

Aufgabe 1: Ankreuzfragen (28 Punkte)

1) Einfachauswahlfragen (20 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 Programmunterbrechungen ist richtig?

2 Punkte

- Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.
- Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Interrupts immer an den gleichen Stellen auf.
- Der Zugriff auf eine logische Adresse kann zu einem Trap führen.
- Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar.

b) Gegeben seien die folgenden Präprozessor-Makros:

```
#define SUB(a, b) a - b
```

```
#define MUL(a, b) a * b
```

Was ist das Ergebnis des folgenden Ausdrucks? $4 * MUL (SUB(3,5) , 2)$

2 Punkte

- 16
- 2
- 16
- 2

c) Welche Aussage über Funktionen der `exec()`-Familie ist richtig?

2 Punkte

- Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- Der an `exec()` übergebene Funktionszeiger wird durch einen neuen Thread im aktuellen Prozess ausgeführt.
- Falls kein Fehler auftritt, kehrt der Aufruf von `exec()` nicht zurück
- `exec()` erzeugt einen neuen Kind-Prozess und startet darin das angegebene Programm.

d) 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.

e) 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.

f) 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.

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 eines Prozesses durch seinen Vaterprozess verändert werden.
- Besitzt ein UNIX-Prozess kein Current Working Directory, so beendet sich der Prozess mit einem Segmentation Fault.

h) Virtualisierung kann als Maßnahme gegen Verklemmungen genutzt werden. Warum?

2 Punkte

- Im Fall einer Verklemmung können zusätzliche virtuelle Betriebsmittel neu erzeugt werden. Diese können dann eingesetzt werden, um die fehlenden physikalischen Betriebsmittel zu ersetzen.
- Durch Virtualisierung kann man über Abbildungsvorgänge Zyklen, die auf der logischen Ebene vorhanden sind, auf der physikalischen Ebene auflösen.
- Eine Verklemmungsauflösung ist einfacher, weil virtuelle Betriebsmittel jederzeit ohne Schaden entzogen werden können.
- Durch Virtualisierung ist ein Entzug von physikalischen Betriebsmitteln möglich, obwohl dies auf der logischen Ebene unmöglich ist.

i) Welche der folgenden Aussagen zum Thema Threads ist richtig?

2 Punkte

- Bei User-Threads ist die Scheduling-Strategie nicht durch das Betriebssystem vorgegeben.
- Kernel-Threads können Multiprozessoren nicht ausnutzen.
- Die Umschaltung von User-Threads ist eine privilegierte Operation und muss deshalb im Systemkern erfolgen.
- Zu jedem Kernel-Thread gehört ein eigener, geschützter Adressraum.

j) Welche der folgenden Aussagen zum Thema Adressraumschutz ist richtig?

2 Punkte

- Adressraumschutz durch Abteilung eignet sich besonders für Systeme, die von mehreren Nutzern gleichzeitig verwendet werden.
- Beim Adressraumschutz durch Abteilung wird der logische Adressraum in mehrere Segmente mit unterschiedlicher Semantik unterteilt.
- Beim Einsatz von Segmentierung ist es möglich, dass dieselbe logische Adresse in unterschiedlichen logischen Adressräumen auf unterschiedliche physikalische Adressen verweist.
- In einem segmentierten Adressraum kann zur Laufzeit kein weiterer Speicher mehr dynamisch nachgefordert 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) Gegeben sei folgendes Programmfragment:

4 Punkte

```
static int a = 2018;

int f1 (const int *y) {
    static int b;
    int c;
    char *d = malloc(1337);
    int (*e)(const int *) = f1;
    y++;
    // ...
}
```

Welche der folgenden Aussagen zum obigen Programmfragment sind richtig?

- a liegt im Datensegment.
- c ist mit dem Wert 0 initialisiert.
- Die Speicherstelle, auf die d zeigt, verliert beim Rücksprung aus der Funktion f1() ihre Gültigkeit.
- b liegt im Stacksegment.
- e liegt im Stacksegment und zeigt in das Textsegment.
- Die Anweisung y++ führt zu einem Laufzeitfehler, da y konstant ist.
- d ist ein Zeiger, der in den Heap zeigt.
- y liegt im Stacksegment.

