182.109 Systemnahe Programmierung
2. Test

25. Jänner 2008

KNr. | MNr. | Zuname, Vorname

Ges.)(70)	1.)(15)	2.)(20)	3.)(20)	4.)(15)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Fork, Exec und Pipes (15)

Schreiben Sie ein Programm `EmphasiseStdout`, das mit den Namen eines auszuführenden Programmes (`prog`) aufgerufen wird:

```
EmphasiseStdout prog
```

Das angegebene Programm EmphasiseStdout soll dafür sorgen, dass das Programm `prog` aufgerufen wird und die Fehlermeldungen von `prog`, die auf `stderr` ausgegeben werden, auf die standard Ausgabe (`stdout`) umgeleitet werden. Die standard Ausgabe (`stdout`) von `prog` soll ebenfalls auf stdout ausgegeben werden, allerdings soll diesen Ausgaben der Text "Standard Output: " vorangestellt werden.

Realisieren Sie das Programm `EmphasiseStdout` unter Verwendung von `fork()`, `exec()` und `unnamed Pipes` unter Beachtung der folgenden Punkte:

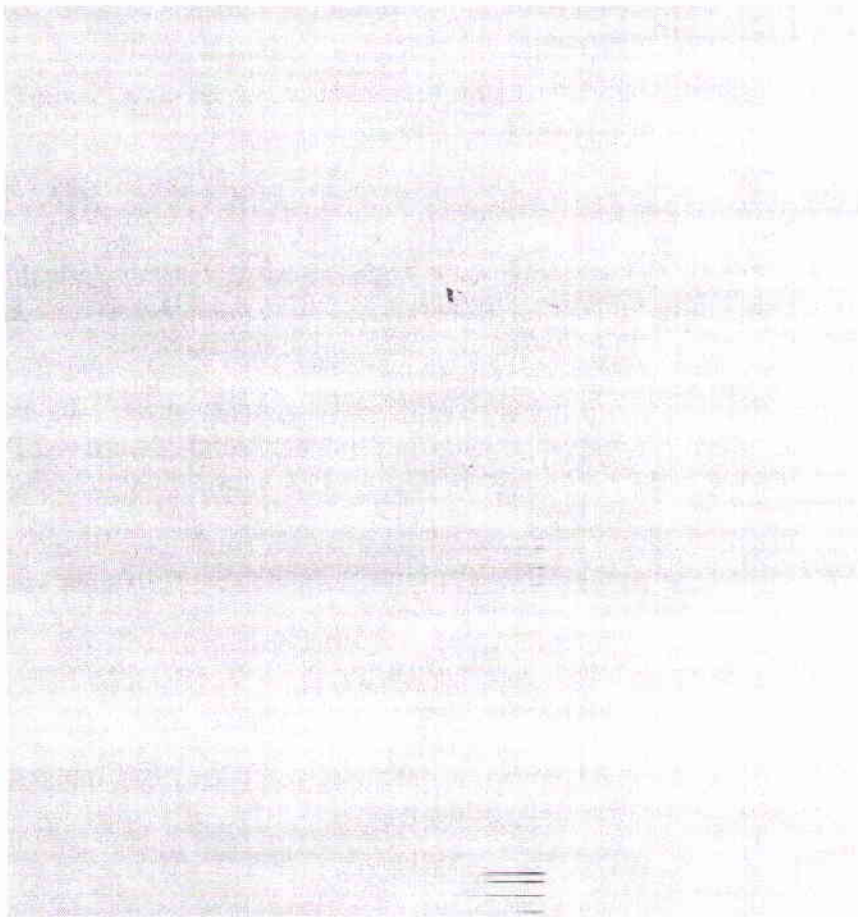
- Überprüfen Sie die korrekte Anzahl der Parameter.
- Verwenden Sie die Funktion `void BailOut(const char *szmsg)` für die Freigabe der Ressourcen im Fehlerfall. Die Funktion `BailOut` terminiert das Programm mit dem Exit-Code `EXIT_FAILURE`. Diese Funktion ist *nicht* zu implementieren!
- Verwenden Sie die Funktion `usage()`, um eine Usage-Meldung auszugeben, falls die Argumente nicht richtig angegeben worden sind. Die Funktion `void usage(void)` terminiert das Programm mit dem Exit-Code `EXIT_FAILURE` plus Fehlermeldung. Diese Funktion ist *nicht* zu implementieren!
- Das Programm `EmphasiseStdout` soll so lange in einer Schleife die Ausgaben von `prog` lesen und mit dem vorangestellten Text "Standard Output: " ausgeben bis `prog` terminiert *(EOF)*.
- Die Ausgaben von `prog` sind maximal 80 Bytes lang.
- Im fehlerfreien Fall soll das Programm `EmphasiseStdout` mit dem Wert `EXIT_SUCCESS` beendet und alle angelegten Ressourcen freigegeben werden.

