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 6
5. Dezember 2002

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 20 Petri-Netz, Semaphore

(a) Jede Stelle, ausgenommen die Eingangsstelle, entspricht hier einem Booleschen Semaphor. Jede Kante, die von einer Stelle wegführt, entspricht einer P-Operation auf dem Semaphor; jede Kante, die zu einer Stelle hinführt, entspricht einer V-Operation. Für die Zuweisungen im Programm ist aus dem Petri-Netz ablesbar, welche Semaphore vorher reserviert und welche Semaphore anschließend frei gegeben werden müssen.

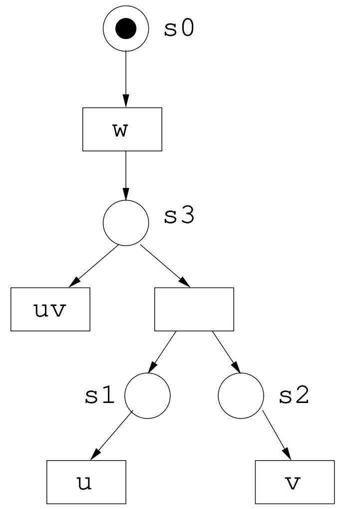
(b)
```
⌈var int x,y,z := 0,0,0;
 sema bool sx_inc,sy_dec := true,true;
 sema bool sx,sy := false,false;

⟦ while true do P(sx_inc); x:=x+1; V(sx) od
‖ while true do P(sy_dec); y:=y-1; V(sy) od
‖ while true do P(sx); P(sy); z:=x+y; V(sx_inc); V(sy_dec) od ⟧
⌋
```

Aufgabe 21 (H) Petri-Netz, Semaphore

a) Die Kanten `(s1,uv)` und `(s2,uv)` können nicht direkt in P-Operationen umgesetzt werden. Beide P-Operationen müssten gemeinsam ausgeführt werden, da es sonst zu einer Verklemmung käme, Beispiel: `P(s2);v` wird direkt nach dem `P(s1)`, aber noch vor dem `P(s2)` aus der Anweisung `P(s1);P(s2);uv` ausgeführt.

b) Damit wie im Petri-Netz sowohl uv wie alternativ u und v an die Reihe kommen, modifizieren wir das Petri-Netz nebenstehend durch einen zusätzlichen Semaphor `s3` und eine Stelle mit der leeren Transition.

c) Das Programm mit entsprechender Synchronisierung:



⌈ **sema bool** s0,s1,s2,s3:=**true,false,false,false;**
⟦ P(s0) ;w();V(s3)
‖ P(s3);uv()
‖ P(s3);V(s1);V(s2)
‖ P(s1);u()
‖P(s2);v() ⟧ ⌋

Aufgabe 22 Prozesssynchronisation

$P||C$ lässt sich einfach in ein Programm übertragen. Nun muss aber noch die Prozesssynchronistion der Aktionen w und r mit dem Puffer realisiert werden. Dazu muessen die folgenden Bedingungen erfüllt seim:

(i) w und r dürfen nicht gleichzeitig ausgeführt werden.

(ii) w und r alternieren.

(iii) w beginnt.

Bei der Realisierung nehmen wir an, dass die Prozeduren `write(x)` und `read(y)` auf einer gemeinsamen Variablen (Puffer) arbeiten, ohne den Zugriff auf diese Variable zu schützen.

(a) (i) `write(x)` und `read(x)` dürfen nur innerhalb eines kritischen Bereichs, d.h. im Rumpf einer **await** Anweisung ausgeführt werden.

(ii) Es wird eine Boolesche Variable `w_enable` eingeführt. die angibt, ob gerade geschrieben werden darf. Diese eine Variable ist ausreichend, denn `read(x)` darf genau dann ausgeführt werden, wenn `write(x)` nicht ausgeführt werden darf.

(iii) Vorbesetzung: `w_enable := true;`

Das ergibt das folgende Programm:

```
⌈ var x, y;
  var bool w_enable : = true;
  ⟦ while true do
      produce(x);
      await w_enable then write(x); w_enable := false endwait
    od
  ||
    while true do
      await ¬ w_enable then read(x); w_enable := true endwait
      consume(x);
    od
  ⟧
⌋
```

(b) (i) Zum gegenseitigen Ausschluss von `write(x)` und `read(x)` wird ein Semaphor `buffer` eingeführt.

