**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 Seitennummer und welcher Versatz gehören bei einer Seitengröße von 1024 Bytes zu folgender logischer Adresse? 0xcafe

2 Punkte

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

b) Was versteht man unter RAID 0?

2 Punkte

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

c) In welcher der folgenden Situationen wird ein laufender Prozess in den Zustand blockiert überführt?

2 Punkte

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





d) Welche der folgenden Aussagen zum Thema Seitenfehler (page fault) ist richtig?

2 Punkte

- ☐ Ein Seitenfehler zieht eine Ausnahmebehandlung nach sich. Diese wird dadurch ausgelöst, dass die MMU das Signal SIGSEGV an den aktuell laufenden Prozess schickt.
- ☐ Wenn der gleiche Seitenrahmen in zwei verschiedenen Seitendeskriptoren eingetragen wird, löst dies einen Seitenfehler aus (Gefahr von Zugriffskonflikten).
- ☐ Seitenfehler können auch auftreten, obwohl die entsprechende Seite gerade im physikalischen Speicher vorhanden ist.
- ☐ Ein Seitenfehler wird ausgelöst, wenn der Offset in einer logischen Adresse größer als die Länge der Seite ist.

e) Welche Aussage zum Thema Adressraumverwaltung ist richtig?

2 Punkte

- ☐ Da das Laufzeitsystem auf die Betriebssystemschnittstelle zur Speicherverwaltung zurückgreift, ist die Granularität der von malloc() zurückgegebenen Speicherblöcke vom Betriebssystem vorgegeben.
- ☐ Ein Speicherbereich, der mit Hilfe der Funktion free() freigegeben wurde, verbleibt möglicherweise im logischen Adressraum des zugehörigen Prozesses.
- ☐ Mit Hilfe des Systemaufrufes malloc() kann ein Programm zusätzliche Speicherblöcke von sehr feinkörniger Struktur vom Betriebssystem anfordern.
- ☐ 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.

f) Welche der folgenden Informationen wird typischerweise in dem Seitendeskriptor einer Seite eines virtuellen Adressraums gehalten?

2 Punkte

- ☐ Die Position der Seite im virtuellen Adressraum
- ☐ Die Identifikation des Prozesses, dem die Seite zugeordnet ist
- ☐ Die Zuordnung zu einem Segment (Text, Daten, Stack, ...)
- ☐ Zugriffsrechte (z. B. lesen, schreiben, ausführen)

g) Welche Aussage über das aktuelle Arbeitsverzeichnis (Current Working Directory) trifft zu?

2 Punkte

- ☐ Jedem UNIX-Benutzer ist zu jeder Zeit ein aktuelles Verzeichnis zugeordnet.
- ☐ Pfadnamen, die nicht mit dem Zeichen '/' beginnen, werden relativ zu dem aktuellen Arbeitsverzeichnis interpretiert.
- ☐ Mit dem Systemaufruf chdir() kann das aktuelle Arbeitsverzeichnis durch den Vaterprozess verändert werden.
- ☐ Besitzt ein UNIX-Prozess kein Current Working Directory, so beendet sich der Prozess mit einem Segmentation Fault.

h) Welche Aussage über den Rückgabewert von `fork()` ist richtig?

2 Punkte

- ☐ Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ☐ Der Kind-Prozess bekommt die Prozess-ID des Vater-Prozesses.
- ☐ Im Fehlerfall wird im Kind-Prozess -1 zurückgeliefert.
- ☐ Der Rückgabewert ist in jedem Prozess (Kind und Vater) jeweils die eigene Prozess-ID.

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

- ☐ Ein Symbolic Link kann nicht auf Dateien anderer Dateisysteme verweisen.
- ☐ 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.
- ☐ Für jede reguläre Datei existiert mindestens ein Hard Link im selben Dateisystem.

j) Welche Aussage zum Thema Betriebsarten ist richtig?

