VO 182.022
Prüfung Betriebssysteme

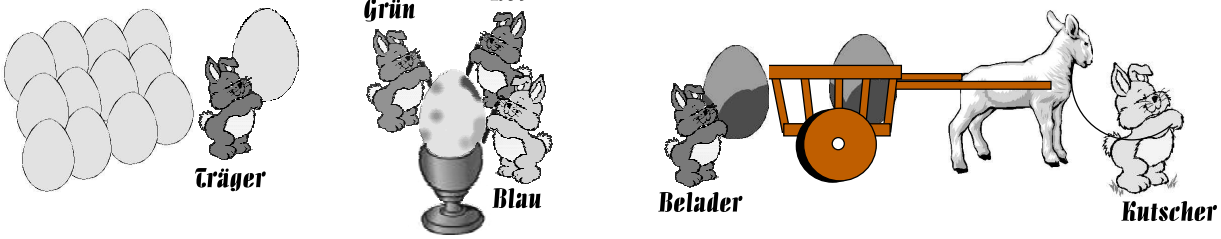
28. April 2005

KNr. | MNr. | Zuname, Vorname

Ges.)(100)	1.)(30)	2.)(25)	3.)(20)	4.)(25)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Immer wenn Ostern vor der Tür steht, herrscht bei den Osterhasen Hochkonjunktur. Eine Abteilung von sechs Osterhasen mit Namen *Träger*, *Rot*, *Grün*, *Blau*, *Belader* und *Kutscher* ist mit der Herstellung und dem Versenden von bemalten Ostereiern beauftragt.

Die Hasen sollen dabei nach folgendem Schema arbeiten:

- *Träger* holt jeweils ein frisches Ei aus einem unbegrenzten Vorrat und platziert dieses im Eierbecher, wenn dieser leer ist.
- *Rot*, *Grün* und *Blau* bemalen die Eier im Eierbecher in den entsprechenden Farben. Das Bemalen eines Eis kann prinzipiell gleichzeitig durchgeführt werden, aus ästhetischen Gründen darf jedes Ei aber nur jeweils **zweifarbig** bemalt werden. Es sollen dabei aber alle Farbkombinationen möglich sein. Welche beiden Farben dabei verwendet werden, darf zufällig sein.
- Wenn das Ei fertig bemalt ist, signalisieren *Rot*, *Grün* und *Blau* ihrem Kollegen *Belader*, dass das Ei abzuholen ist.
- *Belader* transportiert das bemalte Ei zum Wagen und belädt diesen.
- Ist der Wagen mit zwei Eiern voll, so signalisiert *Belader* dem *Kutscher* eine Lieferung zu machen.

- *Kutscher* liefert die Eier dann beim Kunden ab. Solange er unterwegs ist, dürfen keine neuen Eier auf den Wagen gelegt werden.

Synchronisieren Sie den Arbeitsablauf der sechs Hasen mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(` ***Semaphor,init*** `)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`gehezu(` ***Ziel*** `)` Bewegt den Hasen zum *Ziel*. Als Ziel kann `Vorrat`, `Eierbecher` oder `Wagen` angegeben werden.

`nimmEi()` Lässt den Hasen ein Ei aufnehmen. Funktioniert nur, wenn der Hase neben dem Vorrat bzw. neben dem Eierbecher steht.

`bemaleEi()` Diese Funktion lässt einen Hasen das im Eierbecher befindliche Ei verzieren. Diese Funktion blockiert solange bis der Hase mit dem Bemalen in seiner Farbe fertig ist.

`legeEi()` Der Hase legt das Ei dort ab, wo er gerade steht. Befindet er sich vor dem Wagen, so belädt er diesen. Befindet sich der Hase vor dem Eierbecher so legt er das Ei in diesem ab. Diese Funktion beachtet nicht, für das Ei ausreichend Platz zur Verfügung steht!

