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 9
20. Dezember 2002

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 32 E/A-Kanal-Endemeldung

In Betriebssystemen kann die Prozessorzuteilung auch durch Signale von periphären Geräten gesteuert werden. Diese Signale bewirken *Unterbrechungen* der Ausführung von Benutzerprogrammen.

a) Wir stellen das für den Zeitscheibenbetrieb angegebene Betriebssystem auf den Unterbrechungsbetrieb um, so dass E/A-Kanal-Endemeldungen als Unterbrechungen behandelt werden.

Die Prozeduren input_channel und output_chanal signalisieren das Ende ihrer Aktivität über die globale Variable cr_active bzw cp_active. In der Prozedur processor sehen wir zwei zusätzliche Variablen vor:

var bool hr, hp := **true**, **false**;

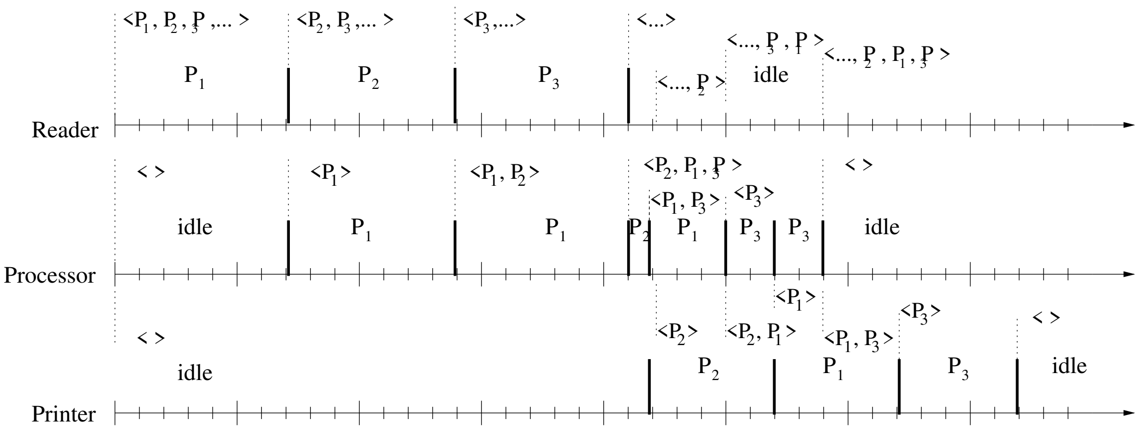
und modifizieren die *Programmbearbeitung* wie folgt.

if ¬isemptyqueue(cp)
then **while** (hr ∨ isemptyqueue(sq)) ∧ (hp ∨ isemptyqueue(pq)) **do**
„Rechne nächsten Schritt des Programms in first(cq),
setze status auf print oder compute“;
await true then hp, hr := cp_active, cr_active **endwait**
od
fi

b) Wir untersuchen das unten aufgeführten Beispiel und geben für jeden der drei Prozesse reader, printer und processor in Form eines Zeitstrahles an, wann er mit welchem Programm aktiv bzw. wann er idle ist. Die Ausführungszeit der Betriebssystemprozeduren ist dabei vernachlässigbar klein. Wir notieren, welche Programme jeweils in den Warteschlangen sq, pq und cq gehalten werden.

Das Beispiel (auf dem vorigen Lösungsblatt ist der Zeitscheibenbetrieb behandelt):

- Im Externspeicher liegen drei Programmstapel P_1, P_2 und P_3;
- das Einlesen eines Programmstapels dauert jeweils 7 Zeiteinheiten;
- das Rechnen der Programme P_1, P_2, P_3 dauert 17, bzw. 1, bzw. 4 Zeiteinheiten;
- der Ausdruck eines Programmergebnisses dauert jeweils 5 Zeiteinheiten.



Aufgabe 33 Bankier-Algorithmus

Auf einem Betriebsystem, das den *Bankier-Algorithmus* zur Vermeidung von Verklemmungen bei der Betriebsmittelvergabe einsetzt, existieren 5 Betriebsmittelklassen. Es seien 4 Prozesse im System, deren maximale Anforderungen gegeben sind durch die Matrix

$$Max = \begin{pmatrix} 3 & 5 & 8 & 10 & 1 \\ 2 & 5 & 3 & 3 & 2 \\ 4 & 12 & 4 & 9 & 2 \\ 6 & 1 & 4 & 5 & 5 \end{pmatrix}.$$

Die vorhandenen Betriebsmittel werden beschrieben durch den Vektor

$$Recources = (6, 15, 8, 10, 9).$$

(a) Der durch die Matrix

$$Allocation = \begin{pmatrix} 0 & 2 & 1 & 1 & 1 \\ 0 & 5 & 3 & 1 & 1 \\ 0 & 7 & 1 & 2 & 1 \\ 3 & 1 & 1 & 1 & 0 \end{pmatrix}$$

gegebene Zustand ist sicher, denn alle Maximalanforderungen aller Prozesse können in einer bestimmten Reihenfolge erfüllt werden:

