen dürfte.
- ☐ Durch die Bereitstellung von Systemaufrufen, kann ein Benutzerprogramm das Betriebssystem um eigene Funktionen erweitern.
- ☐ Die Bearbeitung eines Systemaufrufs findet immer im selben Adressraum statt, aus dem heraus der Systemaufruf abgesetzt wurde.
- ☐ Benutzerprogramme dürfen keine Systemaufrufe absetzen, diese sind dem Betriebssystem vorbehalten.

g) Wie funktioniert Adressraumschutz durch Eingrenzung?

2 Punkte

- ☐ Jedes Programm bekommt zur Ladezeit mehrere Wertepaare aus Basis- und Längenregistern zugeordnet, die die Größe aller Segmente des darin laufenden Prozesses festlegen.
- ☐ Begrenzungsregister legen einen Adressbereich im physikalischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.
- ☐ Der Lader positioniert Programme immer so im Arbeistsspeicher, dass unerlaubte Adressen mit nicht-existierenden physikalischen Speicherbereichen zusammenfallen.
- ☐ Die MMU kennt die Länge eines Segments und verhindert Speicherzugriffe darüber hinaus.

h) Welche Aussage zum Thema Programme und Prozesse ist richtig?

2 Punkte

- ☐ In einem Prozess kann immer nur ein Programm ausgeführt werden.
- ☐ Der Compiler erzeugt aus mehreren Programmteilen (Module) einen Prozess.
- ☐ Ein Programm kann durch mehrere Prozesse gleichzeitig ausgeführt werden.
- ☐ Ein Prozess kann gleichzeitig mehrere verschiedene Programme ausführen.

i) In einem UNIX-Dateisystem gibt es symbolische Namen/Verweise (*symbolic links*) und feste Links (*hard links*) auf Dateien. Welche Aussage ist richtig?

2 Punkte

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

j) Welche der folgenden Aussagen trifft auf das Programmfragment zu?

2 Punkte

```
int f1 (const int *y) {
    static int b;
    char *d = malloc(2407);
    int (*e)(const int *) = f1;
    b += *y;
    y++;
    return b;
}
```

- ☐ Die Speicherstelle, auf die `d` zeigt, verliert beim Rücksprung aus der Funktion `f1()` ihre Gültigkeit.
- ☐ Die Variable `b` liegt im Stacksegment.
- ☐ Die Variable `e` liegt im Stacksegment und zeigt auf eine Stelle im Textsegment.
- ☐ Die Anweisung `y++` führt zu einem Laufzeitfehler, da `y` konstant ist.

k) User-Level- und Kernel-Level-Threads unterscheiden sich in verschiedenen Eigenschaften. Welche Aussage ist richtig?

2 Punkte

- ☐ Bei User-Level-Threads können anwendungsabhängig Schedulingstrategien eingesetzt werden.
- ☐ Blockiert ein Systemaufruf, so wird automatisch der nächste User-Level-Thread ausgeführt.
- ☐ Kernel-Level-Threads werden effizienter umgeschaltet als User-Level-Threads.
- ☐ Nur durch User-Level-Threads können mehrere Prozessoren genutzt 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) Welche der folgenden Aussagen zum Thema Einplanung sind richtig?

4 Punkte

- ○ Für die mittelfristige Einplanung muss das Betriebssystem die Umlagerung (engl. swapping) von kompletten Programmen bzw. logischen Adressräumen unterstützen.
- ○ Prozesse im Zustand gestoppt sind der langfristigen Einplanung zuzuordnen.
- ○ Verdrängende Prozesseinplanung bedeutet, dass das Eintreten des erwarteten Ereignisses unmittelbar die Einlastung des wartenden Prozesses bewirkt.
- ○ 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 (*blocked suspend*) überführt werden.
- ○ Ein Prozess im Zustand erzeugt kann sich selbst durch die Ausführung des Systemaufrufes exec() in den Zustand bereit überführen.
- ○ Prozesse im Zustand blockiert oder bereit können unmittelbar in den Zustand gestoppt überführt werden.
- ○ 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.

b) Man unterscheidet zwischen Traps und Interrupts. Welche der folgenden Aussagen ist richtig?

4 Punkte

- ○ Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- ○ Der Zugriff auf eine physikalische Adresse kann keinen Trap auslösen.
- ○ Der Zugriff auf eine logische Adresse kann zu einem Trap führen.
- ○ Ganzzahl-Rechenoperationen können nicht zu einem Trap führen.
- ○ Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie Interrupt einzuordnen.
- ○ Rechenoperationen können zu einem Interrupt führen.
- ○ Ein Programm darf im Rahmen einer Trapbehandlung abgebrochen werden.
- ○ Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.

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

Aufgabe 2: ypsilon (60 Punkte)

Schreiben Sie ein Programm *ypsilon*, das auf dem TCP/IPv6-Port 2024 (`LISTEN_PORT`) einen Dienst anbietet, über welchen Benutzer Kurznachrichten verbreiten und abrufen können. Parallele Verbindungen sollen dabei von Arbeiter-Threads abgearbeitet werden, welche vom Haupt-Thread nach Verbindungsannahme gestartet werden. Die maximale Anzahl gleichzeitig unterstützter Verbindungen wird dem Server als Kommandozeilenargument übergeben. Jede Nachricht soll vom Dienst außerdem mit einer global eindeutigen, von 0 an aufsteigenden ID vom Typ `unsigned int` versehen werden (**Hinweis:** die maximale Länge der String-Repräsentation ist als Makro `UINT_LEN` gegeben).

Alle Nachrichten werden unterhalb des Verzeichnisses `/var/lib/ypsilon` (`STORAGE_DIR`) in Unterverzeichnissen, die den Namen des jeweiligen Benutzers tragen, als eigene Dateien unter dem Namen `t<ID der Nachricht>` gespeichert (z.B.: `/var/lib/ypsilon/elontusk/t47`).

Ein Client sendet nach erfolgreichem Verbindungsaufbau zunächst eine Zeile, die die auszuführende Aktion spezifiziert; der Dienst unterstützt dabei folgende zwei Befehle:

`PUT` ***username*** initiiert den Sendevorgang einer neuen Kurznachricht an den Server. Der Benutzername hat dabei eine maximale Länge von 15 Zeichen (`MAX_NAME`). Die Nachricht selbst wird anschließend in einer eigenen Zeile an den Server geschickt und darf maximal 280 Zeichen (`MAX_MSG_LEN`) lang sein. Überlange Nachrichten werden abgeschnitten.

`GET` ***username*** liefert alle unter einem Nutzernamen gespeicherten Kurznachrichten zurück.

Implementieren Sie folgende Funktionen:

`int main(int argc, char *argv[])` Das Hauptprogramm initialisiert zunächst alle benötigten Datenstrukturen und nimmt auf einem Socket Verbindungen an, sofern die maximale Anzahl laufender Threads noch nicht erreicht wurde. Eine erfolgreich angenommene Verbindung wird an einen neuen Thread, welcher in der Funktion `handleConnection` startet, weitergegeben.

`void *handleConnection(void *)` Die von den Arbeiter-Threads ausgeführte Funktion. Zur passend übergebenen Verbindung werden zunächst durch einen Aufruf an die von uns vorgegebene Funktion `sock2fds` zwei `FILE *` zur Kommunikation mit dem Client geöffnet. Die Anfragezeile wird dann in der Funktion `parseCommand` bearbeitet. Tritt dabei ein Fehler auf, soll eine kurze Fehlermeldung an den Client gesendet werden (z.B. ”Error handling command”).

`int parseCommand(FILE *rx, FILE *tx)` Liest eine Anfragezeile vom Client, wertet sie aus und ruft dann die passende Funktion (`storeMessage()` bzw. `getMessages()`) auf. Im Falle des PUT-Kommandos muss außerdem die Nachricht vom Client eingelesen werden. Tritt ein Fehler bei der Auswertung der Zeile oder in einer der aufgerufenen Funktionen auf, gibt parseCommand -1 zurück, im Erfolgsfall den Wert 0.

`int storeMessage(const char *username, const char *message)` Nach der Zuweisung der aktuellen ID wird zunächst eine temporäre Datei im Arbeitsverzeichnis des Servers angelegt, welche nach der ID der Nachricht benannt ist. Nachdem die übergebene Nachricht in der Datei gespeichert wurde, soll sie mit Hilfe der Funktion `rename` (siehe Manual-Seiten) an die passende Stelle (in das Unterverzeichnis <username> mit Dateiname t<ID>) verschoben werden. Im Erfolgsfall liefert die Funktion den Wert 0 zurück, im Fehlerfall wird -1 zurückgegeben.

`int getMessage(const char *username, FILE *tx)` Liefert alle Nachrichten des angefragten Benutzers an den Client aus. Dazu wird das passende Unterverzeichnis nach regulären Dateien durchsucht, deren Name mit ’t’ beginnt. Der Inhalt passender Dateien wird dann über den übergebenen Kommunikationskanal ausgeliefert. Im Erfolgsfall liefert die Funktion den Wert 0 zurück, im Fehlerfall wird -1 zurückgegeben.

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

```c
#include <dirent.h>
#include <errno.h>
#include <limits.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>
#include "sem.h"

#define LISTEN_PORT  2024
#define MAX_MSG_LEN  280
#define MAX_NAME_LEN 15
#define STORAGE_DIR  "/var/lib/ypsilon/"
#define UINT_LEN     (sizeof(unsigned int) * 8)

static void die(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}

// öffnet zu dem übergebenen Socket sock zwei FILE * rx (lesbar) und tx
// (schreibbar) und gibt diese über die übergebenen Zeiger zurück. Im
// Fehlerfall wird der Wert -1 zurückgegeben, ansonsten 0.
static int sock2fds(int sock, FILE **rx, FILE **tx) { /* ... */ }

// Gibt einen positiven int zurück, oder -1 im Fehlerfall:
static int parsePositiveInt(char *str) { /* ... */ }
// Allokiert SEM und initialisiert alle Felder. Gibt im Fehlerfall
// NULL zurück und setzt errno:
SEM *semCreate(int initVal);

// Dekrementiert Zähler von SEM um eins und blockiert dabei
// solange der Zähler <= 0 ist.
void P(SEM *sem);

// Inkrementiert Zähler von SEM um eins und weckt dabei einen
// wartenden Thread auf falls vorhanden.
void V(SEM *sem);
```

```
// Funktions- und Strukturdeklarationen, globale Variablen, etc.

int main(int argc, char *argv[]){
    // Argumente einlesen und pruefen, weitere Vorbereitungen
    if (argc != 2){
        fprintf(stderr, "Usage: %s <thread_count>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

```
    // Socket erstellen und vorbereiten
```

```
    // Verbindungen annehmen, Threads starten
```

```
}
```

M:

```
static void* handleConnection(void *arg) {
    // Kommunikation vorbereiten
    return NULL;
}
```

H:

```
    // Anfragezeile auslesen, Behandlungsfunktion aufrufen
    return -1;
}
```

P:

```
static int storeMessage(char *name, char *message) {
    // Nachricht in temporaere Datei schreiben

    // Datei umbenennen

}
```

S:

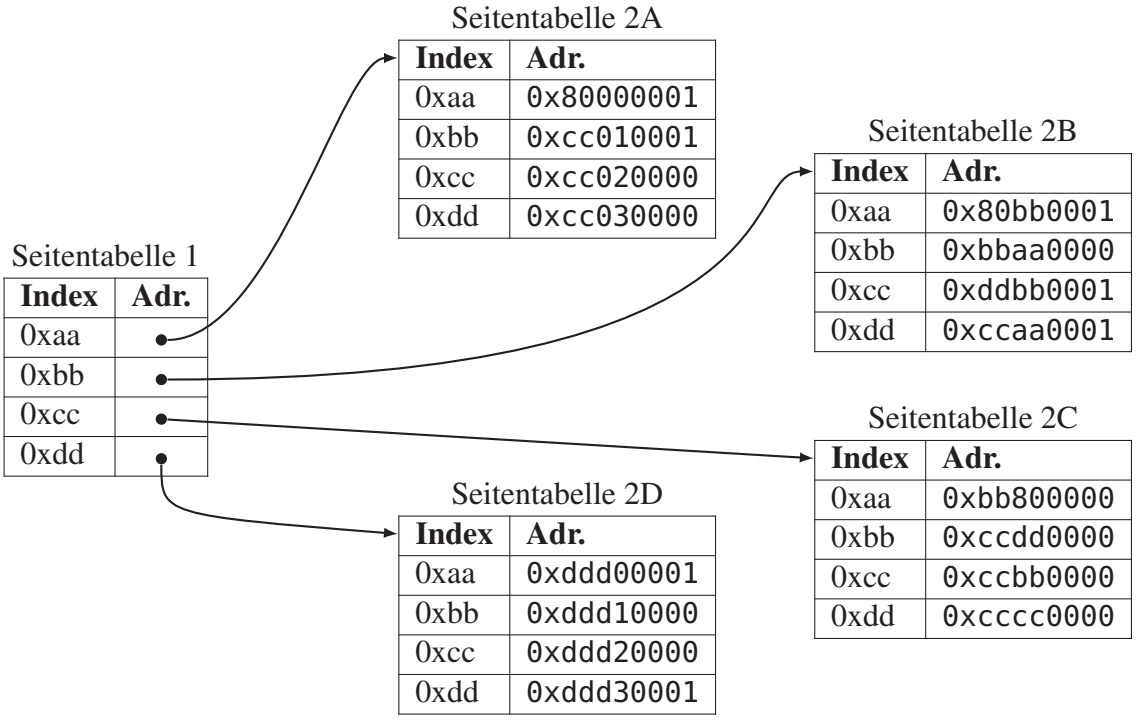
```
static int getMessages(char *name, FILE *tx) {
    // Verzeichnis oeffnen, nach passenden Dateien durchsuchen

    return 0;
}
```

G:

Aufgabe 3: Paging (7 Punkte)

Gegeben sei unten dargestellte Hierarchie zweistufiger Seitentabellen. Die Adresslänge des genutzten Systems sei $32\,\mathrm{Bit}$, die Größe einer Seite $64\,\mathrm{Kibibyte}$ (i.e., 2^{16} Byte). Für die Indizierung der zweistufigen Abbildung werden pro Stufe $8\,\mathrm{Bit}$ genutzt. Das niederwertigste Bit eines Seitentabelleneintrags fungiert als Anwesenheitsbit: Ist es auf 1 gesetzt, ist die Seite anwesend.



1) Bestimmen Sie die **reale Adresse** zur logischen Adresse `0xbbaa1080`. Geben Sie hierbei die zur Bestimmung notwendigen Zwischenschritte stichpunktartig an! (4 Punkte)

2) Was passiert in Hardware und im Betriebssystem bei einem Zugriff auf eine aktuell ausgelagerte Speicherseite? (3 Punkte)

Aufgabe 4: Koordinierung (17 Punkte)

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

2) Betriebsmittel lassen sich in zwei Kategorien einteilen. Nennen und beschreiben Sie diese und geben Sie je zwei Beispiele. (6 Punkte)

3) Zur Koordinierung von nebenläufigen Vorgängen, die auf gemeinsame Betriebsmittel zugreifen, unterscheidet man zwischen einseitiger und mehrseitiger Synchronisation. Wie unterscheidet sich einseitige von mehrseitiger Synchronisation? (2 Punkte)

4) Statt blockierend zu synchronisieren kann in manchen Situationen mit optimistischen, nichtblockierenden Synchronisationsverfahren gearbeiten werden. Beschreiben Sie, wie solch ein Verfahren grundsätzlich funktioniert. (3 Punkte)

Aufgabe 5: Ausnahmen (6 Punkte)

1) Nennen Sie eins der zwei möglichen Modelle der Ausnahmebehandlung. Nennen Sie dabei für das von Ihnen gewählte Modell den Auslösegrund und die vom Modell angedachte Behandlungsstrategie. (3 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)