LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 1
16.10.2001

Informatik III

Lesen: Silberschatz, Kap. 1

Vertiefen: Stallings, Kap. 2; Tanenbaum, Kap. 1

Literatur:
A. Silberschatz, P. B. Galvin: *Operating System Concepts*, Reading MA, 1998, Addison-Wesley.
W. Stallings: *Operating Systems*, Upper Saddle River NJ, 1995, Prentice-Hall.
A. S. Tanenbaum: *Modern Operating Systems*, Englewood Cliffs NJ, 1992, Prentice-Hall.

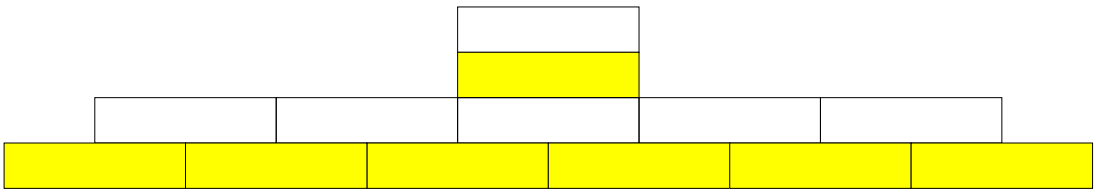
Aufgabe 1: (H) Betriebssystem

(4+1+4+1+3 Pkt.)

Das zentrale Thema der Vorlesung Informatik III ist das Bindeglied zwischen Rechnerarchitektur und Anwendungsprogrammen, das **Betriebssystem**.

a. Tragen Sie die folgenden Begriffe in die Pyrmide ein:
Anwendungsprogramme, Assembler, Bibliothek, Compiler, Dateisystem, Debugger, Ein- und Ausgaberoutinen, Gerätemanagement, Lader, Nutzer, Scheduler, Speicherverwaltung, Texteditor.

Dabei sollen die Objekte einer „Ebene“ auf die Objekte der darunterliegenden Ebene unmittelbar zugreifen, d.h. wenn beispielsweise eine *Seketärin* eine *Schreibmaschine* und ein *Telefon* nutzt, un diese wiederum mit elektrischem *Strom* betrieben werden, dann sollte die Sekretärin ganz oben in der Hierarchie stehen, darunter die Schreibmaschine und das Telefon un in der dritten Ebene der Strom. Begründen sie ihre Entscheidung kurz!



b. Ordnen Sie die Begriffe *Nutzer*, *Betriebssystem*, *Systemprogramme*, *Anwendungsprogramm* und *Hardware* in das Ebenenbild der Pyramide ein.

c. Welchen Ebenen würden Sie die folgenden Begriffe zuordnen:
CPU (Central Processing Unit), Speicher, Ein- und Ausgabegeräte, Compiler, Datenbanksysteme, Graphikprogramme, Dateisystem, Scheduler.

d. Als Computerkomponenten können dessen Hardware und Software angesehen werden. Welche Aufgabe kommt in dieser Einteilung einem Betriebssystem zu?

e. Überlegen Sie sich, welche drei zentralen Aufgaben ein Betriebssystem erfüllen soll.

Aufgabe 2: (T) Betriebssystemtypen (5+1+2+1 Pkt.)

a. Definieren Sie die essenziellen Eigenschaften der folgenden Typen von Betriebssystemen: *Batch, Interaktiv, Time Sharing, Echtzeit, Verteilt.*

b. Was ist der Hauptvorteil von **Multiprogramming**?

c. Was sind die wesentlichen Vor- bzw. Nachteile eines **Multiprozessor**-Systems?

Aufgabe 3: (P) Einfache UNIX-Kommandos (4+3+2+4 Pkt.)

In dieser Aufgabe sollen Sie sich mit dem Betriebssystem UNIX bekannt machen. Zu einem Befehl `cmd` erhalten Sie durch Eingabe von `man cmd` oder `info cmd` eine Erläuterung.

a. Beschreiben Sie kurz Sinn und Zweck der folgenden Kommandos inklusive der angegbenen Flags. Flags sind durch vorangestelltes „-“ gekennzeichnet. Sie können beim `man`-Aufruf weggelassen werden. Schreiben Sie pro Kommando höchstens 5 Zeilen!

   - `ls -l`
   - `mkdir`
   - `rm -i`
   - `more -c`
   - `cp`
   - `wc`
   - `cat`

b. Finden Sie eine Kategorisierung für die folgenden Befehle mit genau drei Kategorien:

   `rm`, `kill`, `mv`, `grep`, `killall`, `mkdir`, `cp`, `cd`, `ls`, `cat`, `more`.

c. Es gibt noch eine weitere wichtige Klasse von Kommandos: Builtin-Kommandos der Shell. Die Shell ist das Programm zur Interpretation Ihrer Befehlseingaben beispielsweise in einem Terminal. Es gibt unterschiedliche Shells mit unterschiedlichen Builtin-Kommandos. Am CIP-Pool wird standardmäßig die `TCSH` eingesetzt. Finden Sie heraus wie bei der `TCSH` Variablen definiert und verändert werden können! Unterscheiden Sie dabei zwischen lokalen und systemweiten Variablen! Ermitteln Sie auch, wie diese Befehle bei der `BASH` lauten!

d. Die Shell liest Kommandos von der Standardeingabe (sofern nicht anders vereinbart: Tastatur) und gib t die Ergebnisse auf der Standardausgabe (i.a. Bildschirm) aus. Durch den *pipe-Operator* „`|`“ wird die Ausgabe eines Kommandos als Eingabe eines weiteren Kommandos benutzt. So wird bei Eingabe von `cmd1 | cmd2` die Ausgabe von `cmd1` als Eingabe für `cmd2` interpretiert. Die Ausgabe erfolgt erst nach Abarbeitung von `cmd2` auf der Standardausgabe. Mehrere pipe-Operatoren in einer Kommandozeile sind möglich!

   So gibt `ls -l | grep "st"` alle Dateien des aktuellen Verzeichnisses aus, deren Name den Teilstring „st“ enthalten.

   Schreiben Sie eine (einzige) Kommandozeile, die alle Benutzer des CIP-Pools, in deren Namen „Frank“ vorkommt, ausgibt! Benutzerdaten wie Benutzername, echter Name, Passwort etc. sind unter UNIX in der Datei `/etc/passwd` abgespeichert. Die Ausgabe soll auf Vor- und Nachnamen beschränkt und alphabetisch sortiert sein! Verwenden Sie nur die Kommandos `cat`, `grep` und `sort`!

   Geben Sie auch die Ausgabe ihres Kommandos an!

Aufgabe 4: (P) Makefiles (6 Pkt.)

Machen Sie sich mit der Aufgabe des Programmes `make` und dem Aufbau eines Makefiles bekannt (z.B. durch `info make` oder im (X)Emacs).
Schreiben Sie ein Makefile, das die beiden folgenden Aufgaben erfüllt:

a. Eine Java-Datei (`sample.java`) soll in Bytecode übersetzt werden (`sample.class`). Dies soll durch Eingabe von `make` im entsprechenden Verzeichnis erfolgen.

b. Durch Eingabe von `make clean` sollen alle regenerierbaren Dateien gelöscht werden.

Beachten Sie, daß die Kommandos unter allen Umständen funktionieren (z.B. auch wenn eine Datei `clean` im entsprechenden Verzeichnis existiert).

Aufgabe 5: (P) Einfache Shellprogrammierung (6 Pkt.)

Gegeben ein Verzeichnis mit 20 Dateien, denen allen das Kürzel `A1_` vorangestellt werden soll. Überlegen Sie sich, wie Sie dieses Problem mit den Mitteln einer Shell und den UNIX-Grundkommandos effizient lösen können. Suchen Sie dazu in der Manpage Ihrer Shell ein Kommando, mit dem man Befehle über alle Elemente einer Auflistung (wie eines Verzeichnisses) iterieren kann! Tip: `for` oder `foreach`.

Aufgabe 6: (P) Java (3 Pkt.)

Kompilieren Sie das folgende Programm und führen Sie es auf verschiedenen, Ihnen zugänglichen Systemen aus. Protokollieren und kommentieren Sie das Ergebnis!

```
//
// simple example of threads in Java, showing whether your system
//      performs timeslicing:  if you see "ping" and "pong" messages
//      interleaved, your system does timeslicing; if not, it doesn't.
//
// see main method below for command-line arguments.
//

//
// a SimplerThread object repeatedly prints message msg.
//
public class SimplerThread extends Thread {

        String msg ;
        int cycles ;

        SimplerThread(String m, int c) {
                msg = m ;
                cycles = c ;
        }

        // overrides run() in Thread class to define object's
        //      behavior.
        public void run() {
                for (int i = 0 ; i < cycles ; i++) {
                        System.out.println(msg) ;
                }
        }

        // command-line arguments are integers C.
        // builds and starts two threads of type SimplerThread.
```

```
32          // continues for C cycles.

34          public static void main(String[] args) {

36                  if (args.length < 1) {
                            System.out.println("Arguments␣are:") ;
38                          System.out.println("␣␣cycles") ;
                            System.exit(-1) ;
40                  }

42                  int C = Integer.parseInt(args[0]) ;

44                  SimplerThread t1 =
                            new SimplerThread("ping---->", C) ;
46                  SimplerThread t2 =
                            new SimplerThread("<----pong", C) ;
48
                    t1.start() ;
50                  t2.start() ;
            }
52  }
```

Aufgabe 7: (P) Environment (6 Pkt.)

Mit Hilfe der eingebauten Shell-Kommandos läßt sich die Umgebung, in der die folgenden Befehle ausgeführt werden, modifiziert werden. Durch Eingabe von `set` ohne Parameter wird die aktuelle Umgebung der Shell ausgegeben. Die Umgebung wird im wesentlichen durch die *Umgebungsvariablen* definiert.
Sie sollen nun den Prompt (die Eingabeaufforderung der Shell) so verändern, daß er etwa das folgende Aussehen hat:

```
user@host [Datum Zeit] Aktuelles_Arbeitsverzeichnis >
caesar@fichte [14.10.00 18:05] /usr/share/doc >
```

Dabei soll die Ausgabe des aktuellen Arbeitsverzeichnis auf vier Pfadbestandteile begrenzt sein, weitere Pfadbestandteile sollen vom Wurzelverzeichnis aus abgeschnitten werden.
Tip: Der Name der Umgebungsvariable, die das Aussehen des Prompts bestimmt, ist je nach Shell unterschiedlich, schauen Sie dazu in der Manpage Ihrer Shell nach (z.B. unter `prompt` oder `PS1`).

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 13
22.01.2002

Informatik III

Achtung: Die **Anmeldung** zur **2. Klausur** ist von **Montag, 28.01.2002 10 Uhr** bis **Freitag, 01.02.2002 12 Uhr** möglich.
Die Anmeldung ist **obligatorisch** und Nachmeldungen werden **nicht akzeptiert!**
Bitte beachten Sie, dass die **Voraussetzung** für die **Zulassung zur 2. Klausur** das **erfolgreiche Vorrechnen** einer Übungsaufgabe ist! Die **Deadline** zum Erreichen dieser Zulassungsbedingung ist der **31.01.2002**. Aktuelle Listen über den gegenwärtigen Stand können sowohl auf der Info3-Homepage eingesehen werden als auch beim jeweiligen Tutor nachgefragt werden.

Lesen: Silberschatz Kap. 9

Aufgabe 67: (H) Speicherverwaltungsstrategien (4 Pkt.)

Für die Bewertung und den Vergleich von Speicherverwaltungsstrategien zum Belegen und Freigeben von zusammenhängenden Speicherbereichen seien die folgenden Eigenschaften von Interesse:

a. die Ausnutzung des Speichers,

b. die Art der Zerstückelung des Speichers, die damit verbundene Anzahl der Freibereiche und der Suchaufwand, sowie

c. der Aufwand zur Erstellung und Führung der Verwaltungsstrukturen.

Vergleichen Sie im Hinblick auf diese Eigenschaften die Strategien First Fit, Rotating First Fit, Best Fit und Worst Fit!

Aufgabe 68: (H) Virtueller Speicher (6+2 Pkt.)

Das Konzept des virtuellen Speichers liefert die Möglichkeit, von den realen Speichern in einem Rechensystem zu abstrahieren. Der virtuelle Speicher besteht aus Segmenten, die wiederum in Seiten zerfallen. Die einzelnen Seiten sind in Zellen unterteilt, die zur Speicherung von (32-bit) Wörtern dienen.
Für die Realisierung des virtuellen Speichers wird der Arbeitsspeicher in Seitenrahmen (frames) unterteilt. Die Größe eines Seitenrahmens entspricht idealerweise der Größe einer Seite des virtuellen Speichers, um eine 1:1-Abbildung von Seite zu Seitenrahmen zu ermöglichen.
Für die Verwaltung des virtuellen Speichers wird eine Segmenttabelle eingeführt, die alle allokierten Segmente des virtuellen Speichers erfaßt. Ein Segment wird eindeutig durch einen Segment-Deskriptor beschrieben. Eine mögliche Realisierung des Segment-Deskriptors und der Segmenttabelle ist:

```
TYPE segment_descriptor IS RECORD
   segment_exists    : BOOLEAN;   -- true = Segment existiert
   length            : INTEGER;   -- Segmentlaenge
   read_access       : BOOLEAN;   -- lesender Zugriff erlaubt
   write_access      : BOOLEAN;   -- schreibender Zugriff erlaubt
   execute_access    : BOOLEAN;   -- ausfuehrender Zugriff erlaubt
   pagetable_pointer : INTEGER;   -- Adresse der zugehoerigen
   -- Seitentabelle oder NULL
END RECORD;

TYPE segment_table IS ARRAY[1..MaxSeg] OF segment_descriptor;
```

Desweiteren wird für jedes Segment eine Seitentabelle benötigt, die die einzelnen Seiten jedes Segmentes und deren Lage im Arbeitsspeicher erfaßt. Eine Seite wird eindeutig durch einen Seiten-Deskriptor beschrieben. Eine mögliche Realisierung des Seiten-Deskriptors und der Seitentabelle ist:

```
TYPE page_descriptor IS RECORD
   page_in_core  : BOOLEAN;   -- true = Seite im Arbeitsspeicher
   frame_pointer : INTEGER;   -- Seitenrahmenadresse des AS oder NULL
   block_pointer : INTEGER;   -- Blockadresse des HS oder NULL
END RECORD;

TYPE page_table IS ARRAY[1..MaxPage] OF page_descriptor;
```

Eine virtuelle Adresse sei gegeben durch ein Paar (*page_name, offset*). Der aktuelle Ausführungskontext sei gegeben durch das Paar (*process_name, segment_name*).

a. Beschreiben Sie in programmiersprachlicher Notation die Durchführung eines Speicherzugriffs [*process_name,segment_name), (page_name,offset), Art*], wobei *Art* $\in$ {read, write, execute} und die übrigen Komponenten vom Typ Integer sind. Führen Sie insbesondere die notwendigen Prüfungen auf Fehler durch! Eine geeignete Funktion zum Auflösen eines Seitenfehlers sei gegeben.

b. Geben Sie die Anzahl der Arbeitsspeicherzugriffe an, die für einen Speicherzugriff – wie oben angegeben – notwendig sind. Sie dürfen davon ausgehen, daß die Segmenttabelle, die Seitentabellen und die Seite, die das spezifizierte Wort enthält, bereits im Arbeitsspeicher vorliegen.

Aufgabe 69: (K) Seitenersetzungsstrategien (10+2 Pkt.)

