Betriebssysteme - Übung 11
(25 Punkte)

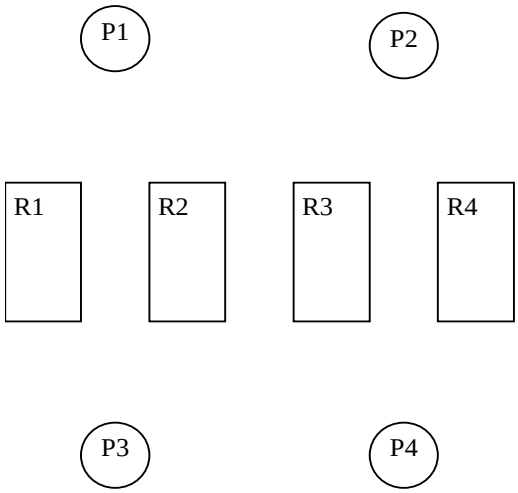
Thema: Deadlocks, Dateisysteme, Pipes

Hinweis: Es werden 5 Punkt abgezogen, wenn Sie die Abgabe nicht in dem gewünschten Format hochladen und weitere 2 Punkte falls Ihr Quelltext nicht ausreichend kommentiert, bzw. leicht verständlich ist. Nicht kompilierbare Aufgaben (auf den Linux Servern des ZIM) werden mit 0 Punkten bewertet. Zu allen Programmieraufgaben müssen Makefiles erstellt werden, und diese zusammen mit dem Quellcode und Lösungsbeschreibung in Moodle hochgeladen werden.

Die Aufgaben 1, 2 und 3 wurden so, oder in ähnlicher Form in vorherigen Semestern bereits als Klausuraufgaben gestellt. Sie dienen ihnen als Orientierung für die Klausur.

Aufgabe 1 : (5P) Ressourcenverwaltung (a+b)

a) Zeichnen Sie für das folgende Szenario den Resource-Allocation Graph. Fügen Sie diesen als jpg ihrer Lösung bei
   - In einem System laufen 4 Prozesse (P1, P2, P3 und P4) und es gibt 4 Ressourcen (R1, R2, R3 und R4). Von den Ressourcen gibt es jeweils nur eine Instanz.
   - Zu einem Zeitpunkt T1 benutzt Prozess P1 die Instanz der Ressource R3; Prozess P2 benutzt die Instanz der Ressource R2; Prozess P3 benutzt die Instanz der Ressource R1; und Prozess P4 benutzt die Instanz der Ressource R4.
   - Gleichzeitig fordern sowohl Prozess P1 als auch Prozess P4 die Instanz der Ressource R1 an; Prozess P2 fordert die Instanzen der Ressourcen R3 und R4 an; und Prozess P3 fordert die Instanz der Ressource R2 an.



b) Stellen Sie anhand Ihres Resource-Allocation Graphs fest, ob ein Deadlock vorliegt. Begründen Sie Ihre Antwort!

Aufgabe 2 : (8P) Synchronisation und Deadlocks (a+b+c)

Beim bekannten "Philosophen-Problem" sitzen fünf Philosophen um einen Tisch mit einer Schüssel Nudeln. Zwischen den Philosophen liegt je eine Gabel. Die Philosophen denken angestrengt nach – und bekommen vom angestrengten Denken regelmäßig Hunger und möchten essen. Dazu benötigen sie zwei Gabeln, nämlich die zwei Gabeln, die links und rechts vor ihnen auf dem Tisch liegen. Hat ihr linker oder rechter Nachbar die Gabel jedoch bereits aufgenommen, müssen sie warten, bis ihr Nachbar die Gabel wieder zurück auf den Tisch gelegt hat, bevor sie die Gabel ergreifen und essen können.

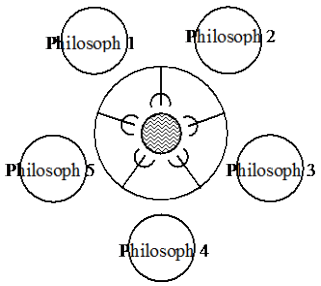
Der folgende Java-Pseudo-Code zeigt eine (unsynchronisierte) Implementation des Philosophen-Problems. Gezeigt sind (Ausschnitte der) Klassen Philosoph und Gabel. Es sei angenommen, dass fünf Gabeln instanziiert werden und fünf Philosophen-Threads, wobei die Attribute "linkeGabel" und "rechteGabel" der Philosophen-Threads entsprechend des oben beschriebenen Szenarios initialisiert werden.

```
1  class Philosoph extends Thread {
2    private boolean fuerImmer = true;
3    private Gabel linkeGabel, rechteGabel;

4    [...]
5    public void run() {
6      while ( fuerImmer ) {
7        denke();
8        nimmGabelnAuf();
9        iss();
10       legeGabelnZurueck();
11     }
12   }

13  public void nimmGabelnAuf() {
14    rechteGabel.nimmAuf();
15    linkeGabel.nimmAuf();
16  }

17  public void legeGabelnZurueck() {
18    linkeGabel.legeZurueck();
19    rechteGabel.legeZurueck();
20  }
21 }
```




```
22  class Gabel {
23    private boolean verfuegbar = true;

24    public void nimmAuf() {
25      while ( !verfuegbar ) {     //warte, bis
26        Thread.yield();           //Gabel verfügbar
27      }
28      verfuegbar = false;         //nimm Gabel auf
29    }

30    public void legeZurueck() {
31      verfuegbar = true;          //lege Gabel zurück
32    }
33  }
```

a) Erläutern Sie **in einem Satz**, zu welchem Problem der auf dieser Seite gezeigte (unsynchronisierte) Programmcode führen kann, **und unterstreichen Sie** im obigen Programm die gemeinsam genutzte Variable, deren (unsynchronisierter) Zugriff zu dem benannten Problem führt.