`lieferung()` Mit dieser Funktion liefert *Kutscher* die Eier beim Kunden ab. Diese Funktion blockiert solange, bis der Wagen wieder leer an seinem Platz steht.

a) Initialisierungen (8)

Initialisieren Sie die notwenigen Semaphore. Der **Anfangszustand** des Systems entspricht obigem Bild, d.h. *Träger* steht mit einem Ei beim Vorrat, *Rot, Grün* und *Blau* schicken sich gerade an, ein Ei zu bemalen und *Belader* steht mit einem Ei vor dem Wagen, auf dem sich bereits ein weiteres Ei befindet.

b) (10)

Entwerfen Sie die Prozesse, die in den Hasen *Träger* sowie *Rot, Grün* und *Blau* ablaufen:

Prozess *Träger*:

```
do forever() {

}
```

Prozess *Rot*|*Grün*|*Blau*:

```
do forever() {

}
```

c) (12)

Entwerfen Sie die Prozesse, die in den Hasen *Belader* und *Kutscher* ablaufen:

Prozess *Belader*:

```
do forever() {

}
```

Prozess *Kutscher*:

```
do forever() {

}
```

2 Scheduling (25)

Priority Scheduling (8)

Schedulen Sie das nebenstehende Task Set. Beachten Sie dabei folgendes: Zu den angegebenen Arrival Times wird ein Task in die der Priorität des Tasks entsprechende Priority Queue gestellt. Jeder Task kann frühestens beim nächsten diskreten Zeitpunkt dem Prozessor zugeteilt werden (Beispiel siehe Task A: Arrival Time= 0; Zuteilung = 1). Ein eintreffender Task wird immer ans Ende der entsprechenden Priority Queue gestellt. Abbildung 1 zeigt das Abarbeitungsschema.

Task	Arrival Time	Service Time	Priority
A	0	2	1
B	1	3	2
C	1	1	3
D	4	1	2
E	4	1	1
F	6	2	3
G	6	1	2
H	7	1	3
I	10	2	1
K	11	3	3

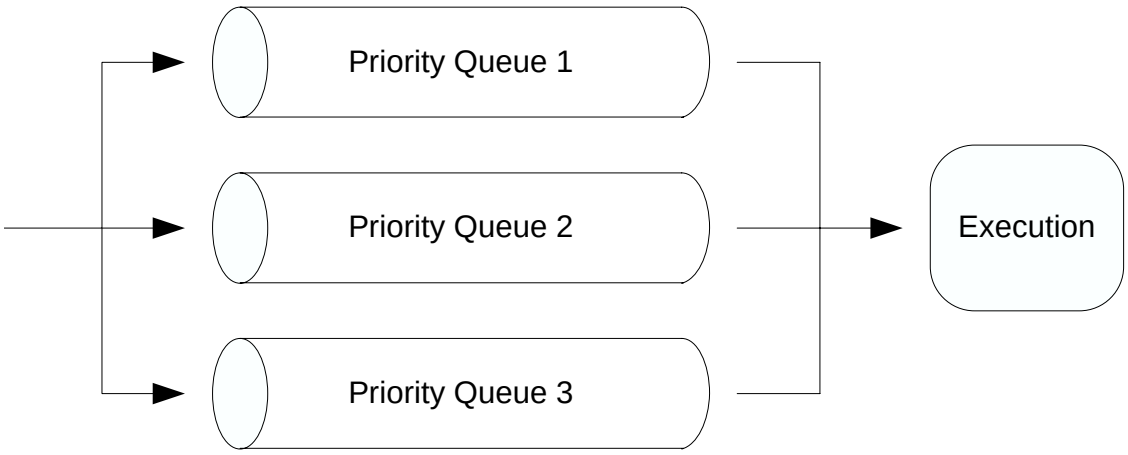
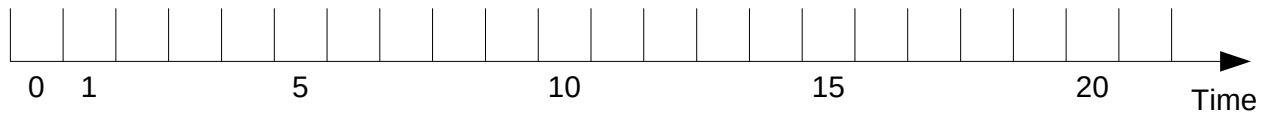