2 Punkte

- ☐ Beim Stapelbetrieb können keine globalen Variablen existieren, weil alle Daten im Stapel-Segment (Stack) abgelegt sind.
- ☐ Echtzeitsysteme findet man hauptsächlich auf großen Serversystemen, die eine enorme Menge an Anfragen zu bearbeiten haben.
- ☐ Mehrzugangsbetrieb ist nur in Verbindung mit CPU- und Speicherschutz sinnvoll realisierbar.
- ☐ Mehrprogrammbetrieb ermöglicht die simultane Ausführung mehrerer Programme innerhalb desselben Prozesses.

k) Welche Aussage zu Terminvorgaben in Echtzeitsystemen ist korrekt?

2 Punkte

- ☐ Beim Überschreiten einer weichen Terminvorgabe wird das Berechnungsergebnis wertlos; die Ausführung wird daher abgebrochen.
- ☐ Das Überschreiten einer harten Terminvorgabe kann zur Katastrophe führen; daher muss für die Anwendung eine Ausnahmebehandlung durchgeführt werden, die zu einem sicheren Zustand führt.
- ☐ Bei festen Terminvorgaben ist eine Terminverletzung tolerierbar, das Ergebnis verliert im Laufe der Zeit aber an Wert.
- ☐ Das Überschreiten einer harten Terminvorgabe ist nicht tolerierbar; daher muss das System in so einem Fall heruntergefahren 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) Man unterscheidet Traps und Interrupts. Welche der folgenden Aussagen sind richtig?

4 Punkte

- ○ Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- ○ Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
- ○ Normale Rechenoperationen können zu einem Trap führen.
- ○ Ein Trap wird immer unmittelbar durch die Aktivität des aktuell laufenden Prozesses ausgelöst.
- ○ Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie Interrupt einzuordnen.
- ○ Der Zugriff auf eine physikalische Adresse kann zu einem Trap führen.
- ○ Traps dürfen nicht nach dem Wiederaufnahmemodell behandelt werden, da ein Trap immer einen schwerwiegenden Fehler signalisiert.
- ○ Ganzzahl-Rechenoperationen können nicht zu einem Trap führen.

b) Welche der folgenden Aussagen zum Thema UNIX-Signale sind richtig?

4 Punkte

- ○ Durch Signale können Nebenläufigkeitsprobleme in grundsätzlich nicht-parallelen Programmen entstehen.
- ○ Es ist immer unnötig, das Signal `SIGCHLD` mittels `sigaction(2)` auf `SIG_IGN` zu setzen, da die Standardbehandlung von `SIGCHLD` bereits `SIG_IGN` ist.
- ○ Nach dem Laden eines anderen Programms mittels `exec(2)` wird die Signalmaske beibehalten.
- ○ Ein Prozess kann das Signal `SIGKILL` abfangen, falls die aktuelle Berechnung wichtiger ist als die Terminierungsanforderung des Benutzers.
- ○ `sleep(3)` kann während des Schlafens durch ein Signal unterbrochen werden, falls das Signal nicht blockiert wurde.
- ○ E/A-Operationen können problemlos in signalbehandelnden Funktionen genutzt werden, da E/A-Operationen typischerweise nicht-blockierend implementiert sind.
- ○ Nach dem Laden eines anderen Programms mittels `exec(2)` wird die Signalbehandlung immer auf `SIG_DFL` zurückgesetzt.
- ○ Das Signal `SIGCHLD` wird in der Standardkonfiguration ignoriert.





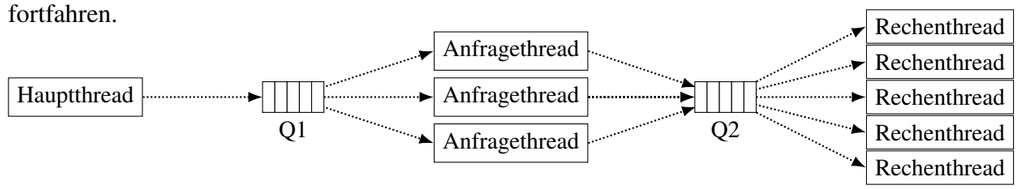
**Aufgabe 2: patriots (60.5 Punkte)**

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

Schreiben Sie ein Programm `patriots` (**pa**rallel **tri**angle c**o**un**t**er **s**ervice), das auf dem TCP/IPv6-Port 1337 (`LISTEN_PORT`) einen Dienst anbietet, um die Anzahl der ganzzahligen Koordinaten innerhalb und auf Dreiecken zu berechnen, die vom Clienten bereitgestellt werden.