Dazu wird zuerst die Matrix *Need* der maximal noch ausstehenden Anforderungen berechnet

$$Need = Max - Allocation = \begin{pmatrix} 3 & 3 & 7 & 9 & 0 \\ 2 & 0 & 0 & 2 & 1 \\ 4 & 5 & 3 & 7 & 1 \\ 3 & 0 & 3 & 4 & 5 \end{pmatrix}$$

Ausserdem wird der Vektor *Available* der noch verfügbaren Betriebsmittel berechnet:

$$Available = (3, 0, 2, 5, 6)$$

Zuerst können die Anforderungen von Prozess 2 erfüllt werden. Darauf gibt Prozess 2 alle seine Betriebsmittel zurück.

$$Available = (3,5,5,6,7)$$

Dann können die Anforderungn von Prozess 4 erfüllt werden. Darauf gibt Prozess 4 alle seine Betriebsmittel zurück.

$$Available = (6,6,6,7,7)$$

Dann können die Anforderungn von Prozess 3 erfüllt werden. Darauf gibt Prozess 3 alle seine Betriebsmittel zurück.

$$Available = (6,13,7,9,8)$$

Schliesslich können die Anforderungen von Prozess 1 erfülllt werden. Darauf gibt Prozess 1 alle seine Betriebsmittel zurück.

$$Available = (6,15,8,10,9)$$

(b) Das System befinde sich in dem Zustand

$$Allocation = \begin{pmatrix} 0 & 1 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 & 2 \\ 2 & 5 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 \end{pmatrix}$$

Dieser Zustand ist sicher (nachprüfen!).

Wenn nun Prozess 3 eine Instanz des 3. Betriebsmittels anfordert ergibt sich der folgende Zustand

$$Allocation = \begin{pmatrix} 0 & 1 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 & 2 \\ 2 & 5 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 \end{pmatrix}$$

$$Need = Max - Allocation = \begin{pmatrix} 3 & 4 & 5 & 9 & 0 \\ 2 & 2 & 2 & 2 & 0 \\ 2 & 7 & 2 & 9 & 1 \\ 5 & 1 & 3 & 5 & 3 \end{pmatrix}$$

$$Available = (3,6,1,8,3)$$

Dieser Zustand ist nicht mehr sicher: Es kann keiner der maxiamalen Anforderungen mehr erfüllt werden. Das Betriebssystem darf die Anforderung nicht erfüllen.

(c) Angenommen, das Betriebssystem vergäbe in dem Zustand $Allocation$ aus Aufgabe (b) auf Anforderung dem Prozess 4 eine weitere Instanz des 4. Betriebsmittels.

Es ergibt sich der Zustand

$$Allocation = \begin{pmatrix} 0 & 1 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 & 2 \\ 2 & 5 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 2 \end{pmatrix}$$

$$Need = Max - Allocation = \begin{pmatrix} 3 & 4 & 5 & 9 & 0 \\ 2 & 2 & 2 & 2 & 0 \\ 2 & 7 & 3 & 9 & 1 \\ 5 & 1 & 3 & 4 & 3 \end{pmatrix}$$

$$Available = (3,6,2,7,3)$$

Die maximalen Anfoderungen von Prozess 2 können erfüllt werden. Darauf gibt Prozess 2 alle seine Betriebsmittel zurück.

$$Available = (3,9,3,8,5)$$

Die maximalen Anforderungen der Prozesse 1, 3, und 4 können nicht mehr erfüllt werden. Der Zustand ist nicht sicher.

Es kann es zu einem Deadlock kommen, wenn alle Prozesse ihre maximalen Anforderungen an Betriebsmitteln stellen. Es muss aber nicht zu einer Verklemmung führen, wenn die maximalen Anforderungen nicht voll ausgeschöpft werden.