Ergänzen Sie das Programmgerüst von EmphasiseStdout

```
/* ********************************************************** includes ***/
        /* includefiles muessen nicht angegeben werden */
/* ********************************************************** globals ***/
```

```
/* ******************************************************** prototypes ***/
    void BailOut(const char *szMessage);
    void usage(void);


/* ******************************************************** functions ***/
int main(int argc, char** argv)
{
```



2 Shared Memory und Semaphore (20)

Das Programm `lager-manager` dient zur Verwaltung des Zentrallagers einer grossen Baumarktkette. Das Programm wird von den Angestellten der verschiedenen Filialien und dem Zentrallagerleiter verwendet. Die Angestellten der Filialien können Artikel die im Lager vorhanden sind anfordern und somit aus dem Lager entfernen. Der Zentrallagerleiter kann neu eingekaufte Artikel dem Lager hinzufügen.

Es gibt in dem Lager 1000 verschiedende Artikel von denen jeweils maximal 200 Stück gelagert werden können.

Das Programm `lager-manager` besitzt die folgende Aufrufsyntax:

```
lager-manager [-h][-q] -a Artikelnummer [-n Anzahl]
```

- `-a Artikelnummer`: Mit diesem Parameter wird der Artikel identifiziert. Der Wertebereich für diesen Parameter ist zwischen 0 und 999.
- `-n Anzahl`: Dieser Parameter ist optional und bestimmt die Anzahl der Artikel für die gewünschte Operation. Der Wertebereich liegt zwischen 0 und 200. Wird dieser Wert nicht angegeben, wird er auf den Defaultwert 0 gesetzt.
- `-h`: Dies ist ein optionaler Parameter der bewirkt, dass der entsprechende Artikel in der entsprechenden Anzahl hinzugefügt wird. Wird dieser Parameter nicht anngegeben, so wird der entsprechende Artikel in der entsprechenden Anzahl entfernt.
- `-q`: Dies ist ein optionaler Parameter der bewirkt, dass der Lagerstand des aktuellen Artikels angezeigt wird. Wenn dieser Parameter angegeben wird, so werden die optionalen Paramter -n und -h ignoriert.

Die gewählte Operation wird nur dann ausgeführt, wenn die Menge eines Artikels im Lager dabei weder 0 unterschreitet noch 200 überschreitet. Wenn die Operation nicht ausgeführt werden kann soll eine Fehlermeldung ausgegeben werden.

Beispiele für gültige Aufrufe sind:

- `lager-manager -a 17 -n 9`

  Unter der Voraussetzung, dass mindestends 9 Artikel mit der Nummer 17 im Zentrallager vorhanden sind, entfernt dieser Aufruf 9 Artikel mit der Nummer 17 aus dem Zentrallager. Anssonsten wird eine Fehlermeldung ausgegeben.

- `lager-manager -h -a 17 -n 9`

  Unter der Voraussetzung, dass nicht mehr als 191 Artikel mit der Nummer 17 im Zentrallager vorhanden sind, fügt dieser Aufruf 9 Artikel mit der Nummer 17 zu dem Zentrallager hinzu. Anssonsten wird eine Fehlermeldung ausgegeben.

- `lager-manager -q -a 17`

  Gibt den Lagerstand der Artikel Nummer 17 zurück.

Implementieren Sie das Programm `lager-manager` mittels Shared Memories und Semaphoren. Da mehrere Angestellte und auch der Zentrallagerleiter gleichzeitig arbeiten, können mehrere nebenläufige Exekutionen von `lager-manager` existieren. Legen Sie die Informationen über den Lagerstand in einem Shared Memory ab, sodass diese Informationen allen Instanzen von `lager-manager` zur Verfügung stehen. Synchronisieren Sie die Instanzen mittels Semaphoren um konsistente Abfragen und Schreiboperationen sicherzustellen. Ist das Shared Memory noch nicht vorhanden, so erzeugen Sie dieses und setzen sie die Anzahl aller Artikel auf 0. Ebenso sind noch nicht angelegte Semaphore zu erzeugen und zu initialisieren.