Dazu sendet der Client zeilenweise Dreiecke im Format *(x1,y1),(x2,y2),(x3,y3)*. Nach dem letzten Dreieck sendet der Client eine Leerzeile. Der Server summiert die gefundenen Punkte aller Dreiecke auf und informiert den Clienten über die aktuelle Anzahl der gefundenen Punkte. Die Berechnung selbst soll mithilfe der Funktion `countPoints()` des vorgegebenen Moduls `triangle.o` geschehen (siehe angehängte Manpage `triangle(3)`).

Zur Vereinfachung dürfen Sie annehmen, dass der Client Zeilen mit einer Maximallänge von `MAX_LINE` Zeichen sendet. Entspricht eine Zeile nicht dem erwarteten Format, dann soll der Server einen entsprechenden Hinweis an den Clienten senden und mit der Bearbeitung der nächsten Zeile fortfahren.




Um die Anfragen mit maximaler Geschwindigkeit abarbeiten zu können, sollen zwei getrennte Threadpools verwendet werden:

Der *Hauptthread* nimmt die Verbindungsanfragen entgegen und fügt sie in den Ringpuffer Q1 ein.

*Threadpool 1* besteht aus `REQUEST_THREADS` Anfragethreads (in obiger Illustration: 3), die jeweils Verbindungen aus dem Ringpuffer Q1 entnehmen, die Arbeitspakete (**`struct`** `work_package`, pro Dreieck ein Arbeitspaket) zusammensetzen und in den Ringpuffer Q2 speichern.

Im Anschluss an das Auslesen aller Dreiecke vom Clienten sendet der Anfragethread den aktuellen Zwischenstand der Berechnung an den Clienten, sobald sich etwas ändert. Zusätzlich soll dem Clienten die Anzahl der noch zu berechnenden Dreiecke mitgeteilt werden. Hierzu signalisiert ihm der Rechenthread mittels eines Semaphors, dass aktualisierte Werte vorliegen.

*Threadpool 2* besteht aus `CALC_THREADS` Rechenthreads (in obiger Illustration: 5), die jeweils Dreiecke aus Q2 entnehmen und die eigentliche Zählung der Punkte mittels `countPoints()` durchführen. Sobald `countPoints()` die berechneten Werte liefert, signalisiert der Rechenthread dem Anfragethread das Vorliegen neuer Werte.

Die Kommunikation zwischen Anfrage- und Rechenthreads geschieht mittels gemeinsamem Speicher (ein **`struct`** `state` pro Verbindung). Nutzen Sie ausschließlich nicht-blockierende Synchronisation (bspw. die `__sync_*`-Operationen, vgl. Makros `AR`, `FAA` und `FAS`) zur Konsistenzsicherung gemeinsamer Datenstrukturen. Semaphore dürfen ausschließlich zur Signalisierung zwischen den Rechen- und Anfragethreads eingesetzt werden.

Stellen Sie sicher, dass der Server nicht aufgrund unterbrochener Verbindungen terminiert.

Hinweise:

- Beide Ringpuffer haben eine Kapazität von `BUFFER_SIZE` Einträgen.
- Anders als in der Übungsaufgabe speichert die vorgegebene Ringpuffer-Implementierung **`void`***. Speichern Sie ausschließlich Zeiger in diesem Ringpuffer. Rufen Sie vor dem Einfügen in Q1 `fdopen(3)` auf.
- Gehen Sie davon aus, dass alle Operationen des Ringpuffers ohne weitere Synchronisation nebenläufig genutzt werden können.
- Nutzen Sie das bereitgestellte Macro `AR` zum atomaren Auslesen von Variablen.
- Ihnen steht das aus der Übung bekannte Semaphor-Modul zur Verfügung. Die Schnittstelle finden Sie im folgenden Programmgerüst nach den **`#include`**-Anweisungen.