Abbildung 1: Priority Queuing

Beachten Sie, dass die Abarbeitung der Tasks *non preemptive* ist. Schreiben Sie in Abbildung 1 über jeder Priority Queue jene Tasks in zeitlich korrekter Reihenfolge, die dieser Queue zugewiesen werden. Tragen Sie weiters in das folgende Zeitschema die Abarbeitungsreihenfolge der Tasks ein. Der Overhead für den Taskwechsel ist vernachlässigbar.



Verständnisfrage (Eine fehlende Antwort wird negativ, eine falsche Antwort wird doppelt negativ gewertet!)

◯ Ja ◯ Nein Kann es beim Scheduling mit Priority Queues zur Starvation kommen?

Starvation (3)

Erklären Sie den Begriff *Starvation*:

Round-Robin Scheduling (7)

Task	Arrival Time	Service Time
A	0	3
B	5	5
C	7	2
D	10	4
E	15	2

Schedulen Sie das nebenstehende Task Set. Beachten Sie dabei folgendes: Zu den angegebenen Arrival Times wird ein Task in die Ready Queue gestellt. Der Task kann frühestens beim nächsten diskreten Zeitpunkt dem Prozessor zugeteilt werden (Beispiel siehe Task A: Arrival Time= 0; Zuteilung = 1). Ein eintreffender Task wird immer ans Ende der Ready Queue gestellt.

Tragen Sie in der Zeile **P** jenen Task ein, der für die entsprechende Zeiteinheit dem Prozessor zugeteilt wird. Der restliche Raster stellt die Ready Queue dar. Tragen Sie hier jene Tasks ein die in der Ready Queue stehen (beginnend in der obersten Zeile mit dem Taks, der als nächstes den Prozessor zugeteilt bekommt). Schedulen Sie das Task Set nach der Round-Robin Methode mit time-slice = 1. Der Overhead für den Taskwechsel ist vernachlässigbar.

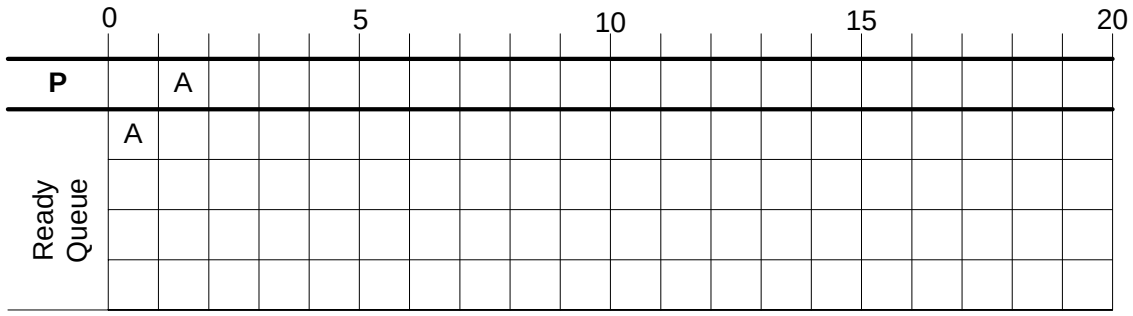


Abbildung 2: Round Robin

Verständnisfragen (7)

Beurteilen Sie die folgenden Aussagen! Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!

◯ Ja ◯ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.

◯ Ja ◯ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die *Processor Utilization* der Tasks.

◯ Ja ◯ Nein Earliest Deadline First Scheduling findet in Single-Prozessor Systemen immer eine Lösung.

◯ Ja ◯ Nein Earliest Deadline First Scheduling findet in Single-Prozessor Systemen immer eine Lösung, wenn eine solche existiert.