Sei die Menge der Seiten gegeben durch $N = \{0,1,2,3,4,5,6\}$ und die Menge der im Arbeitsspeicher zur Verfügung stehenden Seitenrahmen durch $Frame_4 = \{f_1, f_2, f_3, f_4\}$.
Auf die sieben Seiten wird in der folgenden Reihenfolge zugegriffen:

$$w = 6\,3\,4\,5\,2\,1\,5\,0\,6\,5\,0\,3\,4\,0$$

a. Geben Sie für die Seitenersetzungsstrategien *First In First Out (FIFO)* und *Least Recently Used (LRU)* die Seitenrahmenbelegung bei der Referenzierfolge w an sowie die Gesammtsumme der Seitenfehler. Verwenden Sie hierfür eine Tabelle, die das folgende Aussehen hat:

Referenzierte Seiten	f_1	f_2	f_3	f_4	Summe d. Seitenfehler
⋮	⋮	⋮	⋮	⋮	⋮

Beginnen Sie für jede referenzierte Seite eine neue Zeile und protokollieren Sie bei jedem Referenzieren die Summe der Seitenfehler mit!

b. Welche allgemeine Beobachtung führt dazu, dass das LRU–Verfahren im Durchschnitt nahezu optimal ist?

Aufgabe 70: (P) Publisher/Subscriber (8 Pkt.)

Die AWT/Swing-Listener (die Sie kennengelernt haben sollten, als Sie Java gelernt haben) sind ein Beispiel eines allgemeinen Kommunikationsmusters, häufig als „Publisher/Subscriber“ beschrieben: Objekte, die über gewisse Ereignisse informiert werden wollen, „abonnieren“ eine Veröffentlichung eines Publishers (durch Aufruf der Methode `subscribe`). Der Publisher teilt (mittels der Methode `publish`) dem Subscriber mit, daß ein Ereignis eingetreten ist.
Implementieren Sie einen Publisher, der zumindest die Methoden `subscribe` und `publish` unterstützt. Testen Sie ihre Implementation!

Aufgabe 71: (P) Kommunikations-Protokoll (4+1 Pkt.)

Betrachten Sie das Java-Programm *MailTest.java*, das Sie von der Info3-Homepage in der Sektion *Übungsblätter* herunterladen können.
Dies ist ein Beispielprogramm für einen (extrem einfachen) SMTP-Client. Testen Sie das Programm und beobachten Sie die Kommunikation zwischen dem Mail-Client und dem SMTP-Server.
Hinweis: Sie können zum Testen den Mailserver des CIP-Pools (`mail.cip`) verwenden. Die Mails werden tatsächlich gesendet! Achten Sie also darauf, wen Sie als Empfänger angeben.

a. Beschreiben Sie den Ablauf einer SMTP-Kommunikation zwischen Client und Server zum Versenden einer Mail.

b. Was fällt Ihnen an der Kommunikation auf?

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 12
15.01.2002

Informatik III

Achtung: Am nächsten Donnerstag (17.01.02) findet anstelle der Vorlesung die Klausureinsicht statt.

Lesen: Stallings Kap. 7 und Kap. 8, Silberschatz Kap. 8 und Kap. 9

Aufgabe 63: (H) Seitenersetzungsstrategien (8+8 Pkt.)

Bei der Ausführung eines Speicherzugriffs bei der virtuellen Speicherverwaltung kann es vorkommen, daß sich die referenzierte Seite nicht im Arbeitsspeicher befindet. Diese Situation wird Seitenfehler (*page fault*) genannt.
Die Behandlung eines Seitenfehlers erfordert i.a. Maßnahmen zur Ersetzung einer Seite im Arbeitsspeicher, d.h. um die gewünschte Seite in einen Seitenrahmen des Arbeitsspeichers einlagern zu können, muß zunächst eine andere Seite vom Arbeitsspeicher auf den Hintergrundspeicher ausgelagert werden.
Die Menge der Seiten sei gegeben durch $N = \{0,1,2,3,4,5\}$ und die Menge der Seitenrahmen, die für die Speicherung der Seiten im Arbeitsspeicher zur Verfügung steht, sei gegeben durch $\mathit{Frame}_4 = \{f_1, f_2, f_3, f_4\}$.
Auf die 6 Seiten der Menge N werde in folgender Reihenfolge zugegriffen:

$$w = 1\ 3\ 5\ 4\ 2\ 4\ 3\ 2\ 1\ 0\ 5\ 3\ 5\ 0\ 4\ 3\ 5\ 4\ 3\ 2\ 1\ 3\ 4\ 5$$

a. Arbeiten Sie mit den zur Verfügung stehenden Seitenrahmen der Menge Frame_4 die Seitenzugriffsfolge w gemäß den Ersetzungsstrategien

   - FIFO : First in First out
   - LIFO : Last in First out,
   - LRU : Least Recently used,
   - LFU : Least Frequently used

   ab, und vergleichen Sie diese anhand der Seitenfehleranzahlen!

b. Die Menge der Seitenrahmen werde vergrößert zu $\mathit{Frame}_5 = \{f_1, f_2, f_3, f_4, f_5\}$. Arbeiten Sie w analog, und vergleichen Sie die Seitenfehleranzahlen!

Aufgabe 64: (T) Buddy-Verfahren (11+5 Pkt.)

Das von Donald E. Knuth entwickelte Buddy-Verfahren zur Verwaltung von Speicher nutzt die Binäradressierung aus, um effizient benachbarte Freibereiche zu einem grösseren Freibereich zusammenzufassen. Zur Erinnerung sei das Verfahren im folgenden nocheinmal kurz beschrieben:
Die mit dem Buddy-Verfahren verwalteten, zusammenhängenden Speicherbereiche können nur die Länge 2^k, mit $k = 0,1,2,\ldots$, haben. Für einen Speicherbereich der Länge *length* wird ein Speicherbereich der Größe 2^r, mit $r = \lceil \mathrm{ld}\ \mathit{length} \rceil$, belegt. Der Gesamtspeicher habe eine Größe

von 2^M. Zur Verwaltung der Freibereiche existiert für jede mögliche Länge 2^k, mit $k = 0, 1, \ldots, M$ eine Liste. Für eine Belegung eines Bereichs der Länge *length* wird in der entsprechenden Liste nach einem freien Bereich gesucht. Ist bei der Belegung kein Bereich passender Länge frei, so wird ein größerer Bereich genommen und dieser wird halbiert, wobei eine Hälfte in die zugehörige Frei-Liste aufgenommen wird, während der andere Teil entweder weiter halbiert oder entsprechend der gestellten Anforderung reserviert wird. Bei der Halbierung entstehen zwei **Buddies**. Diese und nur diese können zu einem größeren Freibereich wieder zusammengefaßt werden, falls sie beide wieder frei sind. Zu Beginn existiert nur ein Freibereich der Länge 2^M.

a. Geben Sie in programmiersprachlicher Notation die *Belege* und *Freigabe* Operationen an, die einen 1Mb großen Speicher gemäß dem oben beschriebenen Buddy-Verfahren verwalten!

Aufgabe 65: (K) Indirekte Kommunikation (4 Pkt.)

Implementieren Sie eine einfache asynchrone Mailbox zur indirekten Kommunikation zwischen Prozessen. Eine *Mailbox* dient zur indirekten Kommunikation: Der Sender sendet Nachrichten an die Mailbox, die durch den Empfänger abgefragt wird. Sie unterstützt also nur die Methoden `send` und `receive`.

Aufgabe 66: (P) Java Pipes (1+6+4 Pkt.)

Sei ein einfaches Erzeuger/Verbraucher-Kommunikationsszenario gegeben. Wenn keine weiteren Daten verfügbar sind, soll der Verbraucher blockieren. Umgekehrt soll der Erzeuger blockieren, wenn die Daten deutlich schneller erzeugt als verbraucht werden.
Für dieses Szenario stellt Java API-Klassen zur Verfügung, die eine Pipe implementieren: `PipedInputStream` und *PipedOutputStream* für Bytes bzw. `PipedReader` und `PipedWriter` für Unicode Zeichen.

a. Nennen Sie den wesentlichen Vorteil von *Pipes*.

b. Schreiben Sie ein Java-Programm, in dem der Erzeuger zufällig Zufallszahlen erzeugt und in die Pipe schreibt. Der Verbraucher soll diese Zahlen ausgeben.

c. Erweiteren Sie das gerade geschriebene Programm durch einen *Filter*, der die durch den Erzeuger gelieferten Nummern erhält und dem Verbraucher den jeweils aktuellen Durchschnitt aller bereits erzeugten Nummern liefert.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 11
08.01.2002

Informatik III

Lesen: Silberschatz Kap. 6.7

Aufgabe 59: (H) Gotchi-Nanni (10 Pkt.)

Sie haben bereits den Gotchi-Pfleger kennengelernt. Dieser hatte jedoch einige Einschränkungen:

- Er kann nur zwei Gotchis auf einmal betreuen.
- Er neigt dazu, ein Gotchi dem anderen vorzuziehen.

Aus diesem Grund wird jetzt die Gotchi-Nani entwickelt, die mit diesen beiden Problemen aufräumen soll.
Die Nanni wird als Monitor konzipiert, so daß die Gotchi selbst nach Pflege nachfragen müssen. Der Monitor ist dem der Reader-Writer-Variante aus der Vorlesung ähnlich und dürfte deswegen leicht zu implementieren sein.
Dieser Monitor hat drei Aufgaben zu erfüllen:

- mit dem Gotchi spielen,
- die Gotchi füttern und
- die Gotchi säubern.

Füttern hat höchste Priorität, lieber haben wir frustrierte als tote Gotchi. Füttern bzw. säubern kann gleichzeitig erfolgen, d.h. ist der Monitor gerade am Füttern, kommt es auf einen hungrigen Gotchi mehr auch nicht mehr an. Spielen dagegen kann der Monitor nur mit einem Gotchi zu einer Zeit, ein Gotchi erfordert eben die ganze Aufmerksamkeit.
Nanni-Monitor:

```
2  MONITOR nanni;
   VAR hungercount, cleancount: INTEGER;          (* Counter nötig, da mehrere möglich *)
4  VAR busyclean, busyplay, busyhunger :  BOOLEAN; (* Nur ein Bedürfnis gleichzeitig *)
   VAR okhunger, okclean, okplay : CONDITION;
6
   PROCEDURE startfeed;
8  BEGIN
      (*  Sobald Nanni eine Aufgabe erfüllt hat, und ein Gotchi nach Futter schreit
10     *  soll Nanni ungeachtet anderer wartender 'dringender' Bedürfnisse
       *  mit dem Füttern beginnen.
12     *  Während des Fütterns werden andere, neu dazukommende Hungerleider gleich
       *  mitgefüttert. *)
14 END; { startfeed }
   PROCEDURE endfeed;
16 BEGIN
```

```
   (*  Sobald alle gefütter sind, sucht sich Nanni eine neue Aufgabe, vorzugsweise
18  *  das Säubernder jetzt satten Gotchis. *)
END; { endfeed }
20 PROCEDURE startclean;
BEGIN
22   (*  Nanni hat Zeit und keiner schreit nach Futter
     *  auch hier können Nachzügler mitversorgt werden. *)
24 END; { startclean }
PROCEDURE endclean;
26 BEGIN
   (*  Wenn Nanni alle Gotchi sauber hat, sieht sie sich
28  *  nach einer neuen Aufgabe um. *)
END; { endclean }
30 PROCEDURE startplay;
BEGIN
32   (*  Nur wenn alle Gotchi satt und sauber sind, findet Nanni Zeit
     *  etwas mit den Kleinen zu spielen. Aber immer nur mit einem
34   *  gleichzeitig :-) *)
END; { startplay }
36 PROCEDURE endplay;
BEGIN
38   (*  Spielen macht hungrig, wenn nicht ist bestimmt einer dabei
     *  dreckig geworden? *)
40 END; { endplay }
BEGIN (* Initialisierung der Variablen *)
42   hungercount := 0;        cleancount  := 0;
     busyhunger  := FALSE;    busyclean   := FALSE;
44   busyplay    := FALSE;
END;
```

HUNGER-Routine im Gotchi:

```
REPEAT
2    nanni.startfeed;          (* Gotchi schreit nach Futter *)
     Füttern;
4    nanni.endfeed;            (* Bin satt, mag kein Blatt *)
     unkritische Operation;    (* Verdauungsschlaf *)
6 UNTIL FALSE;
```

SCHMUTZ-Routine im Gotchi:

```
REPEAT
2    nanni.startclean;
     Säubern;
4    nanni.endclean;            (* Blitzblank *)
     unkritische Operation;
6 UNTIL FALSE;
```

LANGEWEILE-Routine im Gotchi:

```
REPEAT
2    nanni.startplay;           (* Ich will jetzt aber spielen *)
     Spielen;
4    nanni.endplay;
     unkritische Operation;
6 UNTIL FALSE;
```

Implementieren Sie die Routinen des Monitors!

Aufgabe 60: (T) Synchronisation über Nachrichtenaustausch (10 Pkt.)

Eine weitere Möglichkeit, die Synchronisation zwischen parallelen Abläufen zu erreichen, stellt der Nachrichtenaustausch dar. Zur Kommunikation zwischen Prozessen seien die folgenden Befehle vorhanden:

- **SEND**(Empfänger, Nachricht) Der Sender schickt eine Nachricht an den Empfänger und kann sofort weiterarbeiten.
- **RECEIVE**(Nachricht) Der Empfänger empfängt eine Nachricht. Ist keine an den Empfänger adressierte Nachricht vorhanden, so blockiert dieser bis zum Erhalt einer Nachricht.

Die verschickten Nachrichten setzen sich aus den Komponenten `nachricht.sender` und `nachricht.inhalt` (Typ jeweils String) zusammen.
Die Philosophen haben nach langem Denken und (Semaphor sei Dank) verklemmungsfreien Essen entschieden, daß sie die Semaphore verkaufen und einen magischen Tisch erfinden, der auf Zuruf (Nachrichtenaustausch) die Stäbchen an die Philosophen verteilt. Wie könnte ein solcher Tisch realisiert werden, der ebenfalls die Verklemmungsfreiheit durchsetzt?
Geben Sie einen Konstruktionsentwurf des Tisches in einer programmiersprachlichen Notation an.

Aufgabe 61: (K) Inner-class Synchronisation (5 Pkt.)

Betrachten Sie die folgende Klasse:

```
import javax.swing.*;
import java.awt.event.*;

class Status extends JFrame
{
    private String title    = "Status: idle";
    private String contents = "Nothing happening";

    public Status()
    {
        JButton pop_it_up = new JButton( "Show status" );
        pop_it_up.addActionListener
        (   new ActionListener()
            {   public void actionPerformed( ActionEvent e )
                {   JOptionPane.showMessageDialog( null,
                                contents, title, JOptionPane.INFORMATION_MESSAGE );
                }
            }
        );

        getContentPane().add( pop_it_up );
        pack();
        show();
    }

    public synchronized void change_status( String status, String explanation )
    {   this.title    = "Status: " + status;
        this.contents = explanation;
    }

    public static void main( String[] args )
    {
        Status myself = new Status();
```

```
34
        myself.change_status( "Busy", "I'm busy");
36      work();
        myself.change_status( "Done", "I'm done");
38      work();
        myself.change_status( "Busy", "I'm busy again");
40      work();
        myself.change_status( "Done", "I'm done again");
42      work();

44      System.exit( 0 );
    }
46
    private static void work()
48  {   try{ Thread.currentThread().sleep(4000); }catch( Exception e ){}
    }
50 }
```

Überlegen Sie sich, welches Synchronisationsproblem hier auftreten kann und beheben Sie es!

Aufgabe 62: (P) Synchronisation über Monitore (4+4+1 Pkt.)

