Aufgabe 1.1: Einfachauswahl-Fragen (22 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-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 Adressraumverwaltung ist richtig? 3 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 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.

b) Man unterscheidet Traps und Interrupts. Welche Aussage ist richtig? 3 Punkte

- ❏ Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- ❏ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleicher Eingabe treten Interrupts immer an den gleichen Stellen auf.
- ❏ 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.
- ❏ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.

c) 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)

d) Wodurch kann es zu Seitenflattern kommen? 2 Punkte

- ❏ wenn sich zu viele Prozesse im Zustand blockiert befinden
- ❏ wenn bei einer nicht-verdrängenden Scheduling-Strategie die Zahl der von den Prozessen aktiv genutzten Seiten die Zahl der verfügbaren Seitenrahmen übersteigt
- ❏ durch Programme, die eine Defragmentierung auf der Platte durchführen
- ❏ wenn zu viele Prozesse im Rahmen der mittelfristigen Einplaung ausgelagert (swap-out) wurden

e) Welche Aussage zu Speicherzuteilungsverfahren ist richtig? 2 Punkte

- ❏ best-fit ist in jedem Fall das beste Verfahren
- ❏ buddy-Verfahren sind nur bei sehr leistungsfähigen Rechnern einsetzbar, weil sie aufwändig in der Berechnung sind.
- ❏ die worst-fit-Strategie kann mit der kürzesten Antwortzeit einen ausreichend großen Speicherbereich liefern
- ❏ die worst-fit-Strategie ist lediglich theoretisch interessant, da es in der Praxis nie sinnvoll ist, den am schlechtesten passenden Speicherplatz zuzuweisen.

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

- ❏ Bei RAID 4 Systemen wird Paritätsinformation gleichmäßig über alle beteiligten Platten verteilt.
- ❏ 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 und 5 darf eine bestimmte Menge von Festplatten nicht überschritten werden, da es sonst nicht mehr möglich ist, die Paritätsinformation zu bilden.
- ❏ RAID 0 erzielt Fehlertoleranz durch das Verteilen der Daten auf mehrere Platten.

g) Wodurch kann Nebenläufigkeit in einem System entstehen? 2 Punkte

- ❏ durch Seitenflattern
- ❏ durch langfristiges Scheduling
- ❏ durch Compiler-Optimierungen
- ❏ durch Threads auf einem Monoprozessorsystem mit präemptivem Scheduling

h) Welche Aussage zum Thema "Aktives Warten" ist richtig? 2 Punkte

- ❏ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit
- ❏ Auf Mehrprozessorsystemen ist aktives Warten unproblematisch und deshalb dem passiven Warten immer vorzuziehen
- ❏ Aktives Warten darf bei nicht-verdrängenden Scheduling-Strategien auf einem Monoprozessorsystem nicht verwendet werden
- ❏ Bei verdrängenden Scheduling-Strategien verzögert aktives Warten nur den betroffenen Prozess, behindert aber nicht andere

i) Wozu dient der Maschinenbefehl *cas* (compare-and-swap)? 2 Punkte

- ❏ Um auf einem Multiprozessorsystem einfache Modifikationen an Variablen ohne Sperren implementieren zu können.
- ❏ Um bei Monoprozessorsystemen Interrupts zu sperren.
- ❏ Um bei der Implementierung von Schlossvariablen (Locks) aktives Warten zu vermeiden.
- ❏ Zur Realisierung einer effizienten Verdrängungssteuerung bei einseitiger Synchronisation.

j) 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.

Aufgabe 1.2: Mehrfachauswahl-Fragen (8 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe sind jeweils *m* Aussagen angegeben, *n* ($0 \leq n \leq m$) Aussagen davon sind richtig. Kreuzen Sie **alle richtigen** Aussagen an. Jede korrekte Antwort in einer Teilaufgabe gibt einen halben Punkt, jede falsche Antwort einen halben 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

- ❍ 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* überführt werden.
- ❍ Prozesse im Zustand *gestoppt* sind der langfristigen Einplanung zuzuordnen.
- ❍ 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.
- ❍ Für die mittelfristige Einplanung muss das Betriebssystem die Umlagerung (engl. swapping) von kompletten Programmen bzw. logischen Adressräumen unterstützen.
- ❍ 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.
- ❍ Verdrängende Prozesseinplanung bedeutet, dass das Eintreten des erwarteten Ereignisses unmittelbar die Einlastung des wartenden Prozesses bewirkt.