Hinweise:

- Verwenden Sie `getopt` zur Argumentbehandlung und geben Sie bei Verletzung der Aufrufsyntax eine Usage-Meldung aus.
- Synchronisieren Sie den Zugriff der Prozesse auf das Shared Memory. Verwenden Sie eine möglichst geringe Anzahl von Semaphoren zur Synchronisation und achten Sie darauf, dass Ihre Lösung die maximale erlaubte Paralleliät der Prozesse zulässt!
- Beachten Sie bei der Synchronisation die unterschiedlichen Rollen des Zugriffs auf das Shared Memory. Bei einer Abfrage des Lagerstandes (Option -q) erfolgt ein Lesezugriff. Bei den anderen Operation erfolgt ein Lesezugriff (d.h., Lagerstand überprüfen) gefolgt von einem Schreibzugriff (Lagerstand ändern).
- Fragen Sie analog zu den Richtlinien der LU die Return-Werte von Funktionen auf Fehler-Codes ab. Sie können die Funktion `void error(char *msg)` benutzen, welche eine Fehlermeldung auf *stderr* ausgibt und einen Prozess anschließend terminiert. Diese Funktion brauchen Sie nicht zu implementieren.
- Sie brauchen angelegte Ressourcen nicht freizugeben.

lager-manager.c

```
#include <stdio.h>

#include <sem182.h>

#include <string.h>


// Konstanten und Datentyp fuer Shared Memory



#define MAX_ANZAHL      200

#define ARTIKEL         999
```

```
struct shm_struct {
lager int[ARTIKEL];
};

extern error(char *msg);

int main(int argc, char **argv) {
int i;

 // Argumentbehandlung
 while ((i=getopt(                                    ))!=EOF) {
```

3 Unix/Linux Mechanismen (20)

1) Welche beiden Fälle der Synchronisation unterscheidet Unix/Linux bei der Synchronisation im Kernel für den Zugriff auf gemeinsame Ressourcen?

Beschreiben Sie für jeden der beiden Fälle, wie die Synchronisation in Unix/Linux realisiert wird.

2) Welche Einträge in der `task_struct`-Struktur sind für die Unterstützung der Signalbehandlung in Unix/Linux notwendig? Welche Aufgaben erfüllen diese Einträge?