Aufgabe 34 Bankier-Algorithmus in Java

a)
```
public class Prozess extends java.lang.Thread {

    private int[] bMittel_max = null;   // maximale BM-Forderungen
    private int[] bMittel_cur = null;   // aktulle BM-Belegung
    private int id;
    private Bankier bank;               // Referenz auf den Bankier

    public Prozess(int id, Bankier bank, int[] bm) {
        this.id = id;
        this.bank = bank;
        bMittel_max = bm;
        bMittel_cur = new int[bm.length];
        for (int i=0; i < bm.length; i++) {
            bMittel_cur[i] = 0;
        }
    }

    public void run() {
        boolean finished = false;

        while(!finished) {
            // Selektiere BM zufällig
            int bmNum = (int)(Math.random() * bMittel_max.length);
            while ((bMittel_max[bmNum] - bMittel_cur[bmNum]) == 0) {
                bmNum = (int)(Math.random() * bMittel_max.length);
            }

            // Selektiere BM Menge zufällig
```

```
                int bmMenge = ((int)(Math.random() *
                                (bMittel_max[bmNum] - bMittel_cur[bmNum] - 1)))+1;
                // BM anfordern
                bank.belegen(id, bmNum, bmMenge);
                bMittel_cur[bmNum] += bmMenge;

                // Alle BM angefordert?
                finished = true;
                for (int i=0; (i < bMittel_max.length) && (finished); i++) {
                    if (bMittel_cur[i] < bMittel_max[i]) {
                        finished = false;
                    }
                }

                try { sleep(1); } catch (InterruptedException iex) {}

            }

            // Am Ende alle BM freigeben
            for (int i=0; i < bMittel_max.length; i++) {
                bank.freigeben(id, i, bMittel_cur[i]);
                bMittel_cur[i] = 0;
            }
        }
    }
```

– P –

b)
```
public class Bankier {

        private int[] Resources;
        // Matrix, der maximalen BM-Forderungen der Prozesse
        private int[][] maxAnford;
        // Matrix, der aktuellen BM-Belegung pro Prozess
        private int[][] curProzess;

        public Bankier(int[][] maxAnford, int[] Resources) {
            this.maxAnford = maxAnford;
            this.Resources = Resources;
            // erzeuge curProzess Matrix
            curProzess = new int[maxAnford.length][];
            for (int i=0; i < maxAnford.length; i++) {         // init mit 0
                curProzess[i] = new int[maxAnford[i].length];
                for (int j=0; j < maxAnford[i].length; j++)
                    curProzess[i][j] = 0;
            }
            printBelegung();
        }

        public synchronized void freigeben(int p, int Klasse, int Menge) {
            System.out.println("freigeben("+p+","+Klasse+","+Menge+")");
            curProzess[p][Klasse] -= Menge;
```

```
    Resources[Klasse] += Menge;
    notifyAll();
}

public synchronized void belegen(int p, int Klasse, int Menge) {
    boolean safe = false;
    boolean lebendigeProzesse[] = new boolean[maxAnford.length];
    for (int i = 0; i<lebendigeProzesse.length; i++)
        lebendigeProzesse[i] = true;

    while (!safe) {
        // probeweise zuteilen
        System.out.print("belegen("+p+","+Klasse+","+Menge+")");
        curProzess[p][Klasse] += Menge;
        Resources[Klasse] -= Menge;

        safe = sicher(lebendigeProzesse);

        if (!safe) {    // Zuteilung noch nicht durchführen
            System.out.println(" ---> muss warten.");
            curProzess[p][Klasse] -= Menge;
            Resources[Klasse] += Menge;
            try { wait(); } catch(Exception ex) {}
        }
        else {
            System.out.println();
            printBelegung();
        }
    }
}

private boolean sicher(boolean isAliveProzess[]) {
    int p;
    boolean anyAlive = false, safe = false;

    // 1. Abbruchkriterium der Rekursion: gibt es noch lebendige Prozesse?
    p = 0;
    while (!anyAlive && (p < isAliveProzess.length))
        anyAlive = isAliveProzess[p++];
    if (!anyAlive)
        return true;
    // 2. finde einen vollständig bedienbaren Prozess
    for (p = 0; (p < maxAnford.length) && (!safe); p++)
        if (isAliveProzess[p]) {
            safe = true;
            for (int r = 0; (r < maxAnford[p].length) && safe; r++)
                if (Resources[r] < (maxAnford[p][r] - curProzess[p][r]))
                    safe = false;
        }
    // 3. falls so ein Prozess existiert: pruefe ob die Stellung wieder
    //    sicher ist, wenn dieser Prozess seine Ressourcen freigibt
```

```
        if (safe) {
            p = p-1;  // Korrektur von p nach Ende der for-Schleife oben
            // probeweise: alle Ressourcen dieses Prozesses zurückgeben
            int tempRes[] = new int[maxAnford[p].length];
            isAliveProzess[p] = false;
            for (int r = 0; (r < maxAnford[p].length); r++) {
                tempRes[r] = curProzess[p][r];
                Resources[r] += curProzess[p][r];
                curProzess[p][r] = 0;
            }

            // rekursiv: ist diese Stellung sicher?
            safe = sicher(isAliveProzess);

            // nach dem Test: Ressourcen dieses Prozesses wieder belegen
            isAliveProzess[p] = true;
            for (int r = 0; (r < maxAnford[p].length); r++) {
                curProzess[p][r] = tempRes[r];
                Resources[r] -= curProzess[p][r];
            }
        }
        return safe;
    }

    private void printBelegung() {
        System.out.print("\tBank: [");
        for (int r=0; r < Resources.length; r++) {
            System.out.print(""+Resources[r]);
            if (r < (Resources.length-1))
                System.out.print(",");
        }
        System.out.print("]");

        System.out.print(" Prozesse: ");
        for (int p=0; p < curProzess.length; p++) {
            System.out.print("[");
            for (int r=0; r < curProzess[p].length; r++) {
                System.out.print(""+curProzess[p][r]);
                if (r < (maxAnford[p].length-1))
                    System.out.print(",");
            }
            System.out.print("]");
            if (p < (maxAnford.length-1))
                System.out.print(",");
        }
        System.out.println("");
    }
}
```

