

Betriebssysteme im Wintersemester 2008/2009

Übungsblatt 5

Abgabetermin: 12.11.2008, 13:30 Uhr

Achtung: Die Klausur zur Vorlesung Betriebssysteme findet am **Samstag, 07.02.09 ab ca. 9 Uhr** im Hauptgebäude statt. Nähere Informationen werden rechtzeitig auf der Webseite bekannt gegeben.

Aufgabe 18: (H) Threads in Java I

(6 Pkt.)

- Betrachten Sie das folgende Java-Programm, das Sie auch von der Betriebssysteme-Homepage herunterladen können. Als Eingabeparameter verlangt es eine Integer-Zahl. Wie hängt diese Zahl mit der Ausgabe zusammen? Welche Ausgabe erwarten Sie für verschiedene Integer-Eingaben (zum Beispiel 1, 2, 100 oder 10000)?
- Kompilieren Sie das Programm, und führen Sie es auf verschiedenen, Ihnen zugänglichen Systemen aus. Protokollieren und kommentieren Sie das Ergebnis.

```
public class SimplerThread extends Thread {  
2  
    String msg ;  
4    int cycles ;  
  
6    SimplerThread(String m, int c) {  
        msg = m ;  
8        cycles = c ;  
    }  
  
10    // overrides run() in Thread class to define object's  
12    // behavior.  
    public void run() {  
14        for (int i = 0 ; i < cycles ; i++) {  
            System.out.println(msg) ;  
16        }  
    }  
  
18    // command-line arguments are integers C.  
20    // builds and starts two threads of type SimplerThread.  
    // continues for C cycles.  
22  
    public static void main(String[] args) {  
24  
        if (args.length < 1) {  
26            System.out.println("Arguments_are:") ;
```

```

28         System.out.println("cycles") ;
        System.exit(-1) ;
    }

30

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

32

    SimplerThread t1 =
34         new SimplerThread("ping---->", C) ;
    SimplerThread t2 =
36         new SimplerThread("<----pong", C) ;

38

    t1.start() ;
    t2.start() ;

40    }
}

```

Aufgabe 19: (H) Threads in Java II

(4 Pkt.)

Das folgende kurze Java-Programm können Sie von der Betriebssysteme-Homepage herunterladen:

```

public class Test extends Thread {
2    String msg;

4    Test(String m) {
        msg = m;
6    }

8    public void start() {
        for (int i = 0; i < 1000; i++) {
10        System.out.println(msg);
        }

12    }

14    public static void main(String[] args) {
        Test t1 = new Test("ping---->");
16        Test t2 = new Test("<----pong");
        t1.start();
18        t2.start();
        }

20 }

```

- Welche Ausgabe liefert dieses Programm? Ist die Ausgabe deterministisch?
- Damit die Instanzen `t1` und `t2` threaded ausgeführt werden, muss eine kleine Änderung an der Klasse `Test` vorgenommen werden. Skizzieren Sie diese Änderung und erklären Sie in wenigen Sätzen, was sie bewirkt.

Aufgabe 20: (K) Race Condition

(10 Pkt.)

Ist das Ergebnis von nebenläufigen Threads nicht deterministisch (hängt also von der Ablaufreihenfolge der Threads ab), so spricht man von einer Race Condition. Gehen Sie davon aus, dass mehrere Threads auf den folgenden Klassen arbeiten. Entscheiden Sie, ob die folgenden Aussagen zutreffen. Begründen Sie Ihre Antwort.

