VO 182.022 29. Oktober 2003

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(35)	2.)(25)	3.)(20)	4.)(20)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (35)

Die Fähre *M/s Mariella* transportiert Fahrzeuge und Personen zwischen Helsinki und Stockholm. Um eine maximale Auslastung der Fähre zu erreichen, beschließt der Kapitän, die Fahrt erst dann zu beginnen, wenn die Fähre voll beladen ist.

Die Fähre besitzt Platz für 2500 Passagiere, 200 PKW und 30 LKW.

Implementieren Sie Synchronisationsroutinen für den Betrieb der Fähre. Berücksichtigen Sie dabei folgende Randbedingungen:

- Für die Verwendung von *Semaphoren* stehen Ihnen die Funktionen `initsem(`*`name`*`,` *`initialwert`*`)` und `P(`*`name`*`)` sowie `V(`*`name`*`)` zur Verfügung.
- Für die Verwendung von *Eventcountern* stehen Ihnen die Funktionen `initevc(`*`name`*`,` *`initialwert`*`)`, `read(`*`name`*`)`, `eawait(`*`name`*`)` und `advance(`*`name`*`,`*`value`*`)` zur Verfügung.
- Für die Verwendung von *Sequencern* stehen Ihnen die Funktionen `initseq(`*`name`*`,` *`initialwert`*`)`, `read(`*`name`*`)` und `sticket(`*`name`*`)`. Die Synchronisation mit globalen Variablen ist verboten.
- Verwenden Sie so wenig Synchronisationskonstrukte wie möglich.

1.1 Ressourcen anlegen und initialisieren

Die Initialisierung wird nur ein einziges Mal am Anfang aufgerufen. Bitte definieren Sie hier alle benötigten Ressourcen.

1.2 Kapitän

Der Kapitän legt mit dem Schiff an, koordiniert die Entladung der Fahrzeuge und schickt die Passagiere vom Schiff. Danach wartet der Kapitän bis das Schiff wieder voll beladen ist und die Fahrzeugrampe und der Passagierzugang geschlossen sind und legt dann wieder in Richtung Zielhafen ab. Das Beladen der Fähre mit PKW, LKW und das Aufnehmen von Personen soll gleichzeitig erfolgen.

```
do forever{

 Fahre_in_Hafen()

 Anlegen()

 Ankern()

 Installiere_Passagierrampe()

 Schicke_Passagiere_vom_Schiff()
```

```
 Oeffne_Fahrzeugrampe()

 Entlade_Fahrzeuge()
```

```
 Schliesse_Fahrzeugrampe()

 Entferne_Passagierrampe()

 Anker_lichten()

 Fahre_zum_Zielhafen()

}
```

1.3 PKW beladen

Dieses Programm dient zum Beladen der Fähre mit einem PKW. Es kann parallel mit anderen Programmen und mehrfach gestartet werden. Achten Sie darauf, dass die Fähre am Anlegeplatz steht, die Laderampe heruntergefahren ist und noch genug Platz auf der Fähre ist, andernfalls soll die Funktion blockieren, bis die nächste Fähre zum Beladen bereit ist. Diese Funktion kann gleichzeitig in mehreren Instanzen aufgerufen werden, wobei immer nur **ein** Parkvorgang (sowohl PKW als auch LKW) gleichzeitig durchgeführt werden können. Pro Fahrzeug muss jeweils ein Passagierplatz für den Fahrer reserviert werden.

```

Parke_PKW_auf_Fähre()

```

```
Fahrer_bezieht_Kabine()
```

1.4 LKW beladen

Dieses Programm dient zum Beladen der Fähre mit einem LKW. Es kann parallel mit anderen Programmen und mehrfach gestartet werden. Achten Sie darauf, dass die Fähre am Anlegeplatz steht, die Laderampe heruntergefahren ist und noch genug Platz auf der Fähre ist, andernfalls soll die Funktion blockieren, bis die nächste Fähre zum Beladen bereit ist. Diese Funktion kann gleichzeitig in mehreren Instanzen aufgerufen werden, wobei immer nur **ein** Parkvorgang (sowohl PKW als auch LKW) gleichzeitig durchgeführt werden darf. Pro Fahrzeug muss jeweils ein Passagierplatz für den Fahrer reserviert werden.

```

