A. Büscher, J. Mäter,
D. Müller, W. Rupflin

Dortmund, 8. Mai 2007
Besprechung: 12.06. - 14.06.

Übungen zur Vorlesung

Betriebssysteme, Rechnernetze und verteilte Systeme I

Sommersemester 2007

Projekt 1

Bearbeitungshinweise zu Projekt 1:

- Eine Beschreibung zum Verfahren der Bearbeitung und Abgabe des Projekts mittels der Rechner am Fachbereich in den Rechnerpools/Lernräumen wird in Kürze auf der Webseite zur Übung bereitgestellt.
- Die schriftlich zu bearbeitenden Aufgaben sind als PDF oder .txt-Dateien im Projektordner abzulegen.
- Die bearbeiteten Sourcedateien müssen Java 1.4 oder 1.5 konform sein.
- Es sind lediglich die Dateien *Queue.java*, *ConsumerBusyWating.java* und *ConsumerMonitor.java* abzuändern. Bitte ändern Sie an den anderen Quellcodedateien nichts!
- Die Abgabe muss bis zum 03.06.2007 um 24 Uhr erfolgt sein.

In der Vorlesung haben Sie das Produzenten/Konsumenten-Problem kennengelernt, bei dem ein Produzent Objekte in einen Puffer einfügt und der Konsument diese ausliest. Im ersten Projekt soll eine Variante dieses Problems betrachtet werden. Der Produzent schreibt in beliebiger Reihenfolge die Werte $0...n-1$ in einen Puffer. Der Konsument darf Werte nur der Reihe nach aus dem Puffer lesen. Um z.B. eine 5 aus dem Puffer lesen zu können, muss er vorher bereits eine 4 gelesen haben, usw.

Aufgabe 1.1 Auf den Seiten 2 bis 4 finden sie Javaklassen für den Produzenten, den Konsumenten, die Queue sowie ein Testprogramm.

(a) Was fehlt den vorgegebenen Methodenköpfen in der Klasse *Queue*, damit verschiedene Threads nebenläufig zugreifen können? Implementieren Sie die fehlenden Methodenrümpfe der Klasse *Queue*.

(b) Implementieren Sie die run-Methode der Klasse *ConsumerBusyWaiting* so, dass sie die Funktionalität des *busy waiting* erfüllt. Zur Ausgabe der konsumierten Werte nutzen sie bitte die Methode consume.
(Tipp: In Java kann ein Thread für eine gewisse Zeit deaktiviert werden, indem die sleep Methode aufgerufen wird - Eine Verzögerung um 10ms sollte hier genutzt werden).

(c) Was sind die Nachteile von *busy waiting*? Welche Lösungen gibt es?

Aufgabe 1.2 Da *busy waiting* keine gute Lösung ist, soll das System nun durch Monitore realisiert werden. Machen Sie sich mit der Benutzung von Monitoren in der Sprache Java vertraut (Tipp: Es gibt ein Java Tutorial zu diesem Thema auf http://java.sun.com/docs/books/tutorial.

(a) Erweitern Sie die Klasse *ConsumerMonitor* so, dass die Lösung ohne *busy waiting* auskommt. Zur Ausgabe der konsumierten Werte nutzen sie bitte die Methode consume.

(b) Welche Beschränkung gibt es in Java bezüglich der Realisierung von Monitoren?

Aufgabe 1.3 Bisher hatten wir angenommen, dass der Puffer eine unendlich große Kapazität hat. Nun soll diese Kapazität auf m Elemente begrenzt werden.

(a) Welche Probleme treten auf, wenn diese Kapazität begrenzt ist?

(b) Geben Sie eine Eingabereihenfolge an, für welche das Programm nicht terminiert.

(c) Wie groß muss die Kapazität des Puffers mindestens sein, damit das Programm garantiert terminiert?

Listing 1: TestBusyWaiting.java

```
/** An dieser Klasse duerfen keine Aenderungen vorgenommen werden. */
public class TestBusyWaiting {

    public static final int LENGTH = 100;

        public static void main(String[] args) {
                Queue q = new Queue();
                ConsumerBusyWaiting c = new ConsumerBusyWaiting(q, LENGTH);
                Producer p = new Producer(q, LENGTH);

                c.start();
                p.start();

        }
}
```

Listing 2: TestMonitor.java

```
/** An dieser Klasse duerfen keine Aenderungen vorgenommen werden. */
public class TestMonitor {

    public static final int LENGTH = 100;

        public static void main(String[] args) {
                Queue q = new Queue();
                ConsumerMonitor c = new ConsumerMonitor(q, LENGTH);
                Producer p = new Producer(q, LENGTH);

                c.start();
                p.start();

        }

}
```

Listing 3: Producer.java

```
import java.util.*;

/** An dieser Klasse duerfen keine Aenderungen vorgenommen werden. */
public class Producer extends Thread {

    private final Queue q;
    private final int n;

    public Producer(Queue q, int numInputValues) {
        super();
        this.q = q;
        this.n = numInputValues;
    }

    public void run() {
        int[] inputValues = new int[n];
        for (int i = 0; i < n; i++) {
            inputValues[i] = i;
        }

        // Generiert eine zufaellige Permutation
        Random rng = new Random();

        for (int j = 0; j < 10 * n; j++) {
            int pos1 = rng.nextInt(n);
            int pos2 = rng.nextInt(n);

            int buf = inputValues[pos1];
            inputValues[pos1] = inputValues[pos2];
            inputValues[pos2] = buf;
        }

        for (int i = 0; i < n; i++) {
            System.out.println("Producer: Writing value " + inputValues[i]);
            q.addValue(inputValues[i]);

            try {
                sleep(10);
            } catch (InterruptedException e) {

                e.printStackTrace();
            }
        }
        System.out.println("Producer: "
                + "All values successfully written to the queue");
    }

}
```

Listing 4: Queue.java

```
import java.util.*;

public class Queue {

    public void removeLowestValue() {
    }

    public int getLowestValue() {
    }

    public boolean isEmpty() {
    }

    public void addValue(int n) {
    }
}
```

Listing 5: ConsumerBusyWaiting.java

```
public class ConsumerBusyWaiting extends Thread {

    private final int n;
    private final Queue q;

    public ConsumerBusyWaiting(Queue q, int numOutputValues) {
        this.n = numOutputValues;
        this.q = q;
    }

    public void run() {
    }

    public void consume(int i) {
        System.out.println("Consumer: Reading value " + i);
    }
}
```

Listing 6: ConsumerMonitor.java

```
public class ConsumerMonitor extends Thread {

    private final int n;
    private final Queue q;

    public ConsumerMonitor(Queue q, int numOutputValues) {
        this.n = numOutputValues;
        this.q = q;
    }

    public void consume(int i) {
        System.out.println("Consumer: Reading value " + i);
    }
}
```