Zur Wiederholung: Bei der Bildung abstrakter Datentypen werden die Daten und die Operationen darauf in einem Modul zusammengefaßt. Ein solches Modul heißt **Monitor**, wenn die Operationen von den Prozessen über Prozeduraufrufe zugänglich sind und die Prozeduren in einem Modul unter gegenseitigem Ausschluß ablaufen.
Für die Abläufe in einem Monitor gelten die folgenden Regeln:

a. Die lokalen Variablen und die lokalen Prozeduren sind nur innerhalb des Monitors zugänglich.

b. Die öffentlichen Prozeduren können von außen aufgerufen werden. Beim Aufruf wird der Name des Monitors vorangestellt. Dadurch entsteht eine eindeutige Bezeichnung der Prozedur. Öffentliche Prozeduren werden durch Voranstellen des Wortes `public` gekennzeichnet.

c. Der Monitor ist ein exklusives Betriebsmittel. Die Prozeduren in ihm laufen unter gegenseitigem Ausschluß ab. Prozesse, die Prozeduren des Monitors aufrufen wollen, müssen also gegebenenfalls warten bis der Monitor frei wird.

Darüber hinaus ist es oft notwendig, einen Ablauf im Monitor zu unterbrechen, bis eine bestimmte Bedingung erfüllt ist. Jeder Wartezustand wird dabei an eine sogenannte *condition variable* geknüpft. Folgende Operationen sind definiert:

- **`wait`**`(c: condition)`: Der Prozeß wird wartend gesetzt und verläßt den Monitor temporär.
- **`signal`**`(c: condition)`: Falls kein Prozeß bzgl. `c` wartet, ist der Aufruf ohne Wirkung. Falls mindestens ein Prozeß bzgl. `c` temporär wartet, wird der erste dieser Prozesse nach seinem Aufruf von **`wait`** fortgesetzt. Der aufrufende Prozeß wird zurückgestellt und wartet, bis der Monitor wieder frei wird.

a. Geben Sie eine Lösung des *Problems der speisenden Philosophen* für fünf Philosophen unter Verwendung von Java-Synchronisation an. Versuchen Sie, die Anforderungen an einen Monitor möglichst weitgehend umzusetzen.

b. Geben Sie eine Lösung des *Problems der speisenden Philosophen* für fünf Philosophen unter Verwendung von Java mit der Erweiterung um *condition variables* (wie oben) an. Sie sollen nicht die *condition variables* implementieren, der Typ `condition` sei vordefiniert.

c. Vergleichen Sie Ihre beiden Lösungen!

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 10
18.12.2001

Informatik III

Achtung: Um nach den Feiertagen wieder am gleichen Punkt aufsetzen zu können wie vor den Feiertagen, dient dieses Übungsblatt als Vertiefung in die Semaphorthematik und enthält nur 3 Aufgaben.

Das Info3–Team wünscht Ihnen ein fröhliches Weihnachtsfest und einen guten Rutsch ins neue Jahr!

Lesen: Tannenbaum Kap. 2.2.1–2.2.6
Vertiefen: Stallings Kap. 4

Aufgabe 56: (H) Semaphore (6+2+5+2 Pkt.)

In der Vorlesung haben Sie **Semaphore** als Kontrollkomponenten zur Realisierung des wechselseitigen Ausschluß kennengelernt.

a. Geben Sie informell die Funktionsweise eines Semaphors an und begründen Sie, warum die in der Vorlesung genannten vier Anforderungen an Lösungen des wechselseitigen Ausschlusses erfüllt sind!

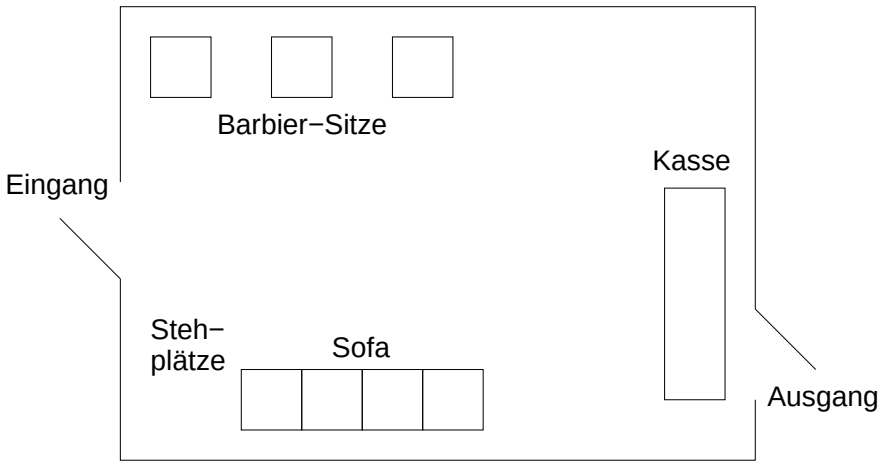
b. Welche Bedeutung hat der Anfangswert der Zählvariablen?

c. Zeigen Sie, daß die (allgemeinen) Semaphore und die binären Semaphore dieselbe Ausdruckskraft besitzen!

d. Begründen Sie (informell) die Korrektheit der angegebenen Lösung!

Aufgabe 57: (H) Barbierladen (6+2 Pkt.)

Ein weiteres Beispiel für die Nutzung von Semaphoren zur Synchronisation ist das **Problem des Barbierladens**.
Unser Barbierladen habe drei Sitze, drei Barbiere und einen Warteraum, in dem Platz für vier Kunden auf einem Sofa und Raum zum Stehen vorhanden ist. Feuerschutzvorschriften beschränken die Gesamtanzahl von Kunden im Laden auf 20.



Verhaltensregeln: Ein Kunde darf den Laden nicht betreten, wenn dieser mit der maximalen Zahl anderer Kunden gefüllt ist. Hat der Kunde einmal den Laden betreten, setzt er sich auf das Sofa, oder, falls das Sofa voll ist, bleibt er im Warteraum stehen. Wenn ein Barbier frei ist, wird derjenige Kunde bedient, der am längsten *auf dem Sofa* sitzt. Wenn es stehende Kunden gibt, wird der frei gewordene Platz auf dem Sofa von dem am längsten stehenden Kunden eingenommen. Wenn ein Kunde bedient wurde, kann bei jedem Barbier die Rechnung bezahlt werden, aber da es nur eine Kasse gibt, kann nur ein Kunde zu einer Zeit bezahlen. Die Barbiere verbringen ihre Zeit mit Haareschneiden, Kassieren und Schlafen (in ihrem Stuhl) wartend auf einen Kunden.

a. Finden Sie eine Lösung für dieses Problem. Synchronisation soll über Semaphore (ohne Warteschlangen!) gelöst werden. Geben Sie eine Pseudocode-Implementation für Kunden, Barbiere bzw. Kassierer an.

b. Stellen Sie sicher, daß Ihre Lösung alle Kunden fair behandelt.
Hinweis: Dazu müssen Sie eine Warteschlange für die Kunden bereitstellen und verwalten.

Aufgabe 58: (P) Java-Semaphor (10+5 Pkt.)

Betrachten Sie das folgende Interface für Semaphore in Java:

```
interface Semaphore
{
    /********************************************
     * A useful symbol constant for passing as a timeout value.
     * Effectively waits forever (actually waits for only
     * 292,271,023 years).
     */
    public static final long FOREVER = Long.MAX_VALUE;

    /********************************************
         * Acquire the semaphore. In the case of a <code>Mutex</code>, for example,
         * you would get ownership. In the case of a <code>Counting_semaphore</code>,
         * you would decrement the count.
         *
         * @return false if the timeout is zero and the lock was not
         *                                  acquired. Otherwise returns true.
         *
         * @throws InterruptedException if the wait to acquire the lock was
         *                       interrupted by another thread.
         *
         * @throws Semaphore.Timed_out if the time out interval expires before
         *                       the semaphore has been acquired.
         */
    public boolean acquire(long timeout) throws InterruptedException,
        Timed_out;

        /********************************************
         * Release the semaphore
         */

    public void release();

    /********************************************
         * Thrown in the event of an expired timeout.
         */

    public static final class Timed_out extends RuntimeException
```

```
38      {   public Timed_out(){ super(
                                    "Timed out while waiting to acquire semaphore"); }
40          public Timed_out(String s){ super(s); }
        }
42
        /*******************************************
44           * Thrown when a thread tries to release a semaphore illegally, e.g.
             * a Mutex that it has not acquired successfully.
46           */

48      public static final class Ownership extends RuntimeException
        {   public Ownership(){ super(
50                                  "Calling Thread doesn't own Semaphore"); }
        }
52  }
```

a. Schreiben Sie eine binäre Semaphor-Klasse `Mutex` in Java, die das angegebene Interface implementiert. Ihre Klasse soll die Möglichkeit bieten, `acquire` mit einem Timeout-Wert aufzurufen, der die maximale Wartezeit auf den Semaphor angibt.

   Hinweis: Sie werden für die Lösung dieses Problems die Methoden der Thread-Klasse `wait` und `notify` benötigen. Beachten Sie auch, daß `acquire` und `release` stets nur von einem Prozeß verwendet werden dürfen. Verwenden Sie dazu `synchronized` Methoden.

b. Erweitern Sie Ihre Lösung für Zählsemaphore.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 9
11.12.2001

Informatik III

Achtung:

Die 1. Klausur findet am Samstag, 15.12.2001, 16 Uhr s.t.
in den Räumen HS 101/201 (LMU Hauptgebäude) statt.
Unbedingt gültigen Studenten- und Personalausweis mitbringen!
Die Nachmeldung ist am Dienstag, 11.12.2001, von 10-17 Uhr möglich.

Lesen: Stallings Kap. 5.1 bis 5.3, Silberschatz Kap. 6.2, 6.3

Wiederholen: Skript Kapitel 6.5 (Petrinetze)

Aufgabe 51: (H) Galanen-Dilemma (10 Pkt.)

Ein Ehemann befindet sich in der prekären Lage, außer einer Geliebten G auch noch ein Verhältnis mit seiner Sektretärin S zu haben. Ersteres ahnt seine Ehefrau E, letzteres die Geliebte. Aus seiner Sicht heraus besteht nun die Gefahr, daß die süßen Geheimnisse gelüftet werden, wenn die betroffenen Frauen Gelegenheit finden, sich auszusprechen. Geschähe das, so befürchtet er, die Kosten der Scheidung und die Kündigung der Sekretärin ohne den Trost der Geliebten überstehen zu müssen. Er muß also vermeiden, daß Frau und Geliebte bzw. Geliebte und Sekretärin unbeaufsichtigt zusammentreffen. Auf einer Party ergibt es sich nun, daß sich unser Mann plötzlich (ein Freund hilft natürlich nach) von seinen Damen umringt wiederfindet. Nach einiger Zeit beschließt die Gesellschaft in die Bar hinauf zu fahren. Der Aufzug des Hauses reicht jeweils nur für zwei Personen. Zwei Damen fahren, da schon allerhand passiert ist im Hause, nie zusammen – das ist eine Regel, es muß stets ein Mann mit im Aufzug sein. Wie aber soll unser Gebeutelter nun die Reihenfolge manipulieren, um den nachteiligen Konsequenzen zu entkommen? Versuchen Sie die Lösung in Form eines markierten Petri-Netzes zu formulieren!

Aufgabe 52: (T) Krach bei Ferrari (1+5+3 Pkt.)

Michael Schumacher ist sauer: Er würde furchtbar gerne auf dem Nürburgring trainieren, aber die ganze Zeit basteln die Mechaniker noch an seinem Wagen herum. Es ist klar, daß die Mechaniker nur arbeiten können, solange Schumacher nicht fährt. Andererseits kann Schumacher nur dann losfahren, wenn gerade einmal kein Mechaniker mehr arbeitet.
Damit nun sowohl das Training Schumachers als auch die Wartungsarbeiten an seinem Wagen nicht zu kurz kommen, legt das Ferrari-Team folgende Regelung fest: Sobald Michael Schumacher eine Runde fahren will, darf kein Mechaniker mehr mit neuen Reparaturen anfangen; die bereits arbeitenden Mechaniker dürfen ihre Reparaturen aber noch beenden. Nach jeder Runde fährt Schumacher dann in die Box, um nachzusehen, ob dort Mechaniker darauf warten, etwas zu reparieren. Ist dies der Fall, so steigt er aus, und die wartenden Mechaniker dürfen mit ihren Arbeiten beginnen.

a. Unter welchem Namen ist dieses Problem üblicherweise in der Literatur bekannt?

b. Synchronisieren Sie Mechaniker und Fahrer! Verwenden Sie hierzu einen einfachen Semaphor namens `Ablösung` und einen Semaphor mit assoziierter Warteschlange namens `der-Nächste-bitte`, sowie eine (durch einen weiteren Semaphor namens `Sicherung` geschützte) Zählveriable n für die Anzahl der momentan arbeitenden Mechaniker. Geben Sie die entsprechenden Algorithmen für Fahrer und Mechaniker an!

Das funktioniert einige Zeit ziemlich gut, bis im Rennen von Jerez „plötzlich dieses blaue Auto neben ihm war“. Damit so etwas nicht mehr vorkommt, beschließt Ferrari die Einstellung eines zusätzlichen Spezialisten, der für die Säuberung des rechten Rückspiegels zuständig ist. Es gelingt, für diese verantwortungsvolle Aufgabe den welbekannten Scheibenwischer Mickey Häckaußen vom Sauber-Team abzuwerben. Daher ist es völlig klar, daß die Verantwortung für die Reinigung des Rückspiegels keinem anderen Mechaniker übertragen werden kann. Außerdem darf Schumi von sofort an nur losfahren, wenn zuvor der Rückspiegel gesäubert wurde.
Die Welt ist für die nächste Zeit wieder in Ordnung, bis sich Häckaußen eines Tages unglücklicherweise gerade in der Mittagspause befindet, als Schumi eine Rennpause einlegen muß, und erst zur Box kommt, nachdem der ungeduldige Fahrer schon wieder seinen Wunsch nach der nächsten Runde angemeldet hat. Da die Regeln bei Ferrari streng eingehalten werden, kann Häckaußen daher seinen Spiegel nicht putzen (und Schumi muß demzufolge in der Box bleiben).
Daraufhin wird die Vorgehensweise folgendermaßen geändert: Wenn Schumacher losfahren will, muß er so lange warten, bis alle gerade arbeitenden Mechaniker fertig sind; später hinzukommende dürfen nicht mehr mit Reparaturen beginnen, außer wenn es um die Reinigung des Rückspiegels geht. Ist der letzte Mechaniker allerdings fertig, bevor sich der Spiegelputzer blicken läßt, darf Schumi auch mit dreckigem Spiegel losfahren.

c. Synchronisieren Sie diese Strategie, indem Sie die Algorithmen für Fahrer und Mechaniker aus Teil (b) entsprechend modifizieren und einen semaphorbasierten Algorithmus für Mickey Häckaußen angeben! Verhindern Sie dabei, daß Schumi vor lauter Putzerei gar nicht mehr zum Fahren kommen könnte. Sie können davon ausgehen, daß der Spiegel vor der ersten Runde, die Schumi dreht, noch sauber sind.

Aufgabe 53: (T) Monitorkonzept in Java (1+2+2+1+1 Pkt.)

In dieser Aufgabe sollen Sie sich Unterschiede und Gemeinsamkeiten zwischen Java-Synchronisation und Monitoren klar machen.

a. Beschreiben Sie das grundlegende Konzept der Synchronisation in Java.

b. Wie werden `wait` und `signal` in Java umgesetzt.

c. Erläutern Sie die Besonderheiten von Java gegenüber „echten“ Monitoren (ohne Beachtung der Signalisierungsmechanismen).

d. Beschreiben Sie die Einschränkungen bei der Verwendung von `wait` und `notify` gegenüber „echten“ Monitoren.

e. Überlegen Sie sich, wie diese Einschränkungen umgangen werden können.

f. Es gibt zwei Modelle, wie die Ausführung in einem Monitor nach dem Aufruf von `signal` fortfährt (A: signalisierender Prozess, B: aufgeweckter Prozeß):

   - *Signal-and-wait:* A muß nach der Signalisierung warten, bis B den Monitor verlassen hat oder auf eine andere Bedingung wartet.
   - *Signal-and-continue:* B muß warten, bis A den Monitor verlassen hat oder auf eine andere Bedingung wartet.