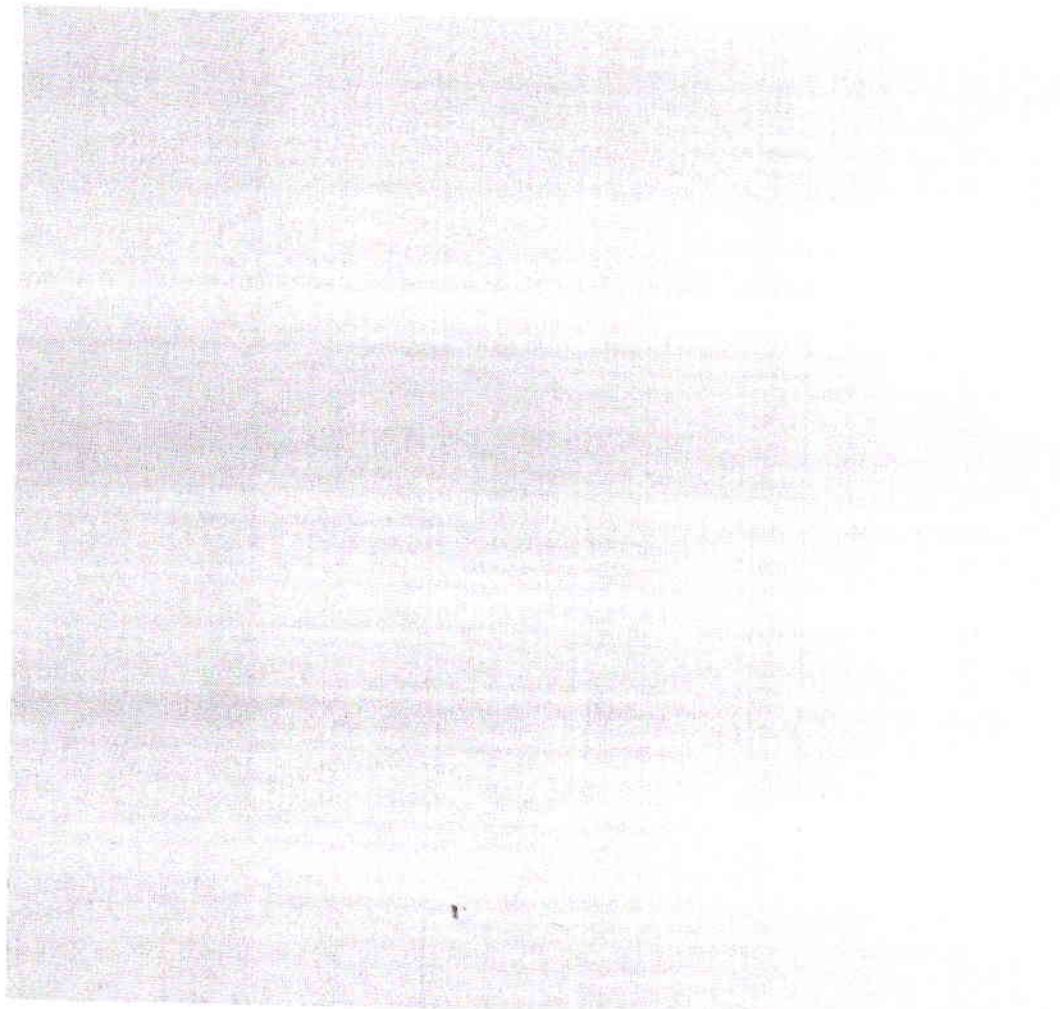
Welche unterschiedlichen Reaktionsmöglichkeiten kann ein Programmierer für das Eintreffen eines Signals programmieren?

3) Welche Parameter sind bei der Kreierung einse Sockets mittels *socket()* System Call anzugeben? Geben Sie für jeden dieser Parameter mindestens zwei Beispiele für mögliche Belegungen an.

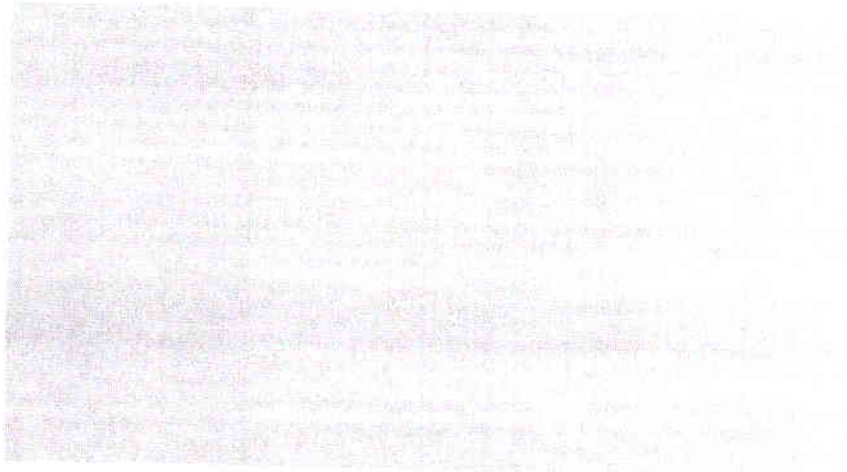
4) Nennen Sie die Anforderungen, die an ein Virtual-Memory Management gestellt werden.

4 Realisierung von Betriebssystemmechanismen (15)

1) Welche beiden grundlegenden Arten von Betriebssystemkernen werden unterschieden? Erläutern Sie die Vor- und Nachteile von beiden. (5)



2) Welche Services bietet das TCP-Protokoll im Gegensatz zum UDP-Protokoll? (4)



3) Wie unterscheiden sich "Software Devices" von regulären Devices? Nennen Sie zumindest 3 Beispiele für Software Devices. (3)

4) Was ist das Virtuelle Filesystem (VFS) in UNIX und wofür wird es verwendet? (3)