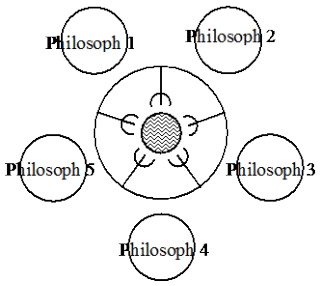
Zur Beseitigung des vorgenannten Problems werden die Methoden "nimmAuf" und "legeZurueck" der Klasse "Gabel" mit einem Monitor synchronisiert. Nach dieser Änderung sieht der Java-Pseudo-Code wie folgt aus:

```
1  class Philosoph extends Thread {
2    private boolean fuerImmer = true;
3    private Gabel linkeGabel, rechteGabel;

4    [...]
5    public void run() {
6      while ( fuerImmer ) {
7        denke();
8        nimmGabelnAuf();
9        iss();
10       legeGabelnZurueck();
11     }
12   }

13  public void nimmGabelnAuf() {
14    rechteGabel.nimmAuf();
15    linkeGabel.nimmAuf();
16  }

17  public void legeGabelnZurueck() {
18    linkeGabel.legeZurueck();
19    rechteGabel.legeZurueck();
20  }
21 }
```




```
22  class Gabel {
23    private boolean verfuegbar = true;

24    synchronized public void nimmAuf() {
25      while ( !verfuegbar ) {      //warte, bis
26        Thread.yield();          //Gabel verfügbar
27      }
28      verfuegbar = false;      //nimm Gabel auf
29    }

30    synchronized public void legeZurueck() {
31      verfuegbar = true;       //lege Gabel zurück
32    }
33  }
```

b) Welches Problem haftet dem auf dieser Seite gezeigten Programmcode nun an, und warum? Begründen Sie!

c) Wie könnte das verbleibende Problem gelöst werden? Erläutern Sie kurz und ergänzen Sie den obenstehenden Programmcode dementsprechend. Geben Sie bei ihrer Lösung zusätzlich die Zeilennummern an, zwischen denen weiterer Code hinzugefügt werden muss.

Aufgabe 3: (5P) Dateisysteme (a+b)

Ein Unix-Dateisystem belegt vollständig eine **64** GB große Festplatte und verwaltet diese mit inodes. Ein inode speichert die Adressen von **12** Datenblöcken und dazu einen Verweis auf einen indirekten sowie einen Verweis auf einen zweifach indirekten Verwaltungsblock. Die Blockgröße beträgt **4** KB (4096 Bytes). Die Blöcke der Platte werden mit **64** Bit großen Adressen angesprochen.

Tipp: Stellen Sie die oben genannten relevanten Zahlen mit Hilfe von 2er-Potenzen dar und rechnen Sie bei der Lösung der Aufgabenteile a) und b) mit diesen!

a) Berechnen Sie, wie viele Adressen ein Verwaltungsblock aufnehmen kann.

b) Berechnen Sie die maximale Größe einer Datei, die mit dem oben genannten Dateisystem verwaltet werden kann? Machen Sie dabei deutlich, wie Sie das Ergebnis berechnet haben!

Kleine Hilfstabelle

2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536
2^{17}	131072
2^{18}	262144
2^{19}	524288
2^{20}	1048576

Aufgabe 4: (7P) mytee

Das Kommandozeilentool `tee` liest Daten von der Standardeingabe, gibt sie auf der Standardausgabe aus und kopiert die Daten gleichzeitig in eine Datei. Das Ziel dieser Aufgabe ist es, ein Tool namens `mytee` in C zu schreiben, welches zusätzliche Informationen in einer log-Datei ablegt.

a) Implementieren Sie in C ein Tool `mytee`, das Daten von der Standardeingabe liest und sie auf der Standardausgabe ausgibt. Gleichzeitig sollen in einer Logdatei (*mytee.log* im selben Verzeichnis) zusätzliche Informationen gespeichert werden. Stehen in dieser Datei schon log-Einträge, soll der neue Eintrag hinter dem letzten Eintrag angehangen werden.
   - Jeder Logeintrag soll das Datum (*Tag.Monat.Jahr*) und Uhrzeit(*Stunde:Minute:Sekunde*) des Aufrufs, sowie den Namen der Ausgabe-Datei Speichen, jeweils getrennt durch ein Leerzeichen (Beispiel für einen Log-Eintrag: „24.01.2014 11:05:20 myFile.txt“

WICHTIGE INFORMATIONEN

Wenn Sie Programmcode schreiben, packen Sie zusätzlich zu Ihren Antworten auch alle Quell-Dateien in eine .zip Datei. Nutzen Sie bitte bei den Antwortdokumenten folgendes Format:

Nachname_Vorname_UebungXX.zip

*Laden Sie Ihr Antwortdokument bis spätestens **30.01.2015, 11:59** in Moodle hoch.*