   Welchem Modell folgt Java?

Aufgabe 54: (K) Grundlagen Java-Synchronisation (1+1+1+2+1 Pkt.)

Entscheiden Sie, ob die folgenden Aussagen wahr oder falsch sind. Begründen Sie ihre Aussage.

a. Eine Methode als `synchronized` zu deklarieren, garantiert, daß kein Deadlock auftreten kann.

b. Die folgende Klasse ist *thread-safe*, d.h. es treten keine unerwünschten Seiteneffekte auf, wenn mehrere Threads auf dem selben Objekt operieren:

```
class example1 {
    private boolean flag;
    private int count;

    public void switch() { flag = !flag; }
    public void set(int i) { count= i; }
    public int get(){ return count; }
}
```

c. Die folgende Klasse ist *thread-safe*:

```
class example2 {
    private boolean flag;
    private int count;

    public void switch() { flag = !flag; }
    public int inc() { return ++count; }
    public void dec() { return --count; }
}
```

d. Ein Aufruf der Methode `test` aus der folgenden Klasse wird nie „flag = true“ ausgeben.

```
class example3 {
    private boolean flag;

    public void switch() { flag = !flag; }
    public synchronized boolean test() {
        flag = false;

        if ( flag ) {
            System.out.println("flag = true");
        }
    }
}
```

e. Ein Aufruf der Methode `test` aus der folgenden Klasse wird stets „flag = true“ ausgeben.

```
class example3 {
    private volatile boolean flag;

    public void switch() { flag = !flag; }
    public synchronized boolean test() {
        flag = false;

        if ( flag ) {
            System.out.println("flag = true");
        }
    }
}
```

Aufgabe 55: (P) Mutual Exclusion (2+2+4 Pkt.)

Betrachten Sie die folgende abstrakte Klasse zur Realisation eines kritischen Bereichs:

```
public abstract class MutualExclusion
{
   /**
    * critical and non-critical sections are simulated by sleeping
    * for a random amount of time between 0 and 3 seconds.
    */
   public static void criticalSection() {
      try {
            Thread.sleep( (int) (Math.random() * 3000) );
      }
      catch (InterruptedException e) { }
   }

   public static void nonCriticalSection() {
      try {
            Thread.sleep( (int) (Math.random() * 3000) );
      }
      catch (InterruptedException e) { }
   }

   public abstract void enteringCriticalSection(int t);

   public abstract void leavingCriticalSection(int t);

   public static final int TURN_0 = 0;
   public static final int TURN_1 = 1;
}
```

Implementieren Sie eine Ableitung dieser Klasse für

a. den einfachen Schalten-Algorithmus (entspricht dem 1. Ansatz des Decker-Algorithmus),

b. den einfachen Sperren-Algorithmus (entspricht dem 2. Ansatz des Decker-Algorithmus),und

c. die Kombination aus Schalten- und Sperren-Algorithmus (entspricht dem 3. Ansatz des Decker-Algorithmus),.

Testen Sie ihre Implementation mit Hilfe zweier Threads, die nebenläufig ausgeführt werden. Testen Sie, wenn möglich, auf einer JVM-Implementation mit Timeslicing.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 8
04.12.2001

Informatik III

Achtung: Die **Anmeldung** zur **1. Klausur** läuft in dieser Woche noch bis **Freitag, 07.12.2001 12 Uhr** über die Info3–Homepage.
Die Anmeldung ist **obligatorisch** und Nachmeldungen nach dem letzten Termin werden **nicht akzeptiert!**

Lesen: Silberschatz Kap. 6.1, Kap. 6.2
Wiederholen: Silberschatz Kap. 7
Vertiefen: Stallings Kap. 4

Aufgabe 45: (H) Nebenläufige Abläufe (3 Pkt.)

Betrachten Sie das untenstehende nebenläufige Programm mit zwei Prozessen, P und Q. A, B, C, D und E sind beliebige atomare Anweisungen. Die beiden Prozesse (realisiert als Prozeduren) seien parallel gestartet:

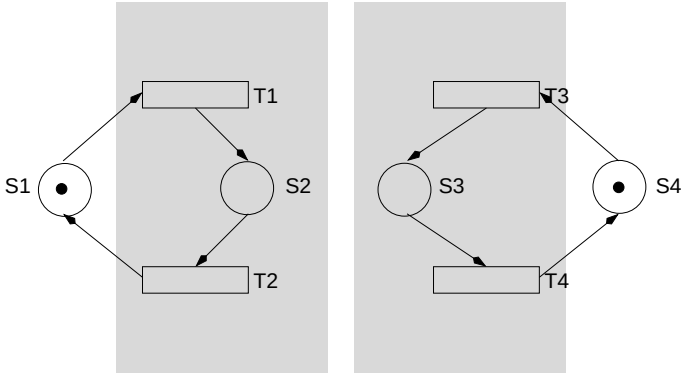
```
PROCEDURE P;
BEGIN
  A;
  B;
  C;
END;

PROCEDURE Q;
BEGIN
  D;
  E;
END;
```

Geben Sie alle möglichen Abläufe dieses Programmes an.

Aufgabe 46: (H) Einführung Petri-Netze (2+3+2 Pkt.)

Gegeben seien zwei kritische Regionen eines Verteilten Systems. Diese umfassen die Transitionen T1 und T2 bzw. T3 und T4, sowie die jeweils eingeschlossenen Prozesse „Druckjob A“ bzw. „Druckjob B“. Nun stehe im System nur ein Drucker zur Verfügung, der zu einem bestimmten Zeitpunkt nur von höchstens einem der Druckjobs in Anspruch genommen werden kann.



a. Ergänzen Sie das dargestellte Petri-Netz unter Hinzufügung einer möglichst minimalen Anzahl von Stellen und/oder Transitionen so, daß sich nicht mehr beide Prozesse gleichzeitig in ihren kritischen Regionen befinden können.

b. Geben Sie den Erreichbarkeitsgraphen Ihres erweiterten Petri-Netzes an!

c. Entscheiden Sie, ob es zu einem Deadlock oder einer teilweisen Verklemmung kommen kann, und begründen Sie Ihre Aussage!

Aufgabe 47: (H) Petrigotchi (4+2 Pkt.)

Sie haben ja bereits das virtuelle Haustier „Tamagottochi“ kennengelernt. Nun ist es möglich, daß sich n Pfleger um m Tamagottochi kümmern. Die Tamagottochi können glücklich, hungrig, schmutzig oder gelangweilt sein. Ein Wechsel zwischen diesen Zuständen ist nicht möglich. Ausdauernder Hunger führt zum Tode und ausdauernde Langeweile zur Flucht. Um dies zu verhindern beschäftigt sich der Pfleger, falls er zugegen ist, mit dem Tamagottochi entsprechend den Bedürfnissen (füttern, waschen, spielen). Ein Pfleger soll zuerst ein hungriges Tamagottochi füttern, bevor er ein schmutziges wäscht, bevor er mit einem gelangweilten spielt. Steht ein Pfleger zur Verfügung, so wird er weder zulassen, daß ein Tamagottochi vor Hunger stirbt noch wegen Langeweile flieht.
Zur Modellierung werden sogenannte *Inhibitor-Arcs* benötigt. Das sind Kanten mit einem Kreis statt einer Spitze, die von einer Stelle zu einer Transition führen und verhindern, daß diese feuert, solange sich noch Marken in der Stelle befinden.

a. Modellieren Sie die beschriebene Situation in einem Petrinetz! Sie benötigen dazu 10 Stellen und 11 Transitionen.

b. Stellen Sie den dazugehörigen Erreichbarkeitsgraphen für $n = m = 1$ auf.

Aufgabe 48: (T) Hyman-Lösung (4+4+2 Pkt.)

Im September 1965 veröffentlichte *E. W. Dijkstra* in der (damals wie heute hochrenommierten) Zeitschrift „Communications of the ACM“ eine Lösung für das wechselseitige Ausschlußproblem für n Prozesse, die die Kriterien für „mutual exlusion“ und „progress“ erfüllte.
Einige Monate später, im Januar 1966, veröffentlichte *Harris Hyman* wiederum in den „Communications of the ACM“ eine Vereinfachung von Dijkstras Lösung für den Fall von nur *zwei* Prozessen i und j, $i \in \{0, 1\}$, $j = i - 1$, und gab für den Prozeß mit Nummer i den folgenden Algorithmus an (in Pseudo-Code):

```
ARRAY    moechtenicht[0..1] OF BOOLEAN;
INTEGER  i, j, Schalter;

C0:  moechtenicht[i] := FALSE;
C1:  IF Schalter <> i THEN
     BEGIN
C2:     IF moechtenicht[j] = FALSE THEN GOTO C2
          ELSE Schalter := i;
        GOTO C1;
     END;
     ELSE Kritischer Bereich;

     moechtenicht[i] := TRUE;
     Unkritischer Bereich;

     GOTO C0;
```

a. Entwickeln Sie zur Veranschaulichung dieses Algorithmus das zugehörige Flußdiagramm und erläutern Sie dieses!

b. Wiederum einige Monate später, im Mai 1966, druckte dieselbe Zeitschrift einen Beitrag von *Donald E. Knuth* ab, in dem er zu den beiden erwähnten Algorithmen von Dijkstra und Hyman einleitend bemerkte:
„... Ich hoffe, mit folgendem Beitrag einige Leser vor den Problemen bewahren zu können, die bei einer Verwendung der beiden Algorithmen auftreten. Insbesondere ist es einfach, ein Gegenbeispiel zu Mr. Hymans „Lösung“ zu finden ...“

Sehen Sie das auch so? Zeigen Sie anhand von Gegenbeispielen, daß der oben angegebene Algorithmus von Hyman zwei der drei bekannten Kriterien für eine korrekte Lösung *nicht* erfüllt!

Aufgabe 49: (K) Deadlock-Erkennung (5+4 Pkt.)

Sie haben bereits Deadlock-Erkennung über Resource-Allocation-Graphen kennengelernt. Hier sollen Sie sich eine algorithmische Lösung entwickeln und *Recovery through Killing Processes* untersuchen.

a. Entwickeln Sie einen Algorithmus, der benutzt werden kann, um zu erkennen, ob ein Deadlock vorliegt oder nicht.

Vielleicht sind Ihnen dabei die folgenden drei Matrizen nützlich:

- `available` – Anzahl der verfügbaren Ressourcen jedes Typs.
- `allocation` – Anzahl der Ressourcen, die für den jeweiligen Prozeß bereits zugeordnet sind.
- `request` – Matrix der Anforderungen der Prozesse bezüglich der Ressourcen.

b. Wenn erkannt wird, daß ein Deadlock vorliegt, ist eine bereits bekannte Lösung, diejenigen Prozesse zu terminieren, deren Terminierung den Deadlock auflößt und mit den geringsten Kosten verbunden ist. Welches Problem kann bei dieser Strategie auftreten? Wie würden Sie es lösen?

Aufgabe 50: (P) Ringpuffer (4+2+1 Pkt.)

a. Schreiben Sie eine Klasse `RingPuffer`, die den in der Vorlesung vorgestellten Ringpuffer in Java implementiert. Die Klasse soll wenigstens die folgenden vier Methoden aufweisen:

- `isEmpty` – zeigt an, ob der Puffer leer ist.
- `isFull` – zeigt an, ob der Puffer voll ist.
- `insert` – fügt ein Objekt in den Puffer ein.
- `remove` – löscht das geeignete Element aus dem Puffer und gibt es zurück.

Beim Erzeugen einer Instanz von `RingBuffer` soll die Maximalgröße spezifiziert werden. Es soll möglich sein, beliebige Java-Objekte einzufügen.

b. Testen Sie Ihre Implementation durch Einfügen und Entfernen von 10 Zufallszahlen! Wie muß dazu die Maximalgröße initialisiert werden.

c. Erklären Sie, wie durch einen Ringpuffer das Reader/Writer-Problem für zwei Prozesse gelöst werden kann.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 7
27.11.2001

Informatik III

Achtung: Die **Anmeldung** zur **1. Klausur** ist von
Montag, 03.12.2001 10 Uhr bis **Freitag, 07.12.2001 12 Uhr** möglich.

Die Anmeldung ist **obligatorisch** und Nachmeldungen werden **nicht akzeptiert!**

Lesen: Stallings Kapitel 6.1 bis einschl. 6.4

Aufgabe 39: (H) Deadlock-Vermeidung (4+2 Pkt.)

Eine Methode Deadlocks zu vermeiden ist es, dass man eine der Bedingungen für das Entstehen von Deadlocks im vorhinein auszuschließen.

a. Geben sie die vier Voraussetzungen für die Entstehung eines Deadlocks an.

b. Beschreiben Sie wie durch eine Ordnung der Ressourcen bei geeigneter Reservierungsstrategie Deadlocks vermieden werden können.

Aufgabe 40: (H) Behebung von Deadlocks (4+2 Pkt.)

a. Nennen Sie vier Ansätze zur Behandlung von (bereits eingetretenen) Deadlocks.

b. In einem Rechensystem seien 8 Magnetbänder vorhanden, um die n Prozesse konkurrieren. Jeder dieser Prozesse benötige maximal 3 Magnetbänder. Für welche Werte von n besteht keine Verklemmungsgefahr? Begründen Sie Ihre Antwort.

Aufgabe 41: (T) Prozeßfortschrittsdiagramm (3+2 Pkt.)

In dieser Aufgabe sollen Sie sich mit Prozeßfortschrittsdiagrammen vertraut machen und einige Eigenschaften kennenlernen.

a. Gegeben seien zwei Prozesse A und B. A benötigt zu seiner Ausführung zwölf Zeiteinheiten, B zehn Zeiteinheiten. Es stehen 3 Betriebsmittel (BM) zur Verfügung, die von den Prozessen während ihrer Ausführung benötigt werden. A benötigt BM1 im Zeitraum $2-6$, BM2 in $4-7$, BM3 in $8-10$ und Prozeß B benötigt BM1 in $5-8$, BM2 in $4-7$ und BM3 in $1-3$.

   Skizzieren Sie das Prozeßfortschrittsdiagramm! Zeichnen Sie unmögliche und unsichere Bereiche ein! Wieviele prinzipiell verschiedene Möglichkeiten gibt es, die Prozesse A und B terminieren zu lassen? Zeichnen Sie diese ein.

b. Gegeben seien zwei Prozesse A und B, A benötigt zu seiner Ausführung zehn Zeiteinheiten, B neun Zeiteinheiten. Ferner stehen sechs verschiedene Betriebsmittel (BM) zur Verfügung, die von den Prozessen während ihrer Ausführung benötigt werden. Die folgende Tabelle zeigt, in welchen Intervallen die beiden Prozesse BM belegen:

Prozeß	BM1	BM2	BM3	BM4	BM5	BM6
A	$1-2, 8-9$	$3-4, 5-8$	$3-5$	$4-6$	$2-4$	$7-8$
B	$1-3, 5-8$	$6-7$	$3-5$	$2-3$	$7-8$	$4-6$

Geben Sie das Prozeßfortschrittsdiagramm an und zeichen Sie alle prinzipiell verschiedenen Möglichkeiten ein, die Prozesse A und B terminieren lassen.

c. Zwei Prozesse A und B benötigen zu ihrer Ausführung je eine gewisse Zeitdauer t_A bzw. t_B und die drei Betriebsmittel X, Y und Z. t_{ij} bezeichne die Zeitdauer, die das Betriebsmittel i von Prozeß j während der Ausführungszeit benötigt wird. Ein Experte macht nun folgende Aussagen:

- Ein Deadlock tritt auf keinen Fall auf, wenn

$$t_{XA} + t_{YA} + t_{ZA} < t_A \quad \text{und} \quad t_{XB} + t_{YB} + t_{ZB} < t_B.$$

- Ein Deadlock kann nicht auftreten, wenn

$$t_{XA} \cdot t_{XB} + t_{YA} \cdot t_{YB} + t_{ZA} \cdot t_{ZB} < t_A \cdot t_B.$$

- Ein Deadlock liegt automatisch vor, wenn

$$t_{XA} + t_{YA} + t_{ZA} > t_A \quad \text{und} \quad t_{XB} + t_{YB} + t_{ZB} > t_B.$$

- Ein Deadlock liegt automatisch vor, wenn

$$t_{XA} \cdot t_{XB} + t_{YA} \cdot t_{YB} + t_{ZA} \cdot t_{ZB} > t_A \cdot t_B.$$

Bewerten Sie diese Aussagen, indem Sie entweder einen Beweis oder ein Gegenbeispiel (in Form eines Prozeßfortschrittsdiagramms) angeben.