```

`Parke_LKW_auf_Fähre()`

```

```

`Fahrer_bezieht_Kabine()`

```

```

1.5 Neuer Passagier

Dieses Programm dient zum Aufnehmen eines neuen Passagiers. Ein neuer Passagier soll sich folgendermaßen verhalten: Ist er unter den glücklichen 2500 Passagieren, so versucht er an Bord zu gehen, ansonsten wartet er auf die nächste Fähre.

```

```

`Passagier_bezieht_Kabine()`

```

```

2 Scheduling (25)

Round-Robin und Feedback Scheduling

Schedulen Sie das nebenstehende Taskset. 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$= -1$; Zuteilung $= 0$). Ein eintreffender Task wird immer ans Ende der Ready Queue gestellt.

Task	Arrival Time	Service Time
A	-1	5
B	2	3
C	3	6
D	5	4
E	6	2

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 Task der als nächstes den Prozessor zugeteilt bekommt, usw., für Feedback-scheduling reihen Sie die höher prioren Queues zuerst). Schedulen Sie das Task Set nach der Round-Robin Methode und nach der Feedback Methode (time-slice $= 1$, die Anzahl der Queues für die Feedback Methode ist nicht beschränkt).

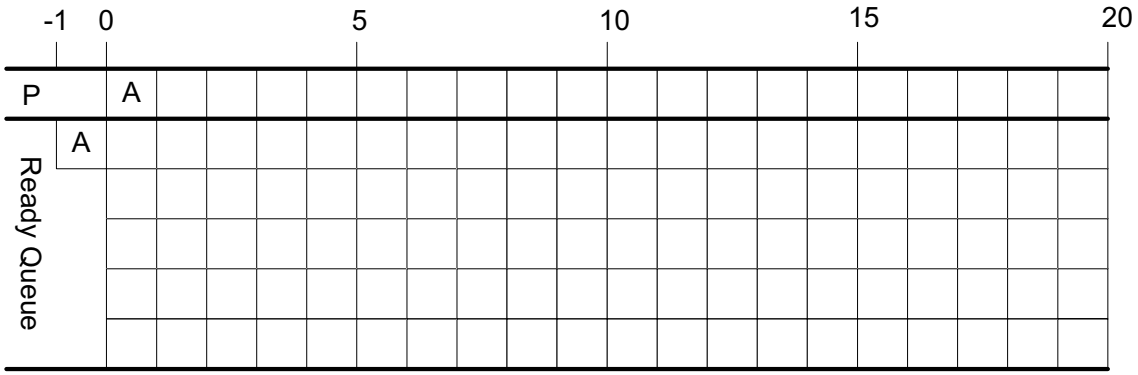


Abbildung 1: Round Robin Scheduling

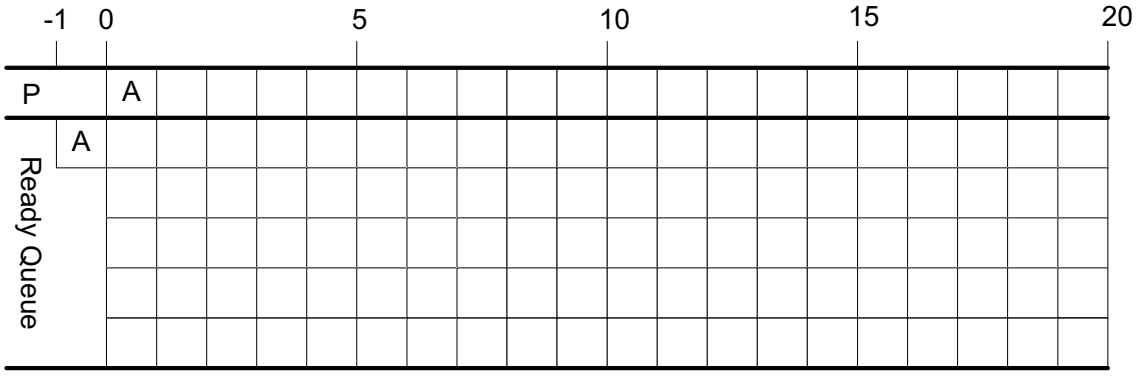


Abbildung 2: Feedback Scheduling

Rate-Monotonic Scheduling

Schedulen Sie das nebenstehende Taskset nach dem Rate-Monotonic Verfahren. Alle Tasks sind periodisch, wobei die Deadlines mit dem Ende der jeweiligen Periode gleichzusetzen sind. Der Overhead für den Taskwechsel ist vernachlässigbar.

