Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
12. Oktober 2006

Lösungsvorschläge der Klausur zu Einführung in die Informatik II

Aufgabe 1 Transferfragen (Lösungsvorschlag) (10 Punkte)

a) Falsch: umso mehr Wiederholungen die Daten aufweisen (niedrige Entropie), desto besser kann komprimiert werden

b) Falsch: die Summe reicht nicht aus, es fehlt eine Normierung, nämlich die Wahrscheinlichkeit der Zeichen

c) Falsch: der Aufwand steigt exponentiell

d) Wahr

e) Wahr

f) Wahr

g) Falsch: Threads führen erst bei Verwendung von Multiprozessorsystemen zu einer Beschleunigung der Ausführung, eher selten auf einem 1-Rechnersystem

h) Falsch: Präemptives Scheduling wird unterstützt, Benutzerprozessen kann somit die CPU entzogen werden, der Rechner bleibt seltener hängen

i) Wahr

j) Wahr

Aufgabe 2 Barrieren (10 Punkte)

```
public class Barrier {
  private int processNumber;
  private int processCount;

  public Barrier (int number)
  {
    processNumber = number;
    processCount = 0;
  }

  public synchronized int getWaitingProcessCount()
  {
    return processCount;
  }

  public synchronized void setReached()
```

```
  {
    processCount++;
    if(processCount==processNumber)
    {
      notifyAll();
    }
    else
    {
      try {
        wait();
      } catch (InterruptedException e) {
      }
    }
  }
}
```

Aufgabe 3 Züge (Lösungsvorschlag) **(10 Punkte)**

Station.java

```
import java.util.concurrent.Semaphore;

public class Station {
  public static void main(String[] args) {
    int trackCount = 3;
    int trainCount = 5;

    Semaphore bottleneck = new Semaphore(1, true);
    Semaphore tracks = new Semaphore(trackCount, true);

    for (int i = 0; i < trainCount; i++) {
      new Train(i, bottleneck, tracks).start();
    }
  }
}
```

Train.java

```
import java.util.concurrent.Semaphore;

public class Train extends Thread {
  public int number;
  private Semaphore bottleneck;
  private Semaphore tracks;

  Train(int number, Semaphore bottleneck, Semaphore
     tracks) {
    this.bottleneck = bottleneck;
    this.number = number;
    this.tracks = tracks;
  }

  void arrive() {
```

```
    try {
      System.out.println("Zug " + this.number + " wartet
         auf Einfahrt...");
      this.tracks.acquire();
      this.bottleneck.acquire();

      System.out.println("Zug " + this.number + " faehrt
         ein...");
      this.sleep(2000);

      this.bottleneck.release();
      System.out.println("Zug " + this.number + " ist
          eingefahren.");
    } catch (InterruptedException e) {
    }
  }

  void depart() {
    try {
      System.out.println("Zug " + this.number + " wartet
         auf Ausfahrt...");
      this.bottleneck.acquire();

      System.out.println("Zug " + this.number + " faehrt
         aus...");
      this.sleep(2000);

      this.bottleneck.release();
      this.tracks.release();
      System.out.println("Zug " + this.number + " ist
          ausgefahren.");
    } catch (InterruptedException e) {
    }
  }

  public void run() {
    while (true) {
      try {
        arrive();
        waitInStation();
        depart();

        Thread.sleep((int) (Math.random() * 4000));
      } catch (InterruptedException e) {
      }
    }
  }

  public void waitInStation() {
    try {
      Conductor conductor = new Conductor(this);
```

```
      conductor.start();
      conductor.join();
    } catch (InterruptedException e) {
    }
  }
}
```

Conductor.java

```
public class Conductor extends Thread {
  private Train train;

  Conductor(Train train) {
    this.train = train;
  }

  public void run() {
    try {
      System.out.println("Schaffner␣von␣Zug␣" + this.
          train.number + "␣wartet...");
      Thread.sleep((int) (Math.random() * 3000) + 2000);
      System.out.println("Schaffner␣von␣Zug␣" + this.
          train.number + "␣ist␣bereit.");
    } catch (InterruptedException e) {
    }
  }
}
```