Aufgabe 42: (T) Deadlock-Detection and -Recovery (4+2+3+4+4 Pkt.)

Diese Aufgabe vertieft eine Methode zur Deadlock-Erkennung aus der Vorlesung: den **Resource-Allocation-Graph**. In einem Resource-Allocation-Graph bezeichnet

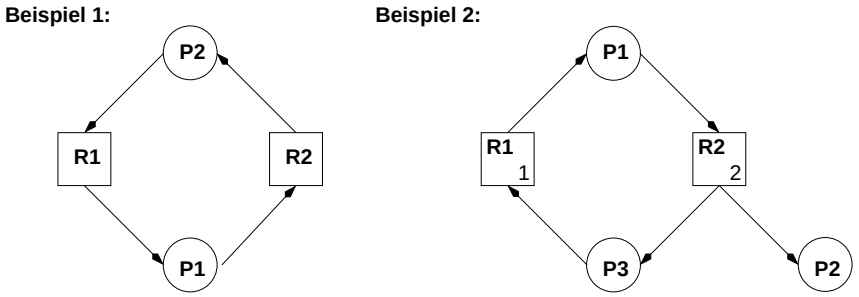
- ein Rechteck eine Ressource R_i, wobei die Anzahl der Instanzen dieser Ressource ebenfalls angegeben werden können,
- ein Kreis einen Prozeß P_i,
- eine Kante (P_i, R_j) mit dem Gewicht n die Anforderung von n Einheiten der Ressource R_j durch den Prozeß P_i,
- eine Kante (R_i, P_j) mit dem Gewicht n die Belegung von n Einheiten der Ressource R_i durch den Prozeß P_j.

Gibt es in einem solchen Graphen keine Zyklen, so besteht auch kein Deadlock in dem zugrundeliegenden Prozeßsystem.
Wenn es einen Zyklus im *Resource-Allocation-Graph* gibt so unterscheide man zwei Fälle:

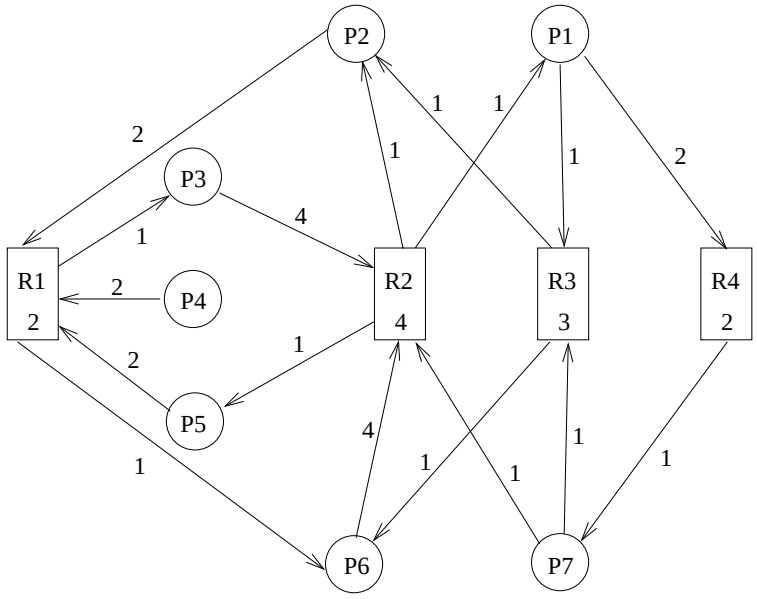
– Besitzt jede Ressource höchstens eine Instanz, so liegt ein Deadlock vor (notwendige und hinreichende Bedingung).

– Besitzt jede Ressource mehr als eine Instanz besitzt, so kann ein Deadlock vorliegen, muß jedoch nicht. In diesem Fall ist der Zyklus eine notwendige, nicht jedoch hinreichende Bedingung.

In der folgenden Abbildung ist also der erste Graph verklemmt, der zweite jedoch nicht (wenn P_2 fertig ist, kann P_1 eine Instanz von R_2 belegen und damit auch fortfahren):



Sei nun der folgende markierte *Resource-Allocation-Graph* G für sieben Prozesse $P_1, \ldots, P_7$ und vier Betriebsmittel (-klassen) $R_1, \ldots, R_4$ gegeben:



a. Geben Sie für alle Prozesse P_i und alle Betriebsmittel R_j folgende Größen an:

   – $b(P_i, R_j)$: Die von Prozeß P_i belegten Einheiten des Betriebsmittels R_j
   – $f(P_i, R_j)$: Die von Prozeß P_i geforderten Einheiten des Betriebsmittels R_j
   – $\nu(R_j)$: Die Anzahl der insgesamt vorhandenen Einheiten des Betriebsmittels R_j
   – $a(R_j)$: Die Anzahl der noch zur Verfügung stehenden Einheiten des Betriebsmittels R_j

b. Offensichtlich läßt sich auf ein solches System die folgende Reduzierung anwenden: Gibt es einen Prozeß P_i, so daß für alle j gilt $f(P_i, R_j) \leq a(R_j)$, so entferne man alle Kanten zu P_i aus dem Graphen und setze für alle j $a(R_j) := a(R_j) + f(P_i, R_j)$.

   Reduzieren Sie G soweit möglich und kommentieren Sie das Ergebnis.

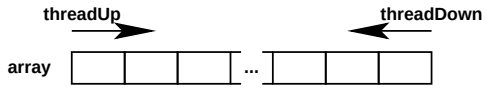
c. Nennen Sie drei Methoden der Deadlock-Behebung (*Deadlock-Recovery*)?

d. Bestimmen Sie, falls im Graphen eine Verklemmung vorliegt, eine minimale Prozeßmenge $\mathrm{mp}_{\min} \subseteq \{P_1, \ldots, P_7\}$, so daß nach Abbruch aller Prozesse in $\mathrm{mp}_{\min}$ keine Verklemmung mehr vorliegt.

e. Zu welchem Ergebnis kommen Sie, wenn das Betriebsmittel R_1 aus zwei Einheiten mehr besteht?

Aufgabe 43: (K) Java-Programmierung (10+5 Pkt.)

Es soll eine Suche nach einem Maximum (mit $\mathrm{Maximum} > 0$) in einem Feld positiver ganzer Zahlen parallelisiert werden. Dazu sollen 2 Threads dieses Feld durchsuchen. Ein Thread `threadUp` durchsucht das Feld aufsteigend, bezogen auf den Index, und ein Thread `threadDown` durchsucht das Feld absteigend.



Die Suche soll terminieren, wenn sich die beiden Threads mit ihren (aktuellen) Indizes überschneiden. Beide Threads sind Instanzen derselben Klasse `ArraySearch`.

a. Implementieren Sie die Klasse `ArraySearch`. Die benötigten Methoden sind:

   (i) Konstrukor
   (ii) Rückgabe des aktuellen Index
   (iii) Rückgabe des gefundenen Maximums
   (iv) Suche (aufwärts)
   (v) Suche (abwärts)
   (vi) Referenz auf den anderen Thread setzen
   (vii) `run()`-Methode

b. Implementieren Sie die Klasse `Test`, welche die beiden Threads startet und das gefundene Maximum zurückgibt.

Aufgabe 44: (P) Java-Tanz mit Prioritäten (6 Pkt.)

Fügen Sie nun dem Programm einen Button `Express` hinzu, der einen `Ball`-Thread mit maximaler Priorität startet. Die über `Start` gestarteten Threads sollen mit normaler Priorität ablaufen. Beobachten Sie die Auswirkung der unterschiedlichen Prioritäten.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 6
20.11.2001

Informatik III

Lesen: Tanenbaum Kap. 2.4, Stallings Kap. 6
Vertiefen: Silberschatz Kap. 5

Aufgabe 34: (H) Dijkstras Banker's Algorithm (4+2+2 Pkt.)

Der in der Vorlesung besprochene *Banker's Algorithm* von Dijkstra soll in dieser Aufgabe angwendet werden. Eine Bank stellt ihren Kunden einen Kreditrahmen (KR) zur Verfügung. Damit die Kreditwünsche der Kunden erfüllt werden können, stellt die Bank 10.000 DM zur Kreditvergabe bereit. Folgende Tabelle zeigt die Kreditrahmen der Kunden und die einzelnen Kreditanfragen:

Kunde	KR (in TDM)	Anfragen (in TDM)
A	6	2, 1, 3
B	2	1, 1
C	8	3, 4, 1
D	4	3, 1

a. Finden Sie eine möglichst kurze Verfügungsreihenfolge, die alle Kreditwünsche befriedigt.

b. Genügt es, um einen Deadlock sicher vermeiden zu können, nur den nächsten Folgezustand zu berücksichtigen?

c. Diskutieren Sie die tatsächliche Anwendung des *Banker's Algorithm* in einem Betriebssystem.

Aufgabe 35: (T) Scheduling in 4.3 BSD Unix (10+2 Pkt.)

Eine wesentliche Aufgabe der Prozessorverwaltung besteht darin, zu entscheiden, welcher der um den bzw. die Prozessor(en) konkurrierenden Prozesse zu einem bestimmten Zeitpunkt an einen Prozessor gebunden wird. Es werden Prozessorverwaltungsstrategien benötigt.
Eine Strategie für die Prozessorverwaltung besteht darin, an die Prozesse **Prioritäten** zu vergeben. Der Prozessor wird dann jeweils dem Pozeß mit der höchsten Priorität zugeteilt. Die Prioritätenvergabe kann dabei statisch oder dynamisch sein. Im ersten Fall hat jeder Prozeß für die Dauer seiner Existenz eine feste Priorität; im zweiten Fall können sich die Prioritäten der Prozesse dynamisch verändern, d.h. sie werden in gewissen Zeitabständen neu berechnet.
Bei einer **Zeitscheibenstrategie** werden die Prozesse an den Prozessor jeweils für ein festgelegtes Zeitquantum (in 4.3 BSD Unix beträgt z.B. die Zeitscheibe 100 ms) gebunden und spätestens nach dem Ablauf dieser Zeitspanne wird den Prozessen der Prozessor wieder entzogen.
Zeitscheibenstrategien und Prioritätenvergabe können zu effizienten Verwaltungsstrategien kombiniert werden.
In dieser Aufgabe wird das Scheduling in dem Betriebssystem **4.3 BSD Unix**[1] genauer betrachtet. Beim Unix-Scheduling handelt sich um eine Zeitscheibenstrategie mit dynamischer Prioritätenvergabe. Unix vergibt für seine Prozesse Prioritäten von 0 bis 127 (0 ist die höchste Priorität),

[1] siehe: Leffler u.a.: Das 4.3 BSD Unix Betriebssystem, Addison-Wesley

die in 32 Warteschlangen verwaltet werden. Ein Prozeß mit Priorität PRIO wird in die Schlange $\mathrm{PRIO}/4$ eingeordnet. Alle Prozesse einer Prioritätsklasse befinden sich in einer Warteschlange, die nach einer Round-Robin Strategie abgearbeitet wird. Zunächst wird allen Prozessen der höchsten Priorität die CPU zugeteilt bis die Warteschlange leer ist. Dann kommen die Prozesse mit der nächstniedrigeren Priorität zum Zuge.
Die Prioritäten werden jedoch fortlaufend neu berechnet (*multilevel-feedback-queue*). Die Prioritäten der Prozesse, die in einem gewissen Zeitabschnitt viel Rechenzeit verbraucht haben, werden erniedrigt; Prozesse, die lange gewartet haben, erhalten eine höhere Priorität. Die Prioritäten werden folgendermaßen berechnet:

$$(1) \quad \textbf{u\_prio} = \mathrm{USER\_PRIO} + \mathrm{p\_cpu}/4 + 2 \cdot \mathrm{p\_nice}, \text{ wobei}$$

$\mathrm{p\_cpu}$ die Prozessornutzung des rechnenden Prozesses ist und alle 10 ms um eins inkrementiert wird. $\mathrm{p\_nice}$ ist ein vom Benutzer bestimmter Gewichtungsfaktor ($-20 \leq \mathrm{p\_nice} \leq 20$) und $\mathrm{USER\_PRIO}$ ist die Priorität, die dem Prozeß beim Start zugeteilt worden ist. $\mathrm{p\_cpu}$ wird jede Sekunde angepaßt durch:

$$(2) \quad \mathrm{p\_cpu} = ((2 * \mathrm{load})/(2 * \mathrm{load} + 1)) * \mathrm{p\_cpu} + \mathrm{p\_nice}, \text{ wobei}$$

load eine Abschätzung der CPU-Auslastung ist. Die Anpassung (2) sorgt dafür, daß bisher verbrauchte Rechenzeit nach einer gewissen Zeit nicht mehr ins Gewicht fällt.

a. In dieser Teilaufgabe sollen Sie sich das Unix-Scheduling an einem kleinen Beispiel verdeutlichen. Gegeben seien dazu folgende Prozesse mit ihrer Bedienzeit und Anfangspriorität:

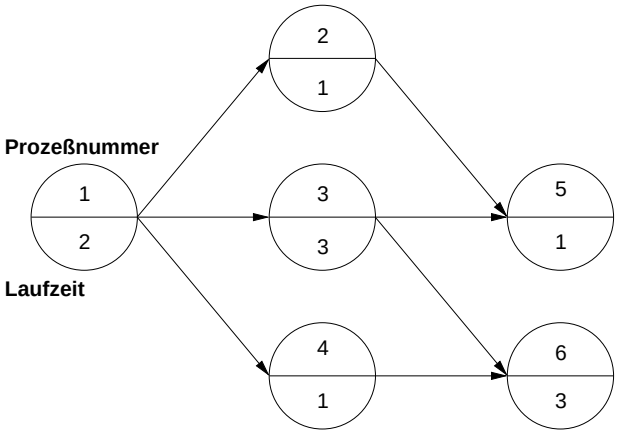
Prozeß	Bedienzeit	Priorität
1	10	3
2	1	1
3	2	3
4	1	4
5	5	2

Zeichnen Sie die Abarbeitungsfolge der Prozesse, die sich bei Anwendung der oben erläuterten Strategie ergibt, geeignet auf. Zur Vereinfachung nehmen Sie an, daß ein Zeitquantum den Wert 1 hat, $\mathrm{p\_cpu}$ (des rechnenden Prozesses!) in jedem Zeitquantum um 8 erhöht wird, $\mathrm{p\_nice}$ den Wert 0 hat und load gleich 1 ist. Die Priorität aller Prozesse soll nach jedem vierten Quantum neu berechnet werden.

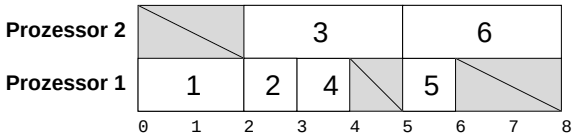
b. Welche Beweggründe stehen hinter der beschriebenen Strategie?

Aufgabe 36: (T) Grahams List Scheduling (5+12 Pkt.)