(ii) Es werden Boolesche Semaphore `ws` und `rs` eingeführt, die angeben, ob als nächstes geschrieben oder gelesen werden darf.

(iii) Vorbesetzung: `ws := true; rs := false`

Das ergibt das folgende Programm:

```
⌈ var x, y;
  sema bool buffer, ws, rs : = true, true, false;
  ⟦ while true do
       produce(x);
       P(ws); P(buffer); write(x); V(buffer); V(rs)
    od
  ||
    while true do
       P(rs); P(buffer); read(x); V(buffer); V(ws)
       consume(x);
    od
  ⟧
⌋
```

Man beachte, dass die Reihenfolge von `P(ws); P(buffer)` bzw `P(rs); P(buffer)` nicht vertauscht werden darf.

Vereinfachung: das Semaphor `buffer` ist überflüssig, da bereits `ws` und `rs` den gegenseitigen Ausschlusss sicherstellen: Der Erzeuger kann `P(ws)` nur am Anfang erfolgreic ausführen, oder wenn der Verbraucher gerade `V(ws)` ausgeführt hat. In beiden Fällen befindet sich der Verbraucher gerade nicht in seinem kritischen Bereich und kann wegen `P(rs)` auch nicht in ihn gelangen. Umgekehrt: der Verbraucher kann `P(rs)` nur erfolgreich ausführen, wenn der Erezuger `V(rs)` ausgeführt hat; dann kann sich der Erzeuger nicht in seinem kritischen Bereich befinden und ihn wegen `P(ws)` auch nicht neu betreten. Entsprechende Programmteile können also noch weggelassen werden.

Aufgabe 23 (P) Parallele Matrixmultiplikation

```
class Matrix {
    private int m_rows;
    private int m_columns;
    private int[][] m_array;

    Matrix(int rows,int columns, int init)  {
        m_rows=rows;
        m_columns = columns;
        m_array = new int[m_rows][m_columns];
        for(int i=0; i < rows; i++)
            for(int j=0; j < columns; j++)
                SetAt(i,j,init);
    }
```

```
    int GetAt(int row,int column)   {
        int retvalue=-1;
        if(row>=0 && row<m_rows && column>=0 && column<m_columns)
            retvalue = m_array[row][column];
        return retvalue;
    }

    void SetAt(int row,int column,int value)  {
        if(row>=0 && row<m_rows && column>=0 && column<m_columns)
            m_array[row][column] = value;
    }

    int GetRows()  {
        return m_rows;
    }

    int GetColumns()  {
        return m_columns;
    }

    public String toString() {
        String s = "";
        for(int i=0; i<m_rows; i++)
        {
            for(int j=0; j<m_columns; j++)
                s += GetAt(i,j) + "\t";
            s += "\n";
        }
        return s;
    }
}


class RowColumnMult extends Thread {
    private Matrix in1,in2,out;
    private int row,column;

    RowColumnMult(Matrix A, Matrix B, Matrix C, int i, int j) {
      in1 = A;
      in2 = B;
      out = C;
      row = i;
      column = j;
    }

    public void run() {
      int innerproduct = 0;
      for(int k=0; k<in1.GetColumns(); k++)
            innerproduct += in1.GetAt(row,k) * in2.GetAt(k,column);
      out.SetAt(row,column,innerproduct);
```

```
    }

    static public void main(String args[]) {
      Matrix A = new Matrix(2,4,2);
      Matrix B = new Matrix(4,2,5);
      Matrix C = new Matrix(A.GetRows(),B.GetColumns(),0);
      RowColumnMult Workers[][] = new RowColumnMult[C.GetRows()][C.GetColumns()];

      for(int i=0; i<C.GetRows(); i++)
            for(int j=0; j<C.GetColumns(); j++) {
                Workers[i][j] = new RowColumnMult(A,B,C,i,j);
                Workers[i][j].start();
            }

      for(int i=0; i<C.GetRows(); i++)
            for(int j=0; j<C.GetColumns(); j++)
            try {
                Workers[i][j].join();
            } catch (InterruptedException e) {};

      System.out.println("A:\n"+A+"B:\n"+B+"C:\n"+C);
    }
}
```