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

4 Punkte

- Der UNIX-Systemaufruf `fork(2)` lädt eine Programmdatei in einen neu erzeugten Prozess.
- Ein Prozess ist ein Programm in Ausführung - ein Prozess kann aber auch mehrere verschiedene Programme ausführen.
- Wenn ein Programm nur einen aktiven Ablauf enthält, nennt man diesen Prozess, enthält das Programm mehrere Abläufe, nennt man diese Threads.
- Ein Prozess kann durch mehrere Programme ausgeführt werden.
- Mit Hilfe des Systemaufrufs `exec(2)` wird das bestehende Programm im aktuell laufenden Prozess ersetzt.
- Mit Hilfe von Threads kann ein Prozess mehrere Programme gleichzeitig ausführen.
- Der UNIX-Systemaufruf `fork(2)` erzeugt eine exakte Kopie (mit Ausnahme der PID) des aufrufenden Prozesses.
- Ein Programm kann durch mehrere Prozesse gleichzeitig ausgeführt werden.

Aufgabe 2: dsqvo (61 Punkte)

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

Schreiben Sie ein Programm `dsqvo`, das auf dem TCP/IPv6-Port 1337 (PORT) einen Dienst anbietet, der es dem Nutzer erlaubt Logdateien zu anonymisieren. Dazu sendet der Client die zu anonymisierenden Loginhalte zeilenweise an den Server. Der Server anonymisiert die Inhalte zeilenweise und sendet das Ergebnis zurück an den Anfrager. Um das Supergrundrecht auf Sicherheit gewährleisten zu können, unterhält der Server zudem eine dauerhaft geöffnete Verbindung zu `weltraum.bnd.de` auf Port 80 und leitet den nicht-anonymisierten Inhalt dorthin weiter. Um die Last auf dem BND-Server gering zu halten, wird vor dem Weiterleiten eine Filterung auf Basis von Selektoren durchgeführt.

Das Hauptprogramm initialisiert zunächst alle benötigten Datenstrukturen, startet THREADS Fäden zur Verarbeitung von eingehenden Verbindungen und einen Faden zur Ausleitung an den BND und nimmt auf einem Socket Verbindungen an. Erfolgreich angenommene Verbindungen werden zur weiteren Verarbeitung in einen Ringpuffer (Verbindungspuffer) eingefügt.

Funktion `void* conn_thread(void *arg)`: Entnimmt die Verbindungsanfragen aus dem Verbindungspuffer und liest zeilenweise, bis der Client eine Leerzeile sendet. Zur Vereinfachung dürfen Sie annehmen, dass der Client Zeilen mit einer Maximallänge von `MAX_LINE` Zeichen sendet. Für jede eingelesene Zeile werden folgende Schritte durchgeführt:

1. Prüfen auf Übereinstimmung mit einem Selektor (Funktion `any_selector_matches()`) und ggf. Einfügen in einen zweiten Ringpuffer (Logpuffer) zur Weiterleitung an `weltraum.bnd.de:80`
2. Anonymisieren und Zurücksenden der Logzeile:
Logzeilen bestehen jeweils aus Beschreibungstext, in dem **eine beliebige Anzahl von IP-Adressen** (im Beispiel hervorgehoben) enthalten sein können. Zur Vereinfachung dürfen Sie davon ausgehen, dass die IP-Adressen immer von `<>` umfasst sind, `<>` in der Zeile sonst nicht vorkommt und eine Zeile nie mit einer IP-Adresse beginnt.

```
1 Incoming requests from <46.38.239.227> and <131.188.34.58>
2 Sending file /home/chris/sp-klausur.tex to <46.38.239.227>
```

Die IP-Adressen sollen durch den Text `REMOVED` ersetzt werden:

```
1 Incoming requests from <REMOVED> and <REMOVED>
2 Sending file /home/chris/sp-klausur.tex to <REMOVED>
```

Funktion `bool any_selector_matches(const char *line)`: Prüft, ob die übergebene Logzeile `line` zu mindestens einem Selektor passt. Bei den zum Vergleich zu nutzenden Selektoren handelt es sich um Zeilen in (einer oder mehreren) Dateien im Verzeichnis `/selectors/` (`SELECTOR_DIR`). Das Verzeichnis soll (nicht-rekursiv & unter Nutzung von `readdir(3)`) durchlaufen werden; Sie dürfen davon ausgehen, dass in dem Verzeichnis nur reguläre Dateien liegen (kein `stat` notwendig). Für jede gelesene (Selektor-)Zeile (maximale Länge `MAX_LINE`) soll die (vorgegebene, vgl. folgende Seite) Vergleichsfunktion `selector_matches` aufgerufen werden.