- a. Wenn ein Thread nur liest und ein anderer Thread nur schreibt, so kann keine Race Condition eintreten.
- b. Der Text 'Race Condition' wird niemals ausgegeben.

```

class example1 {
2   private boolean flag;

4   public void setTrue() {
        flag= true;
6       if ( !flag ) then {
            System.out.println("Race_Condition");
8       }
    }
10  public void setFalse() {
        flag= false;
12     if ( flag ) {
            System.out.println("Race_Condition");
14     }
    }
16 }

```

- c. Es wird stets der inkrementierte bzw. der dekrementierte Wert von count ausgegeben.

```

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

4

6   public int increment() {
        return ++count; }
    public void decrement() {
8        return --count; }
    }

```

- d. Innerhalb einer Methode wird der Wert von count bzw. flag nur wie vorgesehen verändert.

```

class example3 {
2   private boolean flag;
    private int count;

4

6   public void switch() { flag = !flag; }
    public void set(int i) { count= i; }
    public int get(){ return count; }
8 }

```

Aufgabe 21: (T) CPU-Scheduling: Wartezeiten

(20 Pkt.)

Wenn auf einer CPU mehrere Prozesse (pseudo-)parallel ausgeführt werden, kommt es für einzelne Prozesse in der Regel zu Wartezeiten vor ihrer Ausführung und (preemptives Scheduling vorausgesetzt) zwischen zwei Ausführungseinheiten. In dieser Aufgabe sollen verschiedene Scheduling-Strategien im Hinblick auf die mittleren Warte- (und Verweil-)Zeiten der Prozesse verglichen werden.

Dazu sei ein Ein-Prozessor-System mit elf Aufträgen (Prozessen) A_1, \dots, A_{11} , ihren Ankunftszeitpunkten $a = (0, 2, 3, 4, 6, 7, 12, 14, 16, 22, 26)$ und den Bedienzeiten $b = (5, 6, 2, 3, 2, 1, 2, 2, 2, 3, 1)$ gegeben (d.h. Auftrag A_i kommt zum Zeitpunkt a_i ins System und benötigt die CPU für b_i Zeiteinheiten). Vor dem Zeitpunkt $t = 0$ seien keine Aufträge vorhanden.

- a. Stellen Sie das Prozess-Scheduling für die folgenden drei Strategien grafisch dar. Verwenden Sie dabei eine Darstellung, in der auch Warte- und Verweilzeiten erkennbar sind. Lesen Sie aus Ihren Diagrammen jeweils die Terminierungszeitpunkte sowie die Warte- und Verweilzeiten der einzelnen Prozesse ab. Berechnen Sie dann die mittlere Wartezeit und die mittlere Verweilzeit für jede der drei Strategien. Beachten Sie, dass Ausführungsunterbrechungen ausschließlich zu Ankunftszeitpunkten neuer Aufträge oder zu Terminierungszeitpunkten fertiger Aufträge möglich sind.
- (i) **SRPT (Shortest Remaining Processing Time)**: In jedem Zeitintervall wird einer der Aufträge mit kürzester Restbedienzeit ausgeführt.
 - (ii) **LPT (Largest Processing Time)**: Es wird jeweils unterbrechend der Auftrag mit der längsten Gesamt(!)-Bedienzeit ausgeführt.
 - (iii) **FCFS (First Come First Serve)**: Die Aufträge werden in der Reihenfolge ihrer Ankunft nicht-unterbrechend ausgeführt.
- b. Der Vektor der Ankunftszeiten werde modifiziert zu $\alpha' = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, d.h. alle Aufträge liegen schon von Beginn an vor. Die Aufträge werden nun sequenziell nach einer Zeitscheibenstrategie mit dem Quantum q (Round-Robin-Strategie) bedient, beginnend mit A_1 . Ermitteln Sie wieder Terminierungszeitpunkt, Warte- und Verweilzeit für jeden Auftrag, sowie die mittlere Wartezeit und die mittlere Verweilzeit, und zwar
- (i) für das Zeitquantum $q_1 = \frac{3}{2}$ und
 - (ii) für das Zeitquantum $q_2 = \frac{1}{2}$.
- c. Warum lassen sich die Ergebnisse für die mittleren Warte-/Verweilzeiten aus Teilaufgabe a. nicht mit denen aus Teilaufgabe b. vergleichen?