Aufgabe 4 Kompressionsalgorithmen (Lösungsvorschlag) (10 Punkte)

a) Die Tabelle sieht wie folgt aus:

p	*z*	**Neueintrag**	**Kodierung**
""	'M'		
"M"	'A'	27="MA"	13
"A"	'M'	28="AM"	1
"M"	'A'		
"MA"	'M'	29="MAM"	27
"M"	'A'		
"MA"	'M'		
"MAM"	'A'	30="MAMA"	29
"A"	'M'		
"AM"	'I'	31="AMI"	28
"I"	'A'	32="IA"	9

D.h. es ergibt sich die Kodierung 13,1,27,29,28,9,1. Die "1" am Schluss ist die Kodierung des Puffers, wenn keine Eingabezeichen mehr vorhanden sind (Abbruchkriterium).

b) Optimal ist 4{MA}1{MIA}

c) “4,1,3,5,1”

d) Die Antwort sollte in etwa enthalten:
LZW: automatische Wörterbucherzeugung aber evtl. wird unnötig viel Speicher für Einträge vergeudet, die nie zur Anwendung kommen
LLK: gute Kompression von unbekannten Sequenzen aber Wiederholungen müssen zuerst erkannt werden; evtl. durch vormaligen Durchlauf
LFF: liefert im Beispiel beste Kompression aber ansonsten ineffizient und unflexibel, da mehere Durchläufe nötig und festes Wörterbuch

Aufgabe 5 RSA: Schnelles Potenzieren (Lösungsvorschlag) (2+4 Punkte)

a) Ein mögliche Implementierung lautet:

```
let rec n_pot n = match n with
        |1 -> 1
        |n when (n mod 2 = 1) -> 2 + n_pot ((n-1)/2)
        |n -> 1 + n_pot (n/2);;
```

b) Beweis durch Induktion:

Induktionsanfang: $n = 1$; offensichtlich ist dann auch $M(n = 1) = 1$, also nicht größer als $1 + 2\log_2 1 = 1$.

Induktionsschluss: Sei nun $n > 1$. Gemäß IH ist $M(n) \leq 1 + 2\log_2 n$. Für $n > 1$ besitzt der angegebene Algorithmus dann

$$M(n+1) = max\{2 + M(n/2), 1 + M((n+1)/2)\}$$

Aufrufe. Offensichtlich ist dieses Maximum durch

$$M(n+1) = max\{2 + M(n/2), 1 + M((n+1)/2)\} \leq 2 + M((n+1)/2)$$

abschätzbar. Gemäß IV gilt:

$$M(n+1) \leq 2 + 1 + 2log_2((n+1)/2) = 1 + 2log_2(n+1).$$

Aufgabe 6 RSA (Lösungsvorschlag) (4 Punkte)

Bei einfacher RSA-Verschlüsselung verschlüsselt der Sender die Nachricht M mit der Verschlüsselungsfunktion $V_e(M)$, der Empfänger wendet auf diese die Entschlüsselungsfunktion $E_d(V_e(M))$ an und gewinnt so wieder die ursprüngliche Nachricht. Aus dem Beweis auf Folie 128 der Vorlesung geht hervor, dass die Verschlüsselungsfunktionen kommutativ sind, d.h. es gilt $E_d(V_e(M)) = V_e(E_d(M)) = M$. Man kann also auch zuerst die Nachricht mit dem privaten Schlüssel verschlüsseln und diese kann dann mit dem öffentlichen geöffnet werden. Wendet nun der Sender vor dem Verschlüsseln einer Nachricht mit dem öffentlichen Schlüssel des Empfängers V_e seinen privaten Schlüssel $E_{d'}$ auf die Nachricht an, so muss der Empfänger zunächst seinen privaten Schlüssel E_d anwenden und dann den öffentlichen Schlüssel des Senders $V_{e'}$ um die Authentizität des Senders zu gewährleisten. Mathematisch kurz ergibt sich für diesen Vorgang:

$$V_{e'}(E_d(V_e(E_{d'}(M)))) = M$$