c)
```
public class Beispiel_Geld {
    static int[] maxCapital = {30};
```

```
    static int[][] maxCredit = {{20}, {12}, {9}, {14}, {23}};

    public static void main(String[] argv) {
            Bankier myBanker = new Bankier(maxCredit, maxCapital);
            Prozess customers[] = new Prozess[maxCredit.length];
            for (int i=0; i  <maxCredit.length; i++) {
                    customers[i] = new Prozess(i, myBanker, maxCredit[i]);
                    customers[i].start();
            }
    }
}
```

d)

```
public class Beispiel_Aufg33 {
   static int[] maxResources = {6,15, 8,10, 9};
   static int[][] maxMatrix  ={{3, 5, 8,10, 1},
                               {2, 5, 3, 3, 2},
                               {4,12, 4, 9, 2},
                               {6, 1, 4, 5, 5}};


    public static void main(String[] argv) {
            Bankier myBanker = new Bankier(maxMatrix, maxResources);
            Prozess customers[] = new Prozess[maxMatrix.length];
            for (int i=0; i < maxMatrix.length; i++) {
                    customers[i] = new Prozess(i, myBanker, maxMatrix[i]);
                    customers[i].start();
            }
    }
}
```

e) Beispiel-Ausgabe für das Beispiel mit einem Betriebsmittel — „Geld“

```
        Bank: [30] Prozesse: [0],[0],[0],[0],[0]
belegen(0,0,12)
        Bank: [18] Prozesse: [12],[0],[0],[0],[0]
belegen(1,0,8)
        Bank: [10] Prozesse: [12],[8],[0],[0],[0]
belegen(2,0,8)
        Bank: [2] Prozesse: [12],[8],[8],[0],[0]
belegen(3,0,2) ---> muss warten.
belegen(4,0,2) ---> muss warten.
belegen(0,0,1)
        Bank: [1] Prozesse: [13],[8],[8],[0],[0]
belegen(1,0,1) ---> muss warten.
belegen(2,0,1)
        Bank: [0] Prozesse: [13],[8],[9],[0],[0]
belegen(0,0,1) ---> muss warten.
freigeben(2,0,9)
belegen(3,0,2)
        Bank: [7] Prozesse: [13],[8],[0],[2],[0]
```

```
belegen(4,0,2)
        Bank: [5] Prozesse: [13],[8],[0],[2],[2]
belegen(1,0,1)
        Bank: [4] Prozesse: [13],[9],[0],[2],[2]
belegen(0,0,1)
        Bank: [3] Prozesse: [14],[9],[0],[2],[2]
belegen(3,0,10) ---> muss warten.
belegen(4,0,20) ---> muss warten.
belegen(1,0,2)
        Bank: [1] Prozesse: [14],[11],[0],[2],[2]
belegen(0,0,5) ---> muss warten.
belegen(1,0,1)
        Bank: [0] Prozesse: [14],[12],[0],[2],[2]
freigeben(1,0,12)
belegen(4,0,20) ---> muss warten.
belegen(0,0,5)
        Bank: [7] Prozesse: [19],[0],[0],[2],[2]
belegen(3,0,10) ---> muss warten.
belegen(0,0,1)
        Bank: [6] Prozesse: [20],[0],[0],[2],[2]
freigeben(0,0,20)
belegen(4,0,20)
        Bank: [6] Prozesse: [0],[0],[0],[2],[22]
belegen(3,0,10) ---> muss warten.
belegen(4,0,1)
        Bank: [5] Prozesse: [0],[0],[0],[2],[23]
freigeben(4,0,23)
belegen(3,0,10)
        Bank: [18] Prozesse: [0],[0],[0],[12],[0]
belegen(3,0,1)
        Bank: [17] Prozesse: [0],[0],[0],[13],[0]
belegen(3,0,1)
        Bank: [16] Prozesse: [0],[0],[0],[14],[0]
freigeben(3,0,14)
```