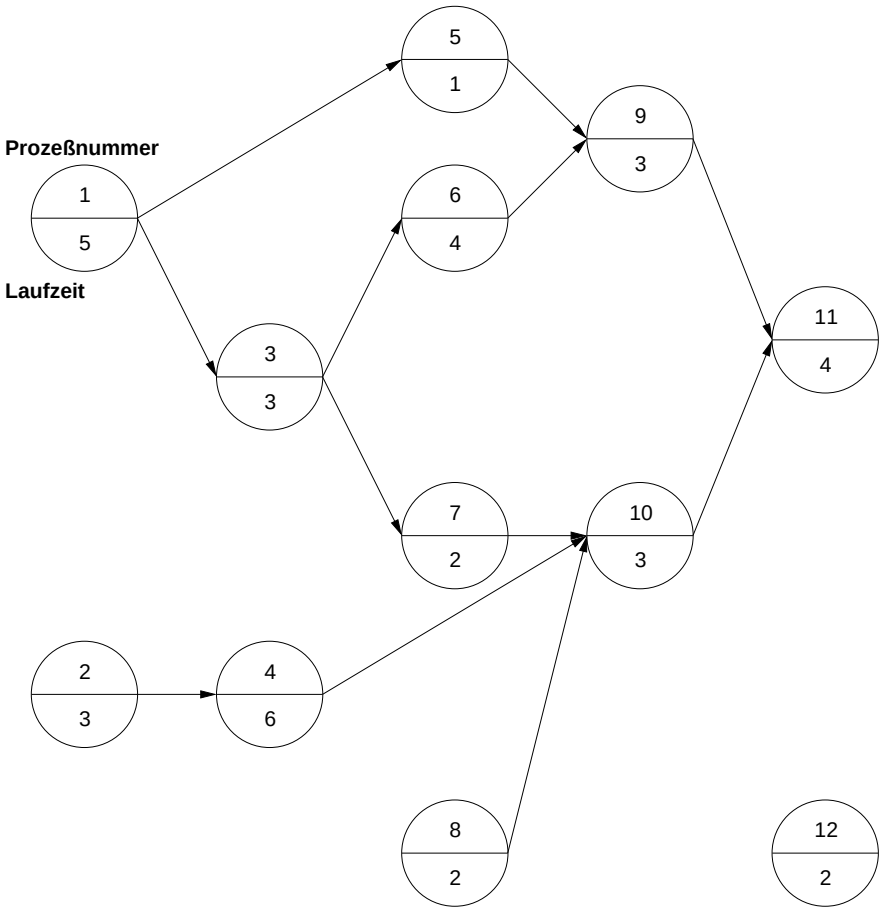
In der heutigen Zeit ist es interessant, Programme auch auf mehreren Prozessoren lösen zu können. Dafür gibt es z.B. Grahams List-Scheduling, welches nach dem Greedy Prinzip arbeitet, das heißt, daß ein freier Prozessor den ersten rechenbereiten Prozeß aus der Prozeßschlange (geordnet nach Prozeßnummern) nimmt und bearbeitet. Sind mehrere Prozessoren frei, bedient sich der mit der kleineren Nummer zuerst.
Seien sechs Prozesse (gegeben durch Nummer und Laufzeit), wie im folgenden Diagramm dargestellt, voneinander abhängig:



Prozesse 2, 3 und 4 müssen also erst auf das Ergebnis von Prozeß 1, 5 auf das von 2 und 3 etc. warten.
Bei zwei Prozessoren ergibt sich folgendes Gantt-Diagramm:



a. Geben Sie für folgendes Diagramm das zugehörige Gantt-Chart für drei Prozessoren an:



b. Bei Grahams List-Scheduling existieren drei Anomalien: Die Laufzeit kann steigen bei

(i) Erhöhung der Prozessoranzahl,

(ii) Entfernung von Kanten und

(iii) Reduzierung der Dauer eines Prozesses.

Geben Sie jeweils ein Beispiel an.

Aufgabe 37: (K) Vergleich zweier Schedulingstrategien (6+2+3 Pkt.)

Diese Aufgabe stammt aus einer Semestralklausur der TU. In dieser Aufgabe sollen die folgenden zwei Strategien zur RK-Vergabe verglichen werden:

– **Shortest job first (SJF):** Vergabe des RK an denjenigen Auftrag mit der geringsten Gesamt-Bedienzeit (non-preemptive).

– **Shortest remaining processing time (SRPT):** Vergabe des RK an denjenigen Auftrag mit der geringsten Rest–Bedienzeit (preemptive, *preemption* bei Auftragsankunft).

Gegeben seien die Aufträge $P_1, \ldots, P_8$ mit den folgenden Ankunfts- und Bedienzeiten:

Auftrag	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Ankunftszeit	0	3	3	4	9	11	19	21
Bedienzeit	6	4	2	1	3	7	3	2

a. Berechnen Sie jeweils für SJF und SRPT für jeden der obigen Aufträge die Verweilzeit V_i und die Wartezeit W_i ($i = 1, \ldots, 8$).

Anmerkung: Sollten Sie diese Aufgabe nicht lösen können, so verwenden Sie im folgenden die Werte

Auftrag	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Wartezeit(SJF)	0	10	4	3	1	4	6	3
Wartezeit(SRPT)	2	9	1	1	0	11	1	2

b. Berechnen Sie jeweils für SJF und SRPT die sich ergebende mittlere Wartezeit $\overline{W}$. Wodurch läßt sich der Unterschied zwischen SJF und SRPT in Bezug auf die mittlere Wartezeit begründen?

c. Tragen Sie jeweils für SJF und SRPT in einem Diagramm die absolute Häufigkeit einer bestimmten Wartezeit gegen die Größe der Wartezeit auf. Vergleichen Sie qualitativ die beiden sich ergebenden Häufigkeitsverteilungen, und begründen Sie die Unterschiede.

Aufgabe 38: (P) Unfairer Java-Tanz (8 Pkt.)

Erweitern Sie das `Ball`-Programm mit Threads (vom vorletzten Übungsblatt), um einen Button `Selfish`, das einen `Ball` erzeugt, der nicht mittels `sleep`, sondern mittels

```
long t = System.currentTimeMillis();
while (System.currentTimeMillis() < t + 5)
  ;
```

busy wartet.
Beobachten Sie die Auswirkung auf Ihrem System, variieren Sie dabei auch die Prioritäten der `SelfishBall`s.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 5
13.11.2001

Informatik III

Lesen: Tanenbaum Kap. 2.4, Stallings Kap. 6
Vertiefen: Silberschatz Kap. 5

Aufgabe 28: (H) CPU-Scheduling – Einführung (5 Pkt.)

Folgende Prozesse sollen betrachtet werden (die Zeiten seien in beliebigen Zeiteinheiten gegeben, die Priorität von 0 bis 2, wobei 0 die höchste Priorität bezeichne):

Prozeß	Ankunftszeitpunkt	Laufzeit	Priorität
A	0	4	1
B	0	6	0
C	1	1	0
D	2	7	1
E	4	3	2
F	4	2	1
G	6	2	0
H	10	6	0
I	11	1	1

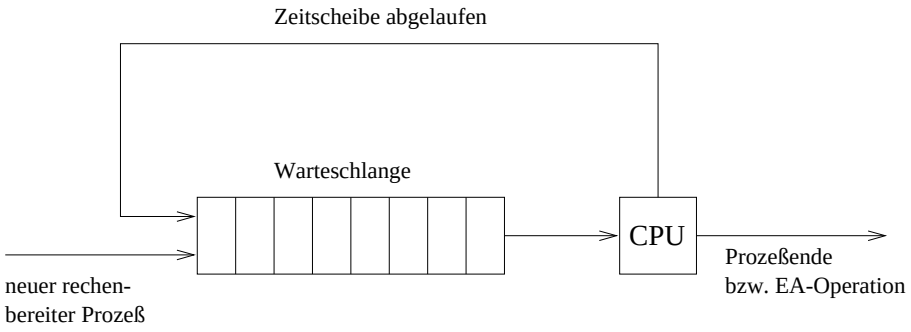
Ein Prozeß, der zum Zeitpunkt t eintritt, wird erst zum Zeipunkt $(t+1)$ berücksichtigt. Kommen zwei Prozesse zur gleichen Zeit, so wird analog zum Bakery Algorithmus entschieden. Wird ein Prozeß vor seinem Terminieren zum Zeitpunkt t' unterbrochen, so reiht er sich hinten in die Warteschlange mit Ankunftszeit t' wieder ein.
Geben Sie für die Strategien FIFO, SJF (SPN), Round Robin mit Quantum $t = 3$, Priority-Scheduling (non-preemptive) und Priority-Scheduling (preemptive) jeweils in Form eines Gantt-Charts für die ersten 20 $(1-20)$ Zeiteinheiten an, wann welchem Prozeß Rechenzeit zugeteilt wird und wann die Prozesse ggf. terminieren.

Aufgabe 29: (H) Dispatcher (8 Pkt.)

Für die Realisierung von Prozessen auf realen Prozessoren sind Verwaltungsoperationen auszuführen, die die Zustandsübergänge zwischen real rechnend und rechenbereit der Prozesse durchführen. Die entsprechenden Operationen (Programme), die zum Dispatcher zusammengefaßt werden, sind das *Binden* eines Prozesses an einen Prozessor und das *Lösen* der Bindung.
Geben Sie eine informelle Beschreibung der Schritte an, die beim Binden und Lösen durchzuführen sind! Welche Informationen und Datenstrukturen werden benötigt?

Aufgabe 30: (H) Round-Robin-Scheduling (3+3+2 Pkt.)

Eine gebräuchliche Strategie zur Rechnerkernvergabe stellt das in der Vorlesung vorgestellte **Round-Robin-Verfahren** dar. Hierbei wird eine Warteschlange von Prozessen verwaltet, wobei jeweils dem in der Schlange ersten Prozeß für eine feste Zeitdauer Q (die sog. *Zeitscheibe*) die CPU zugeteilt wird. Prozesse, deren Zeitscheibe abgelaufen ist, werden am Ende der Warteschlange wieder eingereiht.



Empirische Messungen haben ergeben, daß ein Prozeß im Mittel für eine Zeitspanne T die CPU benützt, bevor er das obige Warteschlangensystem aufgrund einer angestoßenen EA-Operation bzw. bei Prozeßende verläßt. Ein Prozeßwechsel benötige eine Zeitdauer S, die als *overhead* verloren geht.

a. Geben Sie für Round-Robin scheduling mit einer Zeitscheibe Q eine Formel für die CPU-Auslastung an.

b. Welche CPU-Auslastung ergibt sich in den folgenden Fällen:

   - $Q > T$,
   - $Q = S \ll T$ und
   - $Q \approx 0$.

c. Leiten Sie daraus Kritieren für den Wert von Q ab.

Dabei soll vorausgesetzt werden, daß stets rechenbereite Prozesse zur Verfügung stehen.

Aufgabe 31: (T) Exponential-Average (3+3+3+4 Pkt.)

Ein Problem bei der Realisierung von SJF (SPN) besteht darin, daß man a priori nicht weiß, wie lange ein Job dauern wird. Dieses Problem kann durch die Schätzung der Dauer des nächsten Prozesses unter Berücksichtigung der Erfahrungen aus der Vergangenheit gelöst werden. Eine Möglichkeit, aus der Vergangenheit zu lernen, verwendet das **exponential averaging**.
Dieses Verfahren beruht auf der Rekursionsformel für den $n+1$-ten Schätzwert eines Prozesses aus einer Quelle

$$\tau_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot \tau_n.$$

Dabei sei t_n die tatsächliche Dauer, τ_n die geschätzte Dauer des n-ten Prozesses und $0 \leq \alpha \leq 1$.
Gegeben sei ein System, das Prozesse aus drei verschiedenen Quellen A, B und C zugeteilt bekommt. Die von den letzten paar Prozessen jeweils benötigte CPU-Zeit ist in der folgenden Tabelle aufgeführt:

n	A	B	C
1	49	55	14
2	52	60	20
3	50	51	87
4	51	40	39
5	47	35	66
6	49	62	74
7	51	49	23
8	50	53	70

Nehmen Sie an, daß zu einem bestimmten Zeitpunkt alle bisher angefallene Arbeit erledigt ist und von jeder Quelle genau ein Prozeß beim CPU-Scheduler ankommt. Welche Reihenfolge legt der Scheduler unter den drei Prozessen für $n = 1, \ldots, 8$ fest, wenn er gemäß SJF verfährt und die zu erwartenden Prozeßdauern mit obiger Formel für

a. $\alpha = 0,2$,

b. $\alpha = 0,5$ bzw.

c. $\alpha = 0,8$

schätzt? Der Schätzwert τ_1 sei dabei für alle Quellen mit 50 initialisiert.

d. Was folgern Sie aus obigen Ergebnissen für die Wahl von α folgern?

Aufgabe 32: (K) Multi-Level-Feedback-Queueing (1+3+2+1+2 Pkt.)

Beantworten Sie die folgenden Fragen zum Multi-Level-Feedback-Queueing (MLFQ):

a. Warum ist es nicht sinnvoll, oberhalb der untersten Prioritätsklasse non-preemptive Scheduling-Verfahren zu benutzen?

b. Gegeben sei ein MLFQ-Scheduling mit drei Prioritätsklassen wie folgt:

Priorität	Verfahren
0	Round-Robin mit Quantum 1
1	Round-Robin mit Quantum 5
2	Round-Robin mit Quantum ∞ (FCFS)

Betrachten Sie die folgenden Prozesse:

Prozeß	Ankunftszeitpunkt	Laufzeit
A	0	11
B	1	9
C	2	1
D	11	6
E	17	4

Erstellen Sie ein Gantt-Chart und errechnen Sie die Wartezeiten der einzelnen Prozesse. Vergleichen Sie dieses mit den Wartezeiten, die sich bei einem herkömmlichen Round-Robin-Scheduling mit Quantum 3 ergeben würden. Gehen Sie dabei davon aus, daß ein Prozeß mit Ankunftszeit t zur Zeiteinheit $(t + 1)$ betrachtet werden kann.

c. Meist wird den Prioritätsklassen ein Round-Robin-Scheduling zugrunde gelegt, wobei niedrigere Prioritätsklassen höhere Zeitquanten erhalten (wie in Aufgabe (b)). Diskutieren Sie Vor- und Nachteile dieses Verfahrens gegenüber dem herkömmlichen Round-Robin-Verfahren.

d. Welche Vor- und Nachteile hat es, innerhalb der untersten Prioritätsklasse non-preemptives Scheduling zu verwenden?

e. Allgemein läßt sich das hier vorgestellte MLFQ dahingehend erweitern, Prozessen zu ermöglichen, in beide Richtungen die Prioritätsklassen zu wechseln. Wäre dies ihrer Meinung nach sinnvoll? Wenn ja, skizzieren Sie ein mögliches Verfahren, nach welchen Kriterien dies geschehen könnte!

Aufgabe 33: (P) Java Thread Scheduling (15 Pkt.)

Die Java Virtual Machine verwaltet Threads mittels eines preemptiven, prioritätsbasierten Schedulingalgorithmus. Alle Java-Threads werden mit einer Priorität versehen und die JVM führt denjenigen ausführbereiten Thread mit der höchsten Priorität aus; innerhalb der Prioritätsklassen wird FIFO verwendet.
Die JVM trifft die Entscheidung, welcher Thread ausgeführt wird, immer, wenn eines der folgenden Ereignisse eintritt:

a. Der aktuell laufende Thread verläßt den Zustand *Running* (z.B. durch Beenden der `run()`-Methode, durch das Warten auf I/O-Operationen oder `suspend()`).

b. Ein Thread mit einer höheren Priorität als der aktuelle Thread geht in den Zustand *Ready* über. In diesem Fall unterbricht die JVM den aktuellen Thread und führt den Prozeß mit höherer Priorität aus.