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

4 Punkte

- ❍ Da virtuelle Adressräume direkt durch die MMU unterstützt werden, können zur Laufzeit keine zusätzlichen Kosten gegenüber logischen Adressräumen auftreten.
- ❍ Bei einer seitenorientierten Adressraumorganistaion muss die Größe jeder Seite in der Seitentabelle gespeichert werden.
- ❍ Der Adressraumschutz durch Abteilung ermöglicht die Isolation mehrerer Prozesse voneinander.
- ❍ Bei Adressraumschutz durch Abteilung werden die Betriebssystemdaten vor Zugriffen aus dem Anwendungsprogramm geschützt.
- ❍ Ein Seitenfehler wird abhängig von der Ursache nach dem Fortsetzungs- oder dem Beendigungsmodell behandelt.
- ❍ Ein Adressraumwechsel ist eine privilegierte Operation und kann daher nur durch das Betriebssystem vorgenommen werden.
- ❍ Für den Adressraumschutz durch Eingrenzung ist zwingend eine MMU erforderlich.
- ❍ Zur Implementierung von logischen Adressräumen ist spezielle Hardwareunterstützung nötig.

Aufgabe 2: chatserver (60 Punkte)

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

Schreiben Sie ein mehrfädiges Programm `chatserver`, das einen Gruppenkommunikationsdienst anbietet. Teilnehmer (Clients) können sich über den TCP-Port 6670 mit dem Dienst verbinden und untereinander Textnachrichten austauschen. Der Server fungiert als sogenanntes Relay: Jede Nachricht, die er entgegennimmt, leitet er an alle aktuell verbundenen Teilnehmer weiter.

Das Kommunikationsprotokoll ist wie folgt definiert:
Der Dienst kann von maximal 100 (`MAX_CLIENTS`) Clients gleichzeitig genutzt werden. Ein Client muss hierfür eine TCP-Verbindung auf dem angegebenen Port herstellen und an den Server eine Zeile mit seinem Nutzernamen (Maximallänge: 8 Zeichen, `MAX_USER_LEN`) senden. Zur Vereinfachung kann angenommen werden, dass der Server nicht auf Namenskollisionen oder überlange Zeilen prüft. Der Client kann nun an der Gruppenkommunikation teilnehmen, bis die Socket-Verbindung getrennt wird. Jede weitere Textzeile, die er an den Server sendet (Maximallänge: 1024 Zeichen, `MAX_MSG_LEN`), wird von diesem mit einem Präfix versehen und an alle Teilnehmer (auch an den Autor selber) weitergeleitet. Das Präfix enthält den Namen des Autors in spitzen Klammern, gefolgt von einem Tabulator-Zeichen. Eine Zeile, die vom Server versendet wird, könnte zum Beispiel so aussehen:

```
<Faust> Das also war des Pudels Kern!
```

Das Programm (`chatserver.c`) soll folgendermaßen strukturiert sein:

- Das Hauptprogramm nimmt auf dem angegebenen Port Verbindungen an. Jede akzeptierte Verbindung wird in Form eines `FILE *` in einen Ringpuffer ("Verbindungs-Puffer") mit einer Kapazität von 10 Elementen (`CONN_BUF_SIZE`) eingetragen. Die weitere Abarbeitung wird durch separate Threads durchgeführt. Hierfür werden zu Beginn `MAX_CLIENTS` Lese-Threads und ein Sende-Thread gestartet, die dann endlos laufen.
- Lese-Thread (Funktion `void *receive(void *arg)`): Entnimmt jeweils eine Client-Verbindung aus dem Verbindungs-Puffer und bedient diese. Jede gelesene Textzeile wird wie im Protokoll spezifiziert mit einem Präfix versehen und zur Ausgabe an die anderen Teilnehmer in einen zweiten Ringpuffer ("Nachrichten-Puffer") der Größe 128 (`LINE_BUF_SIZE`) geschrieben.
  Die Client-Verbindungen werden in einem globalen Feld mit einem Eintrag pro Lese-Thread abgelegt. Ist ein Lese-Thread nicht aktiv, trägt er `NULL` ein.
- Sende-Thread (Funktion `void *broadcast(void *arg)`): Entnimmt jeweils eine Nachricht aus dem Nachrichten-Puffer und sendet sie an alle aktuell verbundenen Clients.

Implementieren Sie ferner ein generisches FIFO-Ringpuffer-Modul (`bbuffer.c`), das Elemente vom Typ `void *` verwaltet. Der Puffer soll so synchronisiert sein, dass Über- und Unterlaufsituationen verhindert werden und dass seine Operationen im gegebenen Anwendungskontext threadsicher sind.

Der Ringpuffer soll die folgenden Funktionen anbieten:

```
BNDBUF *bb_init(size_t size);          - Konstruktur (liefert im Fehlerfall NULL)
void bb_add(BNDBUF *bb, void *value); - fügt ein Element in den Ringpuffer ein
void *bb_get(BNDBUF *bb);              - entfernt ein Element aus dem Ringpuffer
```

Sie können für Synchronisationszwecke in allen Programmteilen die existierenden Funktionen
`SEM *sem_init(int initVal); void P(SEM *sem); void V(sem *sem);`
eines **nicht** selbst zu implementierenden Semaphor-Moduls benutzen. Dabei können Sie davon ausgehen, dass `sem_init()` nie fehlschlägt.