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

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

#include "triangle.h"
#include "bbuffer.h"
#include "sem.h"

/* Funktionen aus sem.h */
SEM *semCreate(int initVal);
void semDestroy(SEM *sem);
void P(SEM *sem);
void V(SEM *sem);

static const uint16_t LISTEN_PORT     = 1337;
static const size_t   MAX_LINE        = 2048;
static const size_t   BUFFER_SIZE     = 128;
static const size_t   CALC_THREADS    = 5;
static const size_t   REQUEST_THREADS = 3;

#define AR(X) __sync_fetch_and_add(&(X), 0)
#define FAA __sync_fetch_and_add
#define FAS __sync_fetch_and_sub

static void die(const char message[]) {
  perror(message); exit(EXIT_FAILURE);
}

typedef struct state {
  unsigned long boundary;
  unsigned long interior;
  unsigned long pending_triangles; /**< Noch zu bearbeitende Dreiecke */
  SEM *notify; /**< Signalisierungssemaphore für Anfragethread */
} state;

typedef struct work_package {
  struct triangle tri; /**< Tatsächliche Arbeit */
  struct state    *st; /**< state der zugehörigen Anfrage */
} work_package;
```





```c
// Funktions- & Strukturdekl., globale Variablen, etc.

// Hauptfunktion (main)