Die Spezifikation der JVM schreibt nicht vor, ob Zeitscheiben verwendet werden sollen – dies bleibt der jeweiligen Implementation der JVM überlassen. Wenn keine Zeitscheiben verwendet werden, wird ein Thread ausgeführt bis eines der obigen Ereignisse eintritt.
Da ein Java-Programm nicht erwarten kann, daß aktuelle Implementation der JVM Zeitscheiben zur Verfügung stellt, sollte ein Thread regelmäßig die Kontrolle an die CPU zurückgeben, implizit durch eintreten eines der oben genannten Ereignisse oder explizit durch den Aufruf von `yield()`. Diese freiwillige Rückgabe der Kontrolle an die CPU bezeichnet man als *kooperatives Multitasking*.

Eine andere Lösung stellt es dar, einen Round-Robin-Scheduler (mit Zeitscheiben) selbst zu implementieren. Dies läßt sich durch eine geschickte Nutzung der Prioritätsklassen einfach erreichen.
Implementieren Sie einen solchen Round-Robin-Scheduler und testen Sie ihn!

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 4
06.11.2001

Informatik III

Lesen: Stallings Kap. 3, Silberschatz Kap. 4
Vertiefen: Silberschatz Kap. 13.1 – 13.2

Aufgabe 21: (H) Threads – Gut oder Schlecht? (2+2 Pkt.)

In dieser Aufgabe sollen Sie ihr Verständnis für Threads vertiefen:

a. Nennen Sie zwei Gründe, warum es nicht sinnvoll ist, zuviele Threads zu verwenden.

b. Nennen Sie zwei Gründe, warum Threads sinnvoll/wichtig sind.

Aufgabe 22: (H) Prozessinformationen unter Linux (1+1+1 Pkt.)

Auf Linux-Systemen werden Informationen über den aktuellen Systemzustand, darunter vor allem die Prozessinformationen, in einem *Pseudodateisystem* **`/proc`** zugänglich gemacht. Die „Dateien" in diesem Verzeichnis bieten Zugriff auf diverse Datenstrukturen des Kernels.
Die Struktur von `/proc` ist in der zugehörigen Manpage der Sektion Dateiformate (5) beschrieben. Sie läßt sich z.B. mittels
```
man 5 proc
```
anzeigen.
Machen Sie sich mit der Struktur des `/proc`-Dateisystems vertraut und beantworten Sie die folgenden Fragen:

a. Wie findet man Informationen über Prozessstatus und PID, wenn man die Nummer eines Prozesses kennt?

b. Wie lassen sich diese Informationen ermitteln, wenn man nur den Befehlsnamen kennt?

c. Wozu dient der Link `cwd`?

Aufgabe 23: (H) Threads (4+3+2 Pkt.)

a. Welche Zustandsinformationen teilt ein Thread mit Threads desselben Prozesses und welche nicht?

b. Beschreiben Sie die Ähnlichkeiten und Unterschiede zwischen zwei unterschiedlichen Threads, die innerhalb desselben Prozesses ablaufen, und zwei unabhängigen Prozessen. Wann würden Sie zwei Threads in einem Prozeß, wann zwei unterschiedliche Prozesse verwenden?

c. Stimmt es, daß ein Kontextswitch zwischen zwei Threads desselben Prozesses weniger Aufwand verursacht als einer zwischen Threads in unterschiedlichen Prozessen?

Aufgabe 24: (T) Prozessormodi (2+2+2+2 Pkt.)

Das VAX/VMS-Betriebssystem verwendet vier Betriebssystemmodi, um Schutz und Sharing von Systemressourcen zwischen Prozessen zu ermöglichen. Die verschiedenen Modi legen fest:

- *Rechte zur Befehlsausführung:* Bestimmte Befehle dürfen nur in den höheren Modi verwendet werden.
- *Speicherzugriffsrechte:* Bestimmte Speicherbereiche (beispielsweise die anderer Prozesse) dürfen nur in höheren Modi angesprochen werden.

Die folgenden vier Modi werden verwendet:

Kernel: Speichermanagement, Interrupt-Behandlung, E/A-Operationen.

Executive: Viele Systemaufrufe, wie Datei- und Bandoperationen.

Supervisor: Betriebssystemdienste, wie Antworten auf Benutzerkommandos.

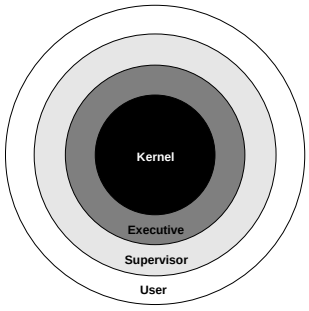
User: Benutzerprogramme; Hilfsprogramme wie Compiler, Editoren, Linker und Debugger.

Ein Prozeß, der in einem niederprivilegierten Modus abläuft, muß oft eine Prozedur in einem privilegierten Modus aufrufen (mit Hilfe von `CHM`), beispielsweise einen Betriebssystemdienst.

a. Viele Betriebssysteme besitzen nur zwei Modi: Kernel und User. Welche Vor- und Nachteile haben vier Modi gegenüber zwei.

b. Finden Sie ein Szenario, in dem sogar mehr als vier Modi sinnvoll wären!

Das hier vorgestellte VMS-Schema wird auch als **ring protection structure** bezeichnet, wie in der folgenden Abbildung dargestellt:



Das bekannte zwei Modischema ist genau genommen eine Ring-protection-structure mit nur zwei Ringen. Silberschatz und Galvin diskutieren ein Problem mit diesem Ansatz:

> The main disadvantage of the ring (hierarchical) structure is that it does not allow us to enforce the need-to-know principle. In particular, if an object must be accessible in domain D_i, but not accessible in domain D_j, then we must have $j < i$. But this means that every segment accessible in D_i is also accessible in D_j.

c. Erklären Sie in klaren Worten das Problem, das im vorangestellten Zitat angesprochen wird.

d. Schlagen Sie eine Methode vor, wie ein Betriebssystem mit einer Ringstruktur dieses Problem behandeln kann.

Aufgabe 25: (K) Java-Threadmodell (6 Pkt.)

Zeichnen Sie ein Threadübergangsdiagramm für das Java-Threadmodell. Markieren Sie die Übergänge mit den Ursachen für den Übergang. Tragen Sie auch *deprecated*-Methoden der Thread-Klasse ein und markieren Sie diese.

Aufgabe 26: (P) Java-Tanz (6+2 Pkt.)

Im folgenden sollen Sie ein Java-Programm entwerfen und implementieren, das einen Ball innerhalb einer gegebenen Fläche „tanzen“ läßt.

a. Schreiben Sie dazu eine Klasse **`Ball`**. Sie soll mit der „Tanzfläche“ initialisiert werden (z.B. ein `JPanel`-Objekt) und drei Methoden besitzen:

   (i) `draw` – zeichnet den Ball an der aktuellen Position. Die Größe des Balles soll durch statische Klassenvariablen festgelegt werden (z.B. Radius 10).

   (ii) `move` – bewegt den Ball einen Schritt (Schrittweite soll in einem Attribut festgelegt sein). Dabei soll der Ball von den Rändern der „Tanzfläche“ in einem 45 Grad Winkel abprallen.

   (iii) `bounce` – zeichnet den Ball an der Startposition und bewegt ihn dann 1000 mal mittels `move`. Nach jedem Zug (`move`-Aufruf) soll etwa 5 ms gewartet werden.

b. Testen Sie diese Klasse in einer Java-Anwendung mit zwei Buttons `Start` und `Close` (natürliche Semantik).

Implementieren Sie den Aufruf der Ball-Klasse *ohne* Verwendung von Threads!

Aufgabe 27: (P) Java-Tanz mit Threads (6 Pkt.)

Nun sollen Sie das eben geschriebene Programm so modifizieren, daß beim Aktivieren des `Start`-Buttons ein neuer Thread gestartet wird. Dazu soll die `Ball`-Klasse zu einem Thread erweitert werden.
Sollten Sie nicht wissen, wie Threads in Java programmiert werden, werfen Sie einen Blick auf
`http://www.javasoft.com/docs/books/tutorial/essential/threads/index.html`
Beschreiben Sie die Vorteile durch die Nutzung von Threads.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 3
30.10.2001

Informatik III

Achtung: Bitte beachten Sie, dass zusätzlich zu diesem Übungsblatt auch ein **Fragebogen** zu Praktika im kommenden SS 2002 ausgeteilt wird. Der Fragebogen soll von jedem Studierenden vollständig ausgefüllt bis zum **15. November 2001** entweder im Lehrstuhlsekretariat bei Frau Pötschke (Zimmer 0.50, Oettingenstr. 67) oder direkt in der Vorlesung bei Frau Prof. Dr. Linnhoff-Popien abgegeben werden.
Da diese Woche der Donnerstag (01.11.2001) ein Feiertag ist, bitten wir die Teilnehmer der Donnerstagsübungsgruppen sich auf die übrigen Übungen zu verteilen.

Lesen: Stallings, Kap. 3

Vertiefen: Silberschatz, Kap. 4

Aufgabe 15: (H) Kontextswitch (2+2+3 Pkt.)

Im folgenden sollen Sie sich Gedanken über den Kontextswitch zwischen Prozessen machen:

a. Welche Aktionen muß ein Betriebssystemen bei einem Kontextswitch zwischen Prozessen vornehmen?

b. Wovon hängt der Aufwand für einen Kontextswitch also im wesentlichen ab?

c. Sie werden von der Firma AB Computer angestellt, um die Geschwindigkeit deren Systems zu verbessern. Ihre Anwendungen nutzen nur 10 der 32 Register der CPU; daher wird vorgeschlagen, die Kontextswitching-Routine des Betriebssystems so zu verändern, daß nur die 10 benötigten Register gesichert werden. Nehmen Sie an, daß Sie die Kontextswitching-Routine korrekt ändern können. Ist es dies eine gute oder schleche Idee? Warum?

Aufgabe 16: (H) Prozessormodi und Unterbrechungen (4+3+3 Pkt.)

a. Warum gibt es üblicherweise mehrere Prozessormodi und wie werden diese in typischen Betriebssystemen verwendet?

b. Unter welchen Umständen sind Interrupts effizienter als Polling? Wann wäre Polling effizienter?

c. Warum kann innerhalb von UNIX ein Prozess, der sich gerade im Kernelmodus befindet, nicht unterbrochen werden? In welcher Hinsicht ist UNIX daher für Echtzeitanwendungen ungeeignet.

Aufgabe 17: (H) Wahr oder Falsch? (4 Pkt.)

Entscheiden Sie, ob die folgenden Aussagen falsch oder wahr sind:

a. Compiler laufen im Kernelmodus.

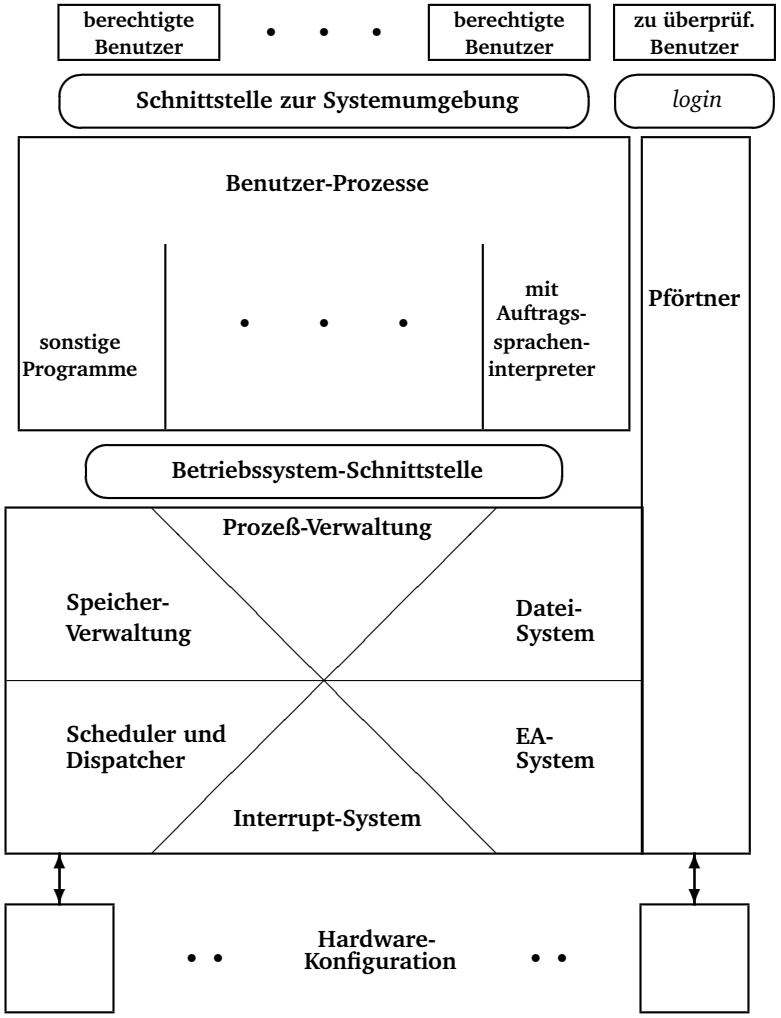
b. Nur Interrupts können die CPU vom Benutzer- in den Kernelmodus schalten.

c. Jeder Interrupt schaltet die CPU vom Benutzer- in den Kernelmodus.

d. In einem Uniprozessor-System kann sich stets nur ein Prozeß im `Running`-Zustand befinden.

Aufgabe 18: (T) Komponenten Betriebssystem (1+4+4 Pkt.)

In der unten angegebenen Abbildung sind noch einmal die wichtigsten Komponenten eines Betriebssystems dargestellt.
Sie sollen nun einen Vergleich der drei Betriebssysteme UNIX, MS-DOS und Windows 95 bezüglich dieser Komponenten anstellen. Geben Sie für die in der Abbildung angegebenen Betriebssystemkomponenten an, in welchen Ausprägungen sie in den Systemen UNIX, MS-DOS und Windows 95 auftreten!



Aufgabe 19: (K) Tamagottochi (3+3 Pkt.)

Das virtuelle Haustier „Tamago*tto*chi“ (entwickelt an der RWTH Aachen) verfügt über verschiedene scheinbar „menschliche“ Züge: Es bekommt Hunger, muß aufs Klo und hat Langeweile. Sein Halter muß jeweils abhelfen, um es glücklich zu machen!
Ihre Aufgabe besteht nun darin, seine Verhaltensweise in einem Diagramm darzustellen. Erörtern Sie dazu, in welchen Zuständen sich unser kleiner Freund jeweils befinden kann, und durch welche Ereignisse diese gewechselt werden.
Zu beachten ist dabei:

- Andauernder Hunger führt zum Tod.
- Andauernder Harndrang oder Langeweile veranlassen Tamagottochi zur Flucht.
- Langeweile kann nur aufkommen, wenn das Tamagottochi weder Hunger hat noch auf die Toilette muß.
- Es ist möglich, daß sich das Tamagottochi gleichzeitig in 2 Teilzuständen befindet. Diese sind dann geeignet zu einem neuen eigenen Zustand zusammenzufassen.

a. Welche sieben Zustände kann das Tamagottochi unter obigen Bedingungen annehmen?

b. Entwerfen Sie ein Zustandsübergangsdiagramm.

Aufgabe 20: (P) Java Native Interface (10 Pkt.)

Diese Aufgabe soll Sie in die Verwendung des *Java Native Interface* (JNI) einführen. Sie sollten sich vor der Lösung dieser Aufgabe mit dem JNI vertraut machen, beispielsweise mit Hilfe der folgenden Links:

`http://java.sun.com/docs/books/tutorial/native1.1/`
„Trail: Java Native Interface“ des Java Tutorial von Javasoft/Sun.

`http://www.javasoft.com/products/jdk/1.1/docs/guide/jni/index.html`
„Java Native Interface“-Site von Javasoft/Sun mit Link zu Tutorial, Spezifikation und FAQ.