Task	Ausführungszeit	Periodendauer
A	2	6
B	2	12
C	1	4
D	1	5

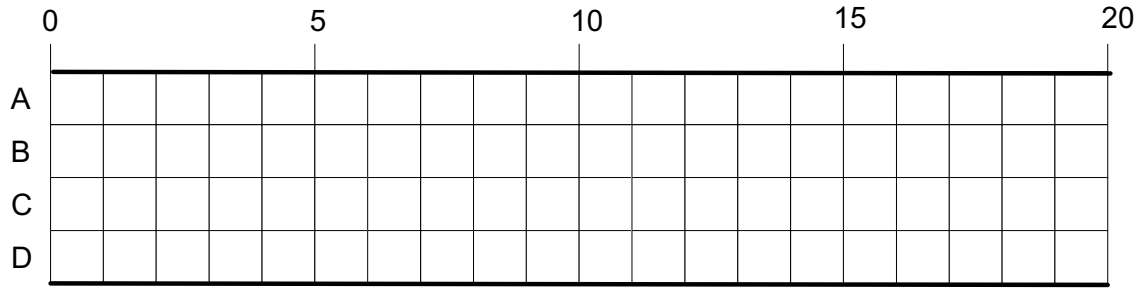


Abbildung 3: Rate-Monotonic Scheduling

Kann es bei diesem Taskset zu einem Deadline-Miss kommen (ja/nein/undefiniert), begründen Sie Ihre Antwort ?

Verständnisfragen

Was bedeutet Starvation?

Geben Sie 3 Scheduling Methoden an bei denen es zu Starvation kommen kann:

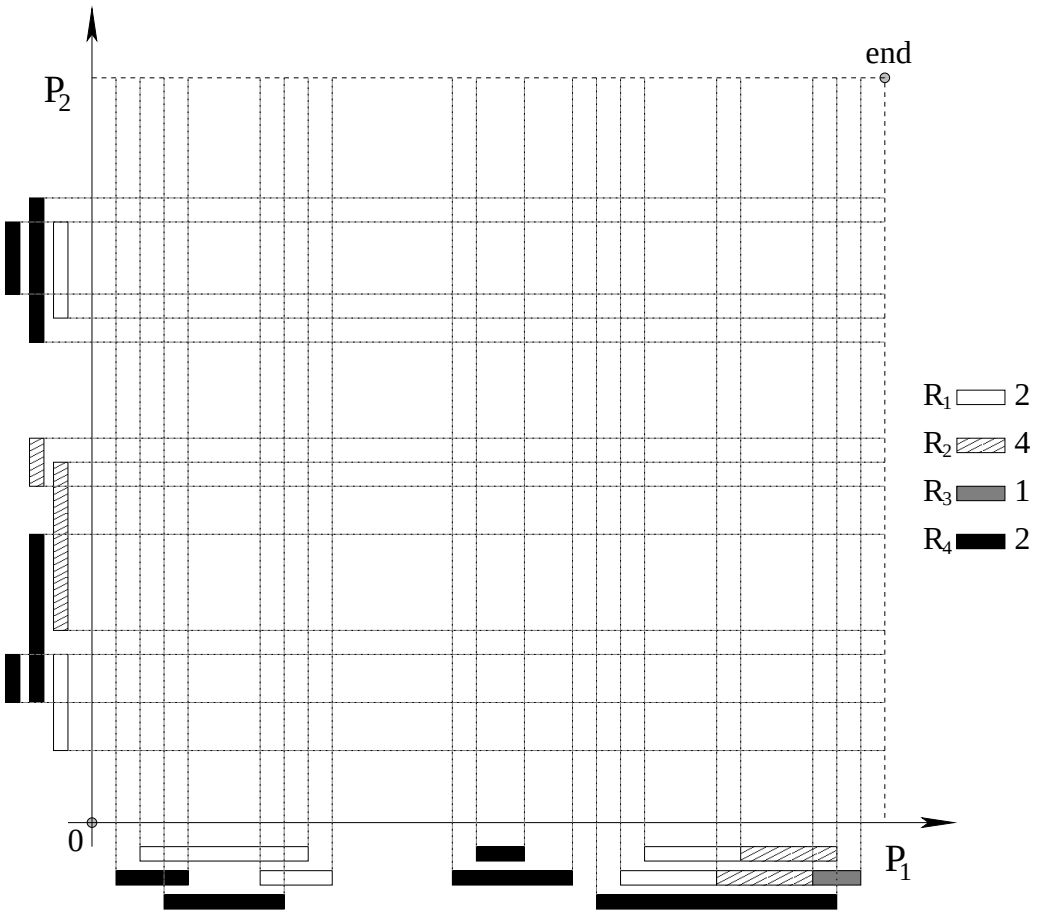
Erklären Sie das **gemeinsame** Problem dieser 3 Methoden, wieso es zu Starvation kommen kann:

