VO 182.022 04. November 2004

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronization (25)

In der Stadt *Peja* gibt es eine Bier-Fabrik. In der Fabrik will man den Bierabfüllungsprozess automatisieren, in dem man 4 Maschinen gekauft hat, die folgende Aufgaben (Funktionen) erfüllen:

Maschine 1: Bierflasche in die Produktionslinie platzieren - `flasche_platzieren()`

Bierflasche von der Produktionslinie entfernen - `flasche_entfernen()`

Maschine 2: Bierflasche mit Bier abfüllen - `flasche_abfüllen()`

Maschine 3: Bierflasche zumachen (mit Kronenkorken) - `flasche_zumachen()`

Maschine 4: Klebt Etikette an die Bierflasche - `etikette_kleben()`

Aufgrund der Ressourcenbeschränkungen kann jeweils nur 1 Flasche in die Produktionslinie platziert werden. Synchronisieren Sie die Arbeit der Maschinen effizient mittels Semaphore! Zusätzlich zu den Operatoren `P()` und `V()` stehen folgende Funktionen zur Verfügung:

- `INIT(sem, value)`: erzeugt und initialisiert den Semaphor `sem` mit dem Wert `value`
- `finshed()`: evaluiert zu `TRUE` wenn die Tagesschicht beendet wird, oder aus anderen Gründen der Abfüllungsprozess gestoppt werden muss. Wenn das der Fall ist `(finshed()=TRUE)`, dann soll die platzierte Flasche fertig abgefüllt, etikettiert, zugemacht und entfernt werden. Erst dann beenden die Maschinen ihre Arbeit.

Passen Sie auf, dass folgende Reihenfolge eingehalten wird:

`Flasche platzieren` -> `Flasche abfüllen` -> `Flasche zumachen` -> `Flasche entfernen`

Nachdem Flaschen platziert und bevor die Flaschen von der Produktionslinie entfernt werden, können die Etiketten geklebt werden. Achten Sie darauf, dass `flasche_abfüllen()` und `flasche_zumachen()` parallel zu der Funktion `etikette_kleben()` ausgeführt werden können. Um den Abfüllungsprozess zu optimieren, soll keine fixe Reihenfolge der Funktionen angegeben werden.

1.1 Initialisierung

1.2 Maschine1_Flaschen

1.3 Maschine2_Bier

1.4 Maschine3_Kronenkorken

1.5 Maschine4_Etiketten

2 Scheduling (25)

2.1 Uniprocessor Scheduling (13)

Gegeben ist nebenstehendes Taskset. Alle Tasks sind periodisch, wobei die Deadlines mit dem Ende der jeweiligen Periode gleichzusetzen sind. Der Overhead für den Taskwechsel ist vernachlässigbar.

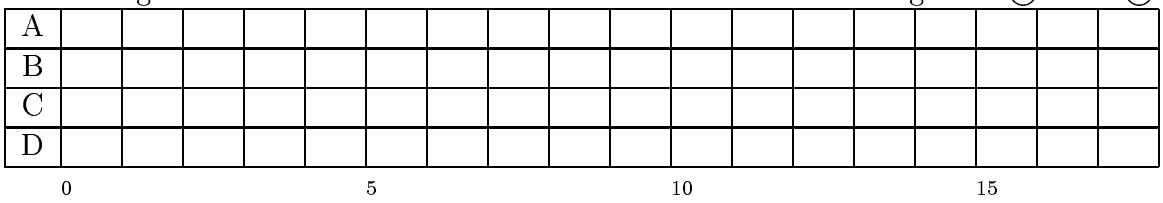
Task	Ausführungszeit	Periodendauer
A	1	5
B	2	9
C	2	7
D	1	4

Ermitteln Sie für dieses Taskset die *notwendige* und die *hinreichende* Bedingung für das *Rate Monotonic Scheduling* (RMS) Verfahren. Berechnen Sie die **Zahlenwerte** überschlagsmäßig (ohne Taschenrechner)!

