

Betriebssysteme im Wintersemester 2008/2009

Übungsblatt 7

Abgabetermin: 26.11.2008, 13:30 Uhr

Aufgabe 27: (H) Deadlock-Vermeidung

(3 Pkt.)

Eine Methode, um Deadlocks zu vermeiden, ist es, eine der Bedingungen für das Entstehen von Deadlocks im Vorhinein auszuschließen.

- Geben sie die vier Voraussetzungen für die Entstehung eines Deadlocks an.
- Beschreiben Sie, wie durch eine Ordnung der Ressourcen bei geeigneter Reservierungsstrategie Deadlocks vermieden werden können.

Tipp: Ordnung der Ressourcen bedeutet: Ob ein Prozess ein Betriebsmittel anfordern darf, hängt nicht nur davon ab, ob dieses gerade frei ist, sondern auch davon, welche Betriebsmittel der Prozess zuvor schon angefordert hat. Welche der vier Deadlock-Bedingungen könnte man mit diesem Ansatz ausschließen?

Aufgabe 28: (H) Behebung von Deadlocks

(4+2 Pkt.)

- Nennen Sie vier Ansätze zur Behandlung von (bereits eingetretenen) Deadlocks.
- In einem Rechensystem seien 8 Magnetbänder vorhanden, um die n Prozesse konkurrieren. Jeder dieser Prozesse benötige maximal 3 Magnetbänder. Für welche Werte von n besteht keine Verklemmungsgefahr? Begründen Sie Ihre Antwort.

Aufgabe 29: (K) Multilevel Feedback Queuing (MLFQ)

(12 Pkt.)

Gegeben seien die Prozesse P_1, \dots, P_5 mit den folgenden Ankunfts- und Rechenzeiten:

Prozess	Ankunftszeitpunkt	Rechenzeit
P_1	0	3
P_2	2	7
P_3	2	2
P_4	6	4
P_5	7	4

Hierbei gilt: Trifft ein Prozess zum Zeitpunkt t ein, so wird er erst zum Zeitpunkt $(t + 1)$ berücksichtigt. Damit verbringt jeder Prozess mindestens eine Zeiteinheit im Zustand wartend. Wird ein Prozess zum Zeitpunkt t' unterbrochen, so reiht er sich auch zum Zeitpunkt t' wieder in die Warteschlange ein. Sind zwei Prozesse absolut identisch bezüglich ihrer relevanten Werte, so wird nach der Prozess-ID entschieden (eine niedrigere Prozess-ID wird bevorzugt).

- a. Geben Sie für die Strategie **Multilevel Feedback Queuing (MLFQ)** in Form eines Gantt-Charts die Zuteilung der Rechenzeit an die Prozesse an. Gehen Sie von 3 Prioritätsklassen mit folgenden Werten aus:

Priorität	Verfahren
0	Round-Robin mit Quantum 1
1	Round-Robin mit Quantum 2
2	First Come First Served (FCFS)

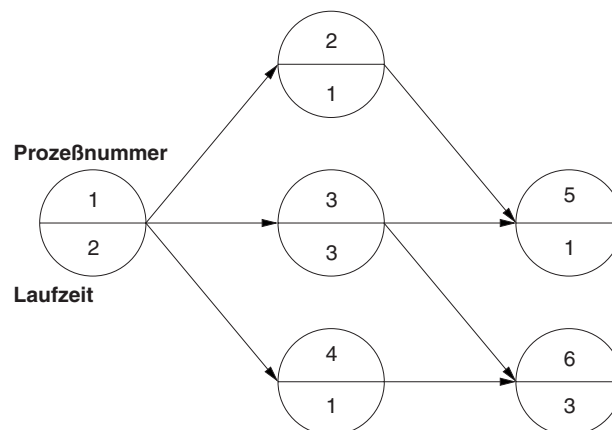
- b. Geben Sie für die Prozesse P_1 und P_4 jeweils die Verweilzeit (Antwortzeit) und die Wartezeit an.

Aufgabe 30: (T) Grahams List Scheduling

(10 Pkt.)

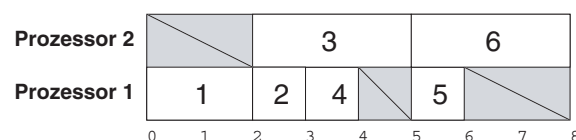
In der heutigen Zeit ist es interessant, parallele Programme auch auf mehreren Prozessoren gleichzeitig ausführen zu können. Grahams List-Scheduling ist ein für diesen Zweck entwickelter Scheduling-Algorithmus, der nach dem Greedy-Prinzip arbeitet: Ein freier Prozessor nimmt den ersten rechenbereiten Prozess aus der Prozesswarteschlange (geordnet nach Prozessnummern) und bearbeitet diesen. Sind mehrere Prozessoren frei, bedient sich der mit der kleineren Nummer zuerst.

