Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, G. Bauer,
Dr. M. Pizka, Dr. B. Rumpe,
Dr. R. Steinbrüggen

WS 2002/2003
Lösungsvorschläge zu Blatt 8
13. Dezember 2002

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 29 Dining Philosophers

(a) Eine Lösung mit Monitoren, die Verklemmung vermeidet:

```
monitor Dining_Philosophers =
  ⌈ var state array [0:4] state: = [thinking, thinking, ..., thinking];
    signal array [0:4] self ;

    proc take_forks = (int i):
      ⌈ state[i] := hungry;
        test(i);
        if state[i] ≠ eating then self[i].wait; ⌋

    proc put_forks = (int i):
      ⌈ state[i] := thinking;
        test(i-1 mod 5);
        test(i+1 mod 5);  ⌋

    proc test = (int i):
      ⌈ if (state[i] = hungry
             ∧ state[i-1 mod 5] ≠ eating
             ∧ state[i+1 mod 5] ≠ eating)
        then state[i] := eating; self[i].signal) fi  ⌋
  ⌋

proc Philosopher (int i) =
⌈ while true do
    think();
    take_forks(i);
    eat();
    put_forks(i);
  od  ⌋
```

(b) Eine Lösung mit Monitoren und einer Warteschlange für die wartenden Prozesse, die Verklemmung und Verhungerung vermeidet:

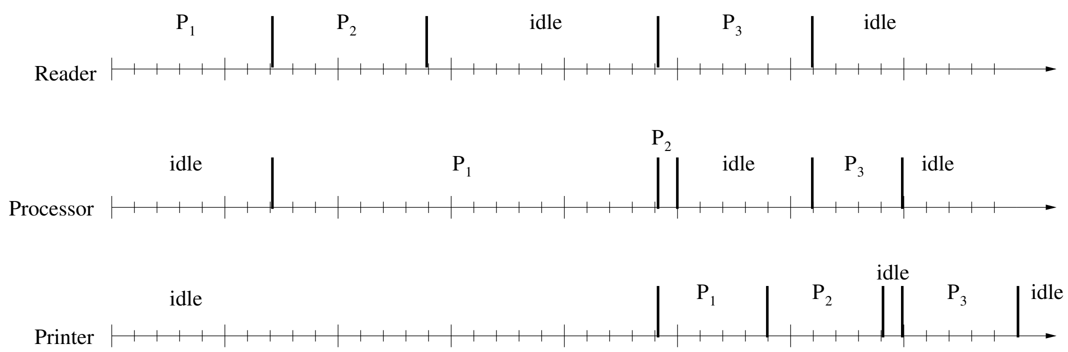
```
monitor Dining_Philosophers =
  ⌈ var state array [0:4] state: = [thinking, thinking, ..., thinking];
```

```
    signal array [0:4] self ;
    var queue nat q = emptyqueue;

    proc take_forks = (int i):
      ⌈ state[i] := hungry;
        q := stock(q, i);
        test(first(q));
        if state[i] ≠ eating then self[i].wait; ⌋

    proc put_forks = (int i):
      ⌈ state[i] := thinking;
        if ¬ isempty(q) then test(first(q));  ⌋

    proc test = (int i):
      ⌈ if (state[i] = hungry
             ∧ state[i-1 mod 5] ≠ eating
             ∧ state[i+1 mod 5] ≠ eating)
        then state[i] := eating; q := rest(q); self[i].signal) fi  ⌋
  ⌋

proc Philosopher (int i) =
⌈ while true do
    think();
    take_forks(i);
    eat();
    put_forks(i);
  od  ⌋
```

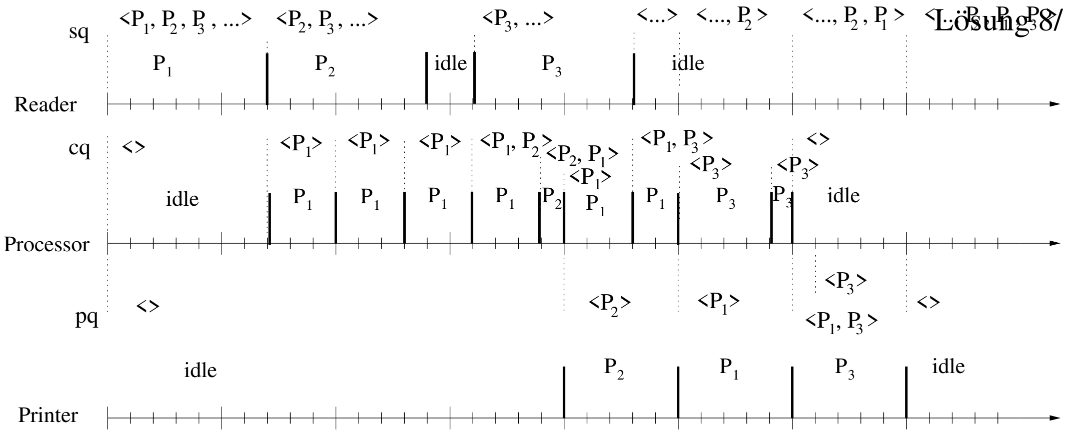
Nachteil bei der zweiten Variante: Falls ein Prozess wartet, müssen alle anderen auch warten. Dadurch werden parallele Abläufe ausgeschlossen, die bei der ersten Variante möglich waren. Aber die Lösung ist immer noch besser als streng sequentielle Ausführung der Prozesse, da im günstigsten Fall auch zwei Philosophen essen können.