Funktion `void* bnd_thread(void *arg)`: Leitet kontinuierlich alle Einträge aus dem Logpuffer an `weltraum.bnd.de:80` durch Schreiben auf den (vorgegebenen) `FILE* bnd`.

Hinweise:

- Anders als in der Übungsaufgabe speichert die teilweise vorgegebene Implementierung des Ringpuffers `void*` (vgl. nachfolgende Teilaufgabe). Rufen Sie vor dem Einfügen `fdopen(3)` auf und nutzen Sie auf Empfängerseite `fileno(3) & dup(2)`.
- Der Verbindungsaufbau zu `weltraum.bnd.de:80` (`FILE *bnd`) ist vorgegeben und wird bereits am Anfang des Hauptprogramms durchgeführt.
- Alle genutzten Ringpuffer haben eine Größe von `BUFFER_SIZE` Einträgen.

// Threads starten

// Socket erstellen und für Verbindungsannahme vorbereiten



// Verbindungen annehmen und in den Puffer legen

}

// Ende Hauptfunktion



M:

// Funktion Anfragethread

// Verbindung auslesen & vorbereiten

// Anfrage zeilenweise einlesen

// Zeile anonymisieren

// Funktion any_selector_matches

// SELECTOR_DIR (nicht-rekursiv) durchlaufen



// Selektor-Datei zeilenweise durchsuchen



```
// Funktion Weiterleitungsthread
```


 S:

Ringpuffer (9 Punkte)

Implementieren Sie **die Funktion bbGet** des Ihnen aus der Übung bekannten Ringpuffers `jbuffer`. Anders als in der Übungsaufgabe soll die Ringpuffer-Implementierung **void*** speichern. Der Puffer soll für mehrere konkurrierende Schreiber-Threads und mehrere konkurrierende Leser-Threads ausgelegt sein und FIFO-Eigenschaften aufweisen. Die Konsumenten (= Aufrufer von `bbGet`) sollen untereinander nicht-blockierend koordiniert werden.

Benutzen Sie hierfür die CAS-Funktion

```
bool __sync_bool_compare_and_swap(type *ptr, type oldval, type newval)
```

Achten Sie dabei darauf, dass mehrere Konsumenten gleichzeitig den kritischen Abschnitt durchlaufen können (keine Locks!).

Um einem möglichen ABA-Problem zu entgegnen, soll der Ringpuffer mittels **Generationszähler implementiert** werden.

```
#include "bbuffer.h"
```

```
#include "sem.h"
```

```
#include <limits.h>
```

```
#include <stdint.h>
```

```
#define CAS __sync_bool_compare_and_swap
```

```
struct BNDBUF {
```

```
    size_t size;
```

```
    volatile size_t r; // Carries an implicit generation counter
```

```
    volatile size_t w;
```

```
    SEM *full;
```

```
    SEM *free;
```

```
    SEM *lock;
```

```
    void* data[];
```

```
};
```

```
/** Creates a new bounded buffer.
```

```
 * On errors during initialization, the implementation frees all
```

```
 * resources already allocated by then and returns NULL.
```

```
 */
```

```
BNDBUF *bbCreate(size_t size);
```

```
void bbDestroy(BNDBUF *bb);
```

```
void bbPut(BNDBUF *bb, void *value) {
```

```
    P(bb->free);
```

```
    P(bb->lock);
```

```
    bb->data[bb->w] = value;
```

```
    bb->w = (bb->w + 1) % bb->size;
```

```
    V(bb->lock);
```

```
    V(bb->full);
```

```
}
```