Die folgenden sechs Prozesse (gegeben durch Prozessnummer und Laufzeit) seien, wie im folgenden Diagramm dargestellt, voneinander abhängig:

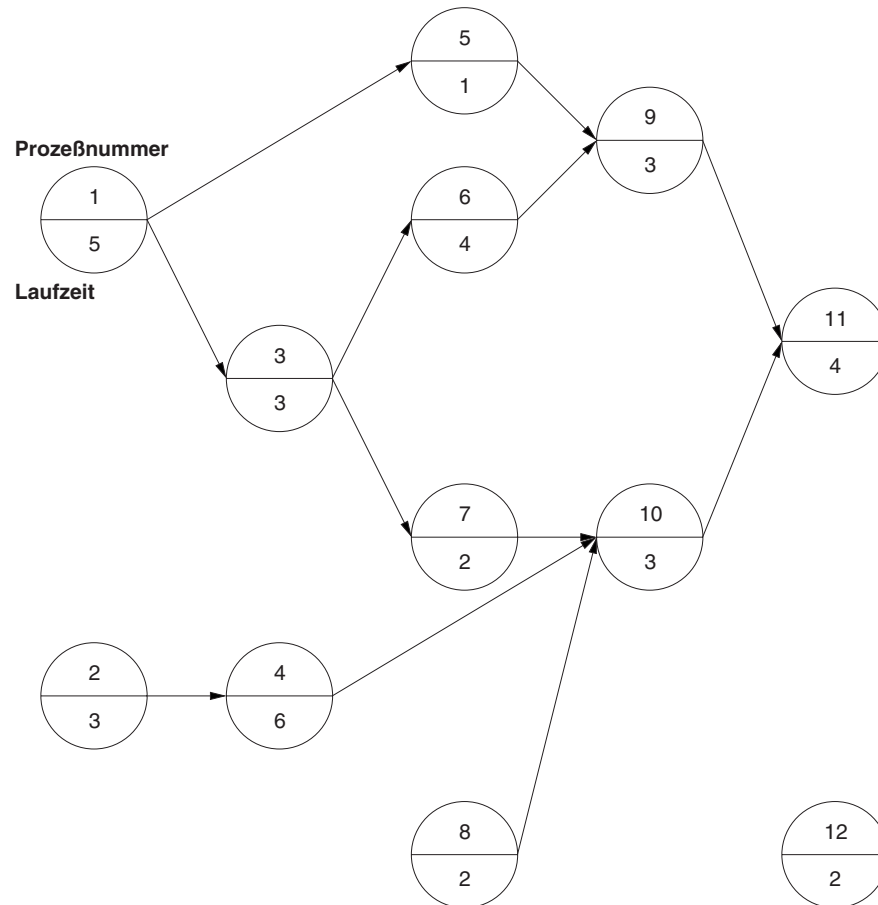


Prozesse 2, 3 und 4 müssen also erst auf das Ergebnis von Prozess 1, Prozess 5 auf das Ergebnis von 2 und 3 etc. warten.

Bei zwei Prozessoren ergibt sich folgendes Gantt-Diagramm:



- a. Geben Sie für folgenden Abhängigkeits-Graphen das zugehörige Gantt-Diagramm für drei Prozessoren an:



- b. Bei Grahams List-Scheduling existieren drei Anomalien: Die Laufzeit kann steigen bei
- Erhöhung der Prozessoranzahl,
 - Entfernung von Kanten und
 - Reduzierung der Dauer eines Prozesses.

Geben Sie jeweils ein Beispiel an.

Aufgabe 31: (H) Scheduler-Simulation: Prozessdeskriptor (6 Pkt.)

Diese Aufgabe baut auf den Überlegungen aus Aufgabe 26 (letztes Übungsblatt) auf. Ziel ist die Implementierung der Simulation eines einfachen Schedulers in Java. Implementieren Sie eine Klasse `Process` für den *allgemeinen* (d.h. für alle gängigen Scheduling-Strategien grundlegenden) Prozessdeskriptor. Beachten Sie die folgenden Modellierungshinweise:

- Jeder Prozess besitzt eine Prozess-ID, die ihm bei seiner Erzeugung automatisch vom System zugeteilt wird.
- Im Prozessdeskriptor muss festgehalten werden, ob ein Prozess gerade aktiv ist oder suspendiert wurde. Die Klasse muss Methoden zur Verfügung stellen, um den jeweiligen Zustand abzufragen oder zu verändern.
- Eine Methode `setArriveTime()` soll dem jeweiligen Prozess bei ihrem Aufruf die aktuelle Systemzeit als Ankunftszeit zuweisen und in einer Variablen `arriveTime` speichern. Diese Methode hat keine Parameter und keinen Rückgabewert. Sie wird ausschließlich **von Unterklassen** von `Process` aufgerufen, also nicht schon im Konstruktor dieser Klasse.
- Analog zu `setArriveTime` soll die Methode `setDepartTime()` dem jeweiligen Prozess bei ihrem Aufruf die aktuelle Zeit als Endzeitpunkt zuweisen und in einer Variablen `departTime` speichern.
- Schließlich soll eine Methode zur Verfügung gestellt werden, die die Lebensdauer eines Prozesses als Differenz von `departTime` und `arriveTime` berechnet und zurück gibt.

Tipp: Das folgende Programmfragment legt ein Objekt `d` vom Typ `Date` an, das das aktuelle Datum und die genau Systemzeit enthält. In der Variablen `t` wird diese Zeit dann als Anzahl der Millisekunden seit dem 1. Januar 1970 (0.00 Uhr) gespeichert.

```
Date d = new Date();  
2 long t = d.getTime();
```

Damit das funktioniert, muss zuvor die Klasse `java.util.Date` importiert werden.