Auf den folgenden Seiten finden Sie ein Gerüst für das beschriebene Programm. In den Kommentaren sind nur die wesentlichen Aufgaben der einzelnen zu ergänzenden Programmteile beschrieben, um Ihnen eine gewisse Leitlinie zu geben. Es ist überall sehr großzügig Platz gelassen, damit Sie auch weitere notwendige Anweisungen entsprechend Ihrer Programmierung einfügen können.

Einige wichtige Manual-Seiten liegen bei - es kann aber durchaus sein, dass Sie bei Ihrer Lösung nicht alle diese Funktionen oder gegebenenfalls auch weitere Funktionen benötigen.

```
/*************************************************************
 * Datei chatserver.c
 *************************************************************/

#include "bbuffer.h"
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>

// Konstanten
#define MAX_CLIENTS    100
#define MAX_USER_LEN     8
#define MAX_MSG_LEN   1024
#define CONN_BUF_SIZE   10
#define LINE_BUF_SIZE  128

// Globale Variablen, Funktionsdeklarationen usw.
```



```
// Funktion main()

    // Allgemeine Vorbereitungen

    // Lese-Threads und Sende-Thread starten
```

I:

```
// Socket erstellen und auf Verbindungsannahme vorbereiten
```

O:


```
// Client-Verbindungen annehmen
```

```
// Ende Funktion main()
```

S:

```
// Lese-Thread: Funktion receive()
```

R:


```
// Sende-Thread: Funktion broadcast()
```

B:

```
/*************************************************************
 * Datei bbuffer.c
 *************************************************************/

#include "bbuffer.h"
#include "sem.h"
#include <stdlib.h>

typedef struct BNDBUF {
    void **data;
    int size;
    volatile int readPos;
    volatile int writePos;
    SEM *full;
    SEM *free;
    SEM *readLock;
    SEM *writeLock;
} BNDBUF;
```

// Funktion bb_init()

// Funktion bb_add()

// Funktion bb_get()

F:

Aufgabe 3: (18 Punkte)

a) Was versteht man unter einem *Inode* in einem Unix/Linux-Dateisystem? (2 Punkte)

b) Wie werden Dateiattribute und Dateiinhalt im Gegensatz zu Linux in einem Windows-NT-Dateisystem (NTFS) verwaltet? (2 Punkte)

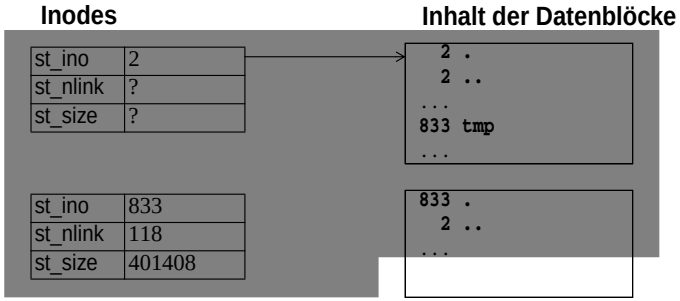
c) Gegeben ist die folgende Ausgabe des Kommandos **`ls -alRi /tmp/sp`** (rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/sp` mit Angabe der Inode-Nummer) auf einem Linux-System. (11 Punkte)

```
fauixx>  ls -alRi /tmp/sp
/tmp/sp:
total 408
 91 drwxrwxr-x   4 jklein i4staff   4096 Jul 19 15:17 .
833 drwxrwxrwt 118 root   root    401408 Jul 19 15:30 ..
 96 drwxrwxr-x   2 jklein i4staff   4096 Jul 19 15:21 dir1
 98 drwxrwxr-x   2 jklein i4staff   4096 Jul 19 15:22 dir2

/tmp/sp/dir1:
total 68
 96 drwxrwxr-x 2 jklein i4staff  4096 Jul 19 15:21 .
 91 drwxrwxr-x 4 jklein i4staff  4096 Jul 19 15:17 ..
536 -rwxr--r-- 2 jklein i4staff 52224 Jul 19 15:20 datei1
258 -rw-rw-r-- 1 jklein i4staff    30 Jul 19 15:21 datei2

/tmp/sp/dir2:
total 68
 98 drwxrwxr-x 2 jklein i4staff  4096 Jul 19 15:22 .
 91 drwxrwxr-x 4 jklein i4staff  4096 Jul 19 15:17 ..
896 lrwxrwxrwx 1 jklein i4staff    14 Jul 19 15:22 dat1 -> ../dir1/datei2
536 -rwxr--r-- 2 jklein i4staff 52224 Jul 19 15:20 xxx
900 -rw-rw-r-- 1 jens   i4staff    30 Jul 19 15:22 yyy
```

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 Sie aus obiger Ausgabe entnehmen 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	

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

Aufgabe 4: (12 Punkte)

Beschreiben Sie in Stichworten die unterschiedlichen Arten ("Gewichtsklassen") von Prozessen/Threads (Eigenschaften, Vorteile, Nachteile).