3 Deadlock (20)

Zwei Prozesse P_1 und P_2 teilen sich gemeinsam Ressourcen des Systems. Der zum System gehörige Ressource Vector ist: $(R_1, R_2, R_3, R_4) := (2, 4, 1, 2)$. Werden von einem Prozess Ressourcen benötigt, welche der andere Prozess gerade belegt, so wird er mindestens bis zum Freiwerden der notwendigen Ressourcen verzögert. Welche Ressourcen von einem Prozess im Laufes seines „Lebens“ benötigt werden, entnehmen Sie bitte der Abbildung.

Durch einen Kantenzug von 0 nach end lässt sich nun der Fortschritt der beiden Prozesse bei der (quasi)parallelen Abarbeitung darstellen.

1. Umranden und schraffieren Sie in der Abbildung all jene Bereiche, durch welche ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen darf.
2. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht passieren kann ohne in einem Deadlock zu enden. Kennzeichen Sie diese Bereiche durch ein „D“.
3. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht erreichen kann obwohl kein Ressourcenkonflikt in diesem Bereich besteht. Kennzeichnen Sie diese Bereiche durch ein „U“.
4. Zeichen Sie einen deadlockfreien Kantenzug von 0 nach end ein.



4 Memory Management (20)

Ein Betriebssystem verwendet Speicherseiten der Größe 4KB ($2^{12} Bytes$). Die Seitentabelle ist als *Inverted Page Table (IPT)* realisiert, d.h., die Seitentabelle hat soviele Einträge, wie es Speicherframes im System gibt. In jedem der Einträge ist für die im entsprechenden Frame geladene Seite die Prozessnummer des Prozesses, zu dem diese Seite gehört, sowie die Seitennummer innerhalb des Prozesses gespeichert. Dabei werden in jedem Eintrag 8 Bits für die Prozessnummer und 8 weitere Bits für die Seitennummer verwendet.

Die Abbildung zeigt eine Folge von sechs Adressreferenzen bestehend aus jeweils Prozessnummer (PID) und logischer Adresse im Prozess. Diese Speicherzugriffe sind von oben nach unten nacheinander durchzuführen, wobei bei Page Faults die Clock Policy zum Replacement von Seiten angewandt wird. Die aktuellen Werte der Use Bits sind links neben der IPT angegeben, der Pfeil kennzeichnet die aktuelle Position des Clock-Zeigers (Laufrichtung des Zeigers ist zyklisch von oben nach unten).

Geben Sie in den Tabellen den Inhalt der IPT und der Use Bits, sowie den Clock-Zeiger nach jeder Speicherreferenz an. Tragen Sie außerdem in die Tabelle rechts oben für jeden Speicherzugriff die physikalische Adresse ein.