Aufgabe 30 Scheduling im Multiplexbetrieb

a)



b)



Aufgabe 31 (P) *n* Leser-*m* Schreiber in Java

a)
```
// Wir gehen davon aus, dass sowohl Leser als auch Schreiber in der
// Klasse "LS" implementiert sind. LS soll hierzu eine Methode "reader"
// und eine Methode "writer" besitzen. Die Entscheidung, ob sich ein
// instanziertes Objekt wie ein Leser oder ein Schreiber verhält wird in
// der Methode "run" getroffen. Die Klassen "Leser" und "Schreiber" können
// demnach als Unterklassen von "LS", mit entsprechend spezialisierten
// "run" Methoden verstanden werden.

// Beispiel für await Variante !!!

    static public void main(String args[]) throws InterruptedException {
        LS reader[] = new LS[10];
        GemVar    g = new GemVar(0); // Instanzierung der gemeinsamen Va;
        LSAwait   m = new LSAwait();

        for (int i = 0; i < reader.length; i++) {
            reader[i] = new LS(g,m,"Reader"+i)
                            // anonyme Unterklasse Leser
                            { public void run() { readerAwait();}};
            reader[i].start();
        }
        new LS(g,m,"Writer") { public void run() { writerAwait(); }}.start();
    }

// Hilfs-Klasse für die gemeinsame Programmvariable

class GemVar {
    private int x = 0;

    GemVar() { x = 0; }
    GemVar(int x) { this.x = x; }

    public void writeVar(int i) { x = i; };
    public int  readVar (int i) { return x; };
}
```

b)
```
// Java bietet lediglich Monitore an, deshalb muss auch das await Konstrukt
// mit einem Monitor realisiert werden
```

```
class LSAwait {
    private int numReaders = 0;
    private int numWriters = 0;

    public synchronized void enterReader() {
        while (numWriters != 0)
            try {wait();} catch (InterruptedException e) {}
        numReaders += 1;
    }

    public synchronized void exitReader() {
        numReaders -= 1;
        notifyAll();            // All: Schreiber sollen Nachricht erhalten
    }

    public synchronized void enterWriter() {
        while (numReaders+numWriters != 0)
            try {wait();} catch (InterruptedException e) {}
        numWriters += 1;
    }

    public synchronized void exitWriter() {
        numWriters -= 1;
        notifyAll();
    }
}

// ----------------------  ----------------- -----------------
// ----------------------  ab hier Programmieraufgabe --------
// ----------------------  ----------------- -----------------

class LS extends Thread {
    private GemVar x;
    LSAwait awaitMonitor;

    LS(GemVar x, LSAwait awaitMonitor, String name) {
        super(name);            // Name an Thread-Konstruktor weiterreichen
        this.x = x;
        this.awaitMonitor = awaitMonitor;
    }

    public void reader() {
        int v = -1, previousV = -1;
        while (v < 100) {
            awaitMonitor.enterReader();
            v = x.readVar();
            awaitMonitor.exitReader();

            // nur Ausgeben, falls nicht identisch mit letztem Wert
            if (v != previousV) {
```

```
                previousV = v;
                System.out.println(getName() +": " + v);
            }
        }
    }

    public void writer()  {
        while (x.readVar() < 100) {
            awaitMonitor.enterWriter();
            x.writeVar(x.readVar()+1);
            awaitMonitor.exitWriter();
            try { sleep(1); } catch(InterruptedException e) {}
        }
    }
}
```

c)

```
public class SemaNat {
    private int sema;           // Semaphor Wert
    String name;                // Semaphor Name

    // Initialisierung eines Semaphores
    public SemaNat(String name, int start)  {
      this.sema=start;
      this.name=name;
    }

    public synchronized void P()
    { // System.out.println("P("+name+") bei Wert "+sema);
      while (sema == 0)  // Warten bis sema>0
          { try { wait(999); } catch (InterruptedException e) {}; }
      // System.out.println("P("+name+") ausgefuehrt");
      sema--;
    }

    public synchronized void V()    {
      sema++;
      notify();          // andere Threads benachrichtigen
    }
}

class LS extends Thread {
    private GemVar x;

    SemaNat mutex, ws;
    GemVar rs;
    LS(GemVar x, SemaNat mutex, SemaNat ws, GemVar rs, String name) {
        super(name);
        this.x = x;
        this.mutex = mutex;
        this.ws = ws;
        this.rs = rs;
```

```
    }

    public void reader() {
        int v = -1, previousV = -1;
        while (v < 100) {
            mutex.P();
            rs.writeVar(rs.readVar()+1);
            if (rs.readVar() == 1)
                ws.P();
            mutex.V();

            v = x.readVar();
            if (v != previousV) {
                previousV = v;
                System.out.println(getName() +": " + v);
            }

            mutex.P();
            rs.writeVar(rs.readVar()-1);
            if (rs.readVar() == 0)
                ws.V();
            mutex.V();
        }
    }

    public void writer()  {
        while (x.readVar() < 100) {
            ws.P();
            x.writeVar(x.readVar()+1);
            ws.V();
            try { sleep(1); } catch(InterruptedException e) {}
        }
    }
}
```