◯ Ja ◯ Nein Bei Round Robin Scheduling kann das Problem der Starvation auftreten.

◯ Ja ◯ Nein Die First-Come-First-Serve-Strategie begünstigt Prozesse mit kurzer Ausführungszeit.

◯ Ja ◯ Nein Das Round-Robin-Verfahren ist preemptive.

3 Deadlock (20)

Ein Stahlhersteller produziert abhängig von der Nachfrage unterschiedliche legierte Stähle mit spezifischen chemischen Zusammensetzungen.

Um die laufenden Aufträge erfüllen zu können, müssen Ressourcen wie Container, Frachtschiffe und Güterzüge koordiniert werden.

Dem Stahlhersteller stehen 80 Container, 3 Frachtschiffe und 5 Züge zur Verfügung.

Zur Zeit sind drei Aufträge in Arbeit:

- Auftrag: # 1 beinhaltet den Transport von 1000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 10 Container, 1 Frachtschiff und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 10 Container und 2 Güterzüge reserviert.
- Auftrag: # 2 beinhaltet den Transport von 4000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 40 Container, 1 Frachtschiff und 4 Güterzüge notwendig. Die Bearbeitung dieses Auftrags hat noch nicht begonnen.
- Auftrag: # 3 beinhaltet den Transport von 1600 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 16 Container, 2 Frachtschiffe und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 16 Container und ein Frachtschiff reserviert.

Process Initiation Denial (10)

Verwenden Sie die Strategie des *Process Initiation Denial*.

Beschreiben Sie *Resource-* und *Available-Vektor* sowie *Claim-* und *Allocation-Matrix*. Die Matrizen sind so auszufüllen, dass die Prozesse (= Aufträge) zeilenweise und die Ressourcen spaltenweise aufgezählt werden.

$$Resource = \begin{pmatrix} & & \end{pmatrix} \quad Available = \begin{pmatrix} & & \end{pmatrix}$$

$$Claim = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Resource Allocation Denial (10)

Verwenden Sie die Strategie des *Resource Allocation Denial*.
Beschreiben Sie für die gegebene Ausgangssituation mit Hilfe des Banker's-Algorithmus eine Lösung, bei der zuerst Auftrag 1, dann Auftrag 2 und zuletzt Auftrag 3 erfüllt werden.

Führen Sie alle Schritte bis zum Abschluss aller Aufträge durch und geben Sie zu jedem Schritt die zugehörige Claim- und Allocation-Matrix, sowie den Available-Vektor an.

Auftrag 1 wird durchgeführt:
(Alle für Auftrag 1 notwendigen Ressourcen sind in Verwendung):

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem Auftrag 1 beendet ist und Auftrag 2 durchgeführt wird:
(Alle für Auftrag 2 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem Auftrag 2 beendet ist und Auftrag 3 durchgeführt wird:
(Alle für Auftrag 3 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem alle Aufträge durchgeführt worden sind:

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

4 Security (25)

Security Threats (6)

Beschreiben Sie die Security Threats? Geben Sie jeweils ein Beispiel an.

Data Oriented Access Control (3)

Beschreiben Sie die *Data Oriented Access Control*? Welche Elemente hat das Modell?

Design for Security (4)

Was bedeutet das Prinzip *Open Design*? Erklären Sie anhand eines Beispiels die Vorteile des Open Design Prinzips.

Verschlüsselung (5)

Beschreiben Sie das Prinzip des Public-Key Verschlüsselungsverfahrens, sowie die Vor- und Nachteile gegenüber symmetrischen Verschlüsselungsverfahren.

Program Related Threats (2)

Beschreiben Sie 4 Program Related Threats.

Reference Monitor (1)

Was ist ein Reference Monitor?

Intrusion Detection (4)

Was ist *Intrusion Detection*, und wozu wird es verwendet?

Beschreiben Sie 3 *Intrusion Detection* Methoden, und geben Sie jeweils ein Beispiel an.