Schreiben Sie eine Java-Klasse `HalloweenNative`, die eine einzige Klassenmethode `greeting` besitzt. Diese Methode soll als native Methode implementiert sein und den String `Happy Halloween!` auf der Konsole ausgeben.
Eine Implementierung für diese Methode in C sieht wie folgt aus:

```
#include "HalloweenNative.h"
#include <stdio.h>

JNIEXPORT void JNICALL Java_HalloweenNative_greeting
  (JNIEnv* env, jclass cl)
{  printf("Happy␣Halloween!\n");
}
```

Ihre Aufgabe besteht also nur darin, die entsprechende Java-Klasse zu schreiben, die Header-Datei zu erzeugen (mit Hilfe von `javah`, genaueres in der angegebenen Dokumentation).

Der C-Code wird (zumindest bei Verwendung von Linux) mittels des folgenden Kommandos in eine *shared library* übersetzt:

```
cc -shared -I/usr/java/jdk1.3/include -I/usr/java/jdk1.3/include/linux
           HalloweenNative.c -o libHalloweenNative.so
```

Die beiden Include-Verzeichnisse müssen Sie u.U. anpassen.

Testen Sie schließlich das Ergebnis, indem Sie die Methode `greeting` aus einer anderen Klasse aufrufen. Beachten Sie, daß Java *shared libraries* nur in den in der Umgebungsvariable `LD_LIBRARY_PATH` angegebenen Verzeichnissen sucht.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 2
23.10.2001

Informatik III

Lesen: Stallings, Kap. 3.1-3.3
Vertiefen: Silberschatz, Kap. 4.1-4.4

Aufgabe 8: (H) Multiprogramming (5 Pkt.)

Nennen Sie die wesentlichen Probleme, die bei der Unterstützung von nebenläufigen Prozessen in einem Betriebssystem auftreten können.

Aufgabe 9: (H) Suspend-Zustand (2+4 Pkt.)

a. Definieren Sie den Suspend-Zustand eines Prozesses und begründen Sie die Notwendigkeit eines solchen Zustandes.

b. Überlegen Sie sich verschiedene Gründe, aus denen ein Prozeß in den Suspend-Zustand übergehen kann!

Aufgabe 10: (T) Prozeßzustände I (2+4+4 Pkt.)

Die folgende Tabelle stellt die Prozeßzustände des VAX/VMS-Betriebssystemes dar:

Prozeßzustand	**Beschreibung**
Currently executing, CUR	Prozeß wird ausgeführt (*Running*)
Computable (resident), COM	Bereit und im Hauptspeicher
Computable (outswapped), COMO	Bereit, aber ausgelagert
Page fault wait, PFW	Prozeß hat auf einen Speicherbereich zugegriffen, der sich nicht im Hauptspeicher befindet und muß darauf warten, bis dieser Bereich geladen ist.
Collided page wait, COLPG	Prozeß hat auf einen Speicherbereich zugegriffen, der gerade aus- oder eingelagert wird.
Common event flag wait (resident), CEF	Wartend auf ein shared event flag (i.A. für Interprozesskommunikation verwendet).
Common event flag wait (outswapped), CEFO	analog, ausgelagert
Free page wait, FPG	Wartend bis ein Speicherbereich zu dem aktuellen Speicherbereich des Prozesses hinzugefügt wird.
Hibernate wait (resident), HIB	Prozess versetzt sich selbst in Ruhezustand.
Hibernate wait (outswapped), HIBO	analog, ausgelagert.
Local event flag wait (resident), LEF	Prozess wartet auf ein lokales Ereignis (i.a. E/A-Operation), im Speicher.
Local event flag wait (outswapped), LEFO	analog, ausgelagert.
Suspended wait (resident), SUSP	Prozeß durch anderen Prozeß in Wartezustand.
Suspended wait (outsw.), SUSPO	analog, ausgelagert.
Mutex/Miscellaneous wait, MWAIT	Prozeß wartet auf verschiedene Systemressourcen.

a. Überlegen Sie sich eine Begründung für die Existenz derartig vieler Prozeßzustände!

b. Warum haben einige Zustände keine Unterscheidung in resident und outswapped?

c. Zeichnen Sie das Zustandsübergangsdiagramm und benennen Sie für jeden Zustandsübergang die Aktion oder das Ereignis, das ihn bewirkt.

Aufgabe 11: (T) Betriebsmittel (7+4 Pkt.)

Eine der wesentlichen Aufgaben eines Betriebssystems ist die Verwaltung von Betriebsmitteln.

a. Nennen Sie einige Klassen von Betriebsmitteln!

b. Bei der Betriebsmittelverwaltung sind eine Menge von strategischen Entscheidungen zu treffen. Geben Sie einige Problemkreise an, in denen solche Entscheidungen anfallen, und geben Sie Strategien bzw. Mechanismen zu deren Lösung an.

Aufgabe 12: (K) Zustandsübergänge (1+1+1 Pkt.)

Geben Sie für jeden der folgenden Zustandsübergänge an, ob der Übergang zulässig ist und auf welche Weise der Übergang stattfindet:

a. Ändern des Prozeß-Zustandes von `Blocked` zu `Running`.

b. Ändern des Prozeß-Zustandes von `Running` zu `Blocked`.

c. Ändern des Prozeß-Zustandes von `Ready` zu `Blocked`.

Aufgabe 13: (P) Prozeßinformationen unter UNIX (1+1 Pkt.)

Unter UNIX werden Informationen über die laufenden Prozesse durch das Kommando `ps` ausgegeben.

a. Geben Sie `ps -efaux` ein und protokollieren Sie das Ergebnis. Beachten Sie die Prozeßhierarchie und den Prozeßzustand.

b. Finden Sie die möglichen Werte für den Prozeßzustand und deren Bedeutung heraus und versuchen Sie für jeden Zustand einen Prozeß in den entsprechenden Zustand zu versetzen.

Aufgabe 14: (P) A responsive GUI (4 Pkt.)

Schreiben Sie ein (einfaches!) Java-Programm (mit AWT oder Swing), das zwei Buttons anzeigen soll: Ein Button `Hello` soll den String „`Hello World`“ auf der Konsole ausgeben, ein weiterer Button `Sleep` soll das Java-Programm für einige Zeit (beispielsweise 5 s) mit Hilfe von

```
Thread.currentThread().sleep
```

in den Wartezustand versetzen.
Beobachten Sie, was passiert, wenn Sie während der Wartezeit den `Hello`-Button drücken.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2001/02
Übungsblatt 14
29.01.2002

Informatik III

Achtung:

- Die **Anmeldung** zur **2. Klausur** ist noch bis **Freitag, 01.02.2002 12 Uhr** möglich.
  Die Anmeldung ist **obligatorisch** und Nachmeldungen werden **nicht akzeptiert!**
  Für die 2.Klausur ist anzumerken, dass der **gesamte** Stoff der Vorlesung und Übung klausurrelevant ist. Beachten Sie bitte auch, dass das erfolgreiche Vorrechnen mindestens einer Übungsaufgabe **Zulassungsvoraussetzung** ist.
- Ab **Mitte nächster Woche** läuft die **Anmeldung zu den Praktika** für das **SS 2002**. Die Anmeldung ist **obligatorisch** und nur bis **Mittwoch, 20.02.2002** möglich! Auf der Lehrstuhl–Homepage wird ein Link zur entsprechenden Anmeldungsseite eingerichtet:

  `http://www.nm.informatik.uni-muenchen.de/`

Wiederholen: Silberschatz, Stallings, Tannenbaum

Aufgabe 72: (H) Wiederholung

(3+1+1+8+4+4+4+6+6 Pkt.)

Die folgenden Aufgaben stammen zum größten Teil aus Klausuren der Central Queensland University, Australien:

a. Ergänzen Sie folgenden Lückentext:

   1) Die zwei gültigen Opertionen auf einer Semaphore sind ..... und .....
   2) Zwei Beispiele für das Management von virtuellem Speicher sind ..... und .....
   3) Ein geläufiges Programmierkonstrukt zur Implementation von wechselseitigem Ausschluß unter Nutzung von Bedingungsvariablen wird als .....

b. Der SJF-Algorithmus ist

   1) ein präemptiver CPU-Schedulingalgorithmus.
   2) ein Disk-Schedulingalgorithmus.
   3) eine Modifikation des Round Robin CPU-Schedulingalgorithmus.
   4) ein nicht-präemptiver CPU-Schedulingalgorithmus.

c. Welche der folgenden Prozeßzustandsübergänge sind ungültig?

   1) ready $\rightarrow$ running.
   2) ready $\rightarrow$ blocked.
   3) blocked $\rightarrow$ ready.
   4) running $\rightarrow$ dead.

d. Erklären bzw. beantworten Sie kurz die folgenden Fragen:

(1) Multiprogramming.
   - Warum wird es verwendet?
   - Welche Probleme entstehen durch seine Verwendung?

(2) Semaphoren.
   - Was sind Semaphoren?
   - Wozu dienen Sie?
   - Welche Operationen auf ihnen gibt es und welche Aufgabe haben diese?

(3) Fassen Sie die Argumente zusammen für und gegen kleine bzw. große *page size*.

(4) Starvation.
   - Was versteht man darunter?
   - Wodurch entsteht es?
   - Wie kann ein Betriebssystem/Algorithmus es vermeiden?
   - Welcher andere Ausdruck wird dafür auch verwendet?

e. Was ist ein PCB? Wozu dient er? Welche Informationen werden dort gespeichert? Wozu dienen diese Informationen? Welche Operationen können auf ihm ausgeführt werden? Erläutern Sie ausführlich!

f. Angenommen zwei Prozesse A und B enthalten beide die folgenden Befehle in einer Schleife:

```
int Numres = 0;
Numres = Numres + 1;
if Numres == 50 then Numres = 0;
```

Nehmen Sie weiterhin an, daß beide die Variable `Numres` teilen und beide aktiv sind. Erklären Sie, wie selbst bei nur einer CPU es eintreten kann, daß `Numres` nie auf 0 zurückgesetzt wird.

g. Beschreiben Sie die Ideen und Konzepte im Umfeld von Monitoren.

h. Betrachten Sie eine Variante des Round Robin-Schedulingalgorithmus, in der die Einträge in der Warteschlange Zeiger auf die PCBs sind.

(i) What wäre der Effekt, wenn man zwei Zeiger auf den selben Prozeß in die Warteschlange einfügt.

(ii) Wie würden Sie den ursprünglichen Round Robin-Algorithmus verändern, um den selben Effekt ohne doppelte Zeiger zu erhalten.

Aufgabe 73: (H) Java-Certification Exam (10 Pkt.)

Die folgenden Aufgaben sind Übungsaufgaben zur Vorbereitung der Java-Certification Prüfung:

a. Was wird passieren, wenn Sie versuchen den folgenden Code zu kompilieren und auszuführen?

```
abstract class Base{
        abstract public void myfunc();
        public void another(){
        System.out.println("Another method");
        }
}

public class Abs extends Base{
```

```
        public static void main(String argv[]){
10      Abs a = new Abs();
        a.amethod();
12      }
        public void myfunc(){
14              System.out.println("My Func");
                }
16      public void amethod(){
        myfunc();
18      }
}
```

1) Der Code wird kompiliert und ausgeführt, wobei „My Func“ ausgedruckt wird.
2) Der Compiler bemängelt, daß die `Base`-Klasse keine abstrakte Methode hat.
3) Der Code wird kompiliert, aber es tritt ein Laufzeitfehler auf: Die `Base`-Klasse hat keine abstrakten Methoden.
4) Der Compiler wird bemängeln, daß die Method `myfunc` in der `Base`-Klasse keinen Funktionskörper besitzt.

b. Welche Gründe gibt es, eine Methode als *native* zu definieren:

1) Um Zugriff auf Hardware zu bekommen, die von Java nicht unterstützt wird.
2) Um einen neuen Datentyp wie z.B. `unsigned integer` zu definieren.
3) Um optimierten Code zu schreiben.
4) Um die Beschränkung des `private` Gültigkeitsbereichs einer Methode aufzuheben.

c. Sie wollen den Wert des letzten Elements eines Arrays mit dem folgenden Code herausfinden. Was passiert, wenn Sie ihn kompilieren und ausführen?

```
public class MyAr{
2 public static void main(String argv[]){
        int[] i = new int[5];
4       System.out.println(i[5]);
        }
6 }
```

d. Was wird passieren, wenn Sie versuchen den folgenden Code zu kompilieren und auszuführen?

```
public class Bground extends Thread{
2 public static void main(String argv[]){
                Bground b = new Bground();
4               b.run();
        }
6       public void start(){
                for (int i = 0; i <10; i++){
8                       System.out.println("Value of i = " + i);
                }
10      }
}
```

e. Was kann einen Thread dazu bringen, die Ausführung zu unterbrechen?

1) Das Programm beendet sich mit einem Aufruf von `System.exit(0);`.
2) Einem anderen Thread wird höhere Priorität gegeben.

3) Ein Aufruf der `stop`-Methode des Threads.

4) Ein Aufruf der `halt`-Methode des Threads.

f. Welche der folgenden Aussagen über Threads sind wahr?

1) Man kann einen wechselseitigen, exklusiven Lock für Methoden in einer Klasse erhalten, die die `Thread`-Klasse erweitert oder das Interface `runnable` implementiert.

2) Man kann einen wechselseitigen, exklusiven Lock für jedes Objekt erhalten.

3) Ein Thread kann einen wechselseitigen, exclusiven Lock für eine `synchronized` Method eines Objekts erhalten.

4) Thread–Scheduling Algorithmen sind plattformabhängig.

g. Welche der folgenden Aussagen beschreibt am besten die Funktionsweise des `synchronized`-Schlüsselwortes?

1) Erlaubt zwei Prozessen parallel ausgeführt zu werden, aber miteinander zu kommunizieren.

2) Stellt sicher, daß nur ein Thread zur selben Zeit auf eine Methode oder ein Objekt zugreifen kann.

3) Stellt sicher, daß zwei oder mehr Prozesse zur selben Zeit starten und enden werden.

4) Stellt sicher, daß zwei oder mehr Threads zur selben Zeit starten und enden werden.

Aufgabe 74: (P) Java-Threads again (5 Pkt.)

Auf der Info3-Homepage ist ein Applet verfügbar, das für eingegebene Zahlen prüft, ob diese Primzahlen sind:
`http://www.nm.informatik.uni-muenchen.de/Vorlesungen/info3.shtml`

Da diese Prüfung bei größeren Zahlen u.U. länger dauert und die Oberfläche in dieser Zeit nicht blockiert werden soll, soll für die Berechnung ein Thread gestartet werden.

a. Fügen Sie in der Klasse PrimeCalculator eine neue innere Klasse hinzu, die das `Runnable`-Interface implementiert und die zeitraubenden Berechnungen aus `actionPerformed` durchführt.

Aufgabe 75: (P) Moving Threads (10 Pkt.)

Auf der Info3-Homepage wird ein Applet zur Verfügung gestellt, das einen Ball über zwei Buttons bewegen kann.
Ändern Sie das Applet so, daß sich der Ball nach einmaligen Betätigen eines Richtungsknopfes so lange in die angegebene Richtung bewegt, bis er durch Druck auf den anderen Button wieder angehalten wird.
Fügen Sie hierzu beispielsweise innerhalb der Klasse BallCanvas eine innere Klasse hinzu, die die Klasse Thread erweitert und in Sekundenintervallen die Position des Balles anhand seiner Geschwindigkeit anpaßt. Wie können Sie aus der Klasse MoveBall heraus die Thread-Methoden `suspend` und `resume` zur Implementation der Applet-Methoden `stop` und `start` aufrufen? Sie können die `suspend`- und `resume`-Methoden der Thread-Klasse verwenden (auch wenn diese *deprecated* sind), müssen also keine eigene Implementation angeben.