  // Ringpuffer initialisieren

  // Signalbehandlung aufsetzen
```

```
  // Threads starten
```

```
  // Socket erstellen und für Verbindungsannahme vorbereiten
```





```
  // Verbindungen annehmen und in den Puffer legen
```

```
// Ende Hauptfunktion
```

**M:**

```
// Funktion Anfragethread

    // Verbindung auslesen

    // geteilten Zustand anlegen & initialisieren

    // Arbeitspakete vom Clienten einlesen
```





```
    // Zwischenstände an Client senden

// Ende Funktion Anfragethread
```

**V:**

```
// Funktion Rechenthread + Hilfsfunktionen
```

**R:**





Schreiben Sie ein Makefile, welches die Targets `all` und `clean` unterstützt. Ebenfalls soll ein Target `patriots` unterstützt werden, welches das Programm `patriots` baut. Greifen Sie dabei stets auf Zwischenprodukte (z.B. `patriots.o`) zurück. Gehen Sie davon aus, dass die vorgegebenen Module `sem.o`, `bbuffer.o` und `triangle.o` stets vorliegen und daher nicht erzeugt werden müssen.

Das Target `clean` soll alle erzeugten Zwischenergebnisse und das Programm `patriots` löschen. Definieren und nutzen Sie dabei die Variablen `CC` und `CFLAGS` konventionskonform. Achten Sie darauf, dass das Makefile ohne eingebaute Variablen und Regeln (Aufruf von `make -Rr`) funktioniert!

**Mk:**

**Aufgabe 3: Adressräume & Freispeicherverwaltung (17 Punkte)**

1) Ein in der Praxis häufig eingesetztes Verfahren zur Verwaltung von freiem Speicher ist das *Buddy*-Verfahren.

Nehmen Sie einen Speicher von 1024 Bytes an und gehen Sie davon aus, dass die Freispeicher-Verwaltungsstrukturen separat liegen. Initial ist bereits ein Datenblock der Größe 128 Bytes vergeben worden. Ein Programm führt nacheinander die im folgenden Bild angegebenen Anweisungen aus. (11 Punkte)

⓪ p0 = malloc(100); // 0 (initial vergebener Block)

① p1 = malloc(50);

② p2 = malloc(110);

③ p3 = malloc(32);

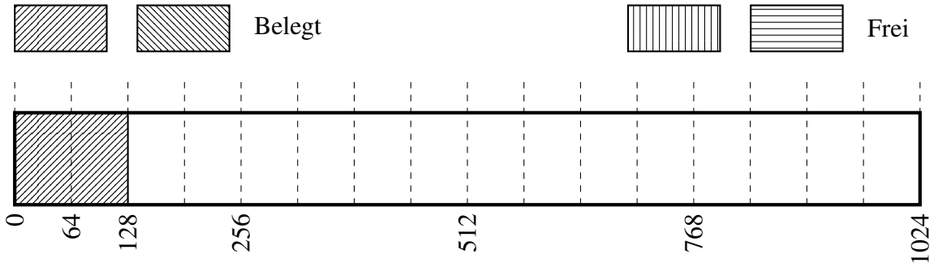
④ free(p1);

⑤ p4 = malloc(386);

⑥ free(p3);

⑦ free(p4);

Tragen Sie hinter den obigen Anweisungen jeweils ein, welches Ergebnis die `malloc()` - Aufrufe zurückliefern. Skizzieren Sie in der folgenden Grafik, wie der Speicher nach **Schritt** ⑤ aussieht, und tragen Sie in der Tabelle den aktuellen Zustand der Lochliste nach **jedem** Schritt ein. Für Löcher gleicher Größe schreiben Sie die Adressen einfach nebeneinander in die Tabellenzeile (es ist nicht notwendig, verkettete Buddys wie in der Vorlesung beschrieben einzutragen).




| Größe der Buckets | initial ⓪ | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ |
|---|---|---|---|---|---|---|---|---|
| $2^5$ | | | | | | | | |
| $2^6$ | | | | | | | | |
| $2^7$ | 128 | | | | | | | |
| $2^8$ | 256 | | | | | | | |
| $2^9$ | 512 | | | | | | | |
| $2^{10}$ | | | | | | | | |





2) 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)

3) 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 4: Das ABA-Problem (12.5 Punkte)**

Die untenstehende Implementierung des Ihnen aus der Übung bekannten Ringpuffers (`jbuffer`) leidet unter einem ABA-Problem. Beschreiben Sie ein Szenario mithilfe eines Ringpuffers der Größe 2 und zwei konkurrierenden Threads `T1` und `T2`, in dem ein ABA-Problem auftritt. Geben Sie die zum Nachvollziehen notwendigen Schritte analog der Vorgabe auf der nächste Seite an. Tragen Sie ebenso den Inhalt der Datenstrukturen **nach** dem jeweiligen Schritt in die bereitgestellten Felder ein.

Sollten Sie Zustände innerhalb einer Funktionsausführung beschreiben, dann vermerken Sie die beschriebene Stelle eindeutig. Vermerken Sie die Rückgabewerte von fertiggestellten Aufrufen von `bbGet()`.

*Hinweis:* Der Ringpuffer ist – wie auch der Ringpuffer der Übungsaufgabe – für mehrere nebenläufige Leser und einen Schreiber ausgelegt.

```
1  struct BNDBUF {
2      size_t size;          /*< Maximum number of elements */
3      volatile size_t r, w; /*< Read and write position    */
4      SEM *full, *free;
5      int data[];
6  };
7
8  // Adds an element to a bounded buffer.
9  void bbPut(BNDBUF *bb, int value) {
10     P(bb->free);
11     bb->data[bb->w] = value;
12     bb->w = (bb->w + 1) % bb->size;
13     V(bb->full);
14 }
15
16 // Retrieves an element from a bounded buffer.
17 int bbGet(BNDBUF *bb) {
18     int    value;
19     size_t pos, nextPos;
20     P(bb->full);
21     do {
22         pos = bb->r;
23         nextPos = (pos + 1) % bb->size;
24         value = bb->data[pos];
25     } while(__sync_bool_compare_and_swap(&bb->r, pos, nextPos) == 0);
26     V(bb->free);
27     return value;
28 }
```





Ausgangszustand: 2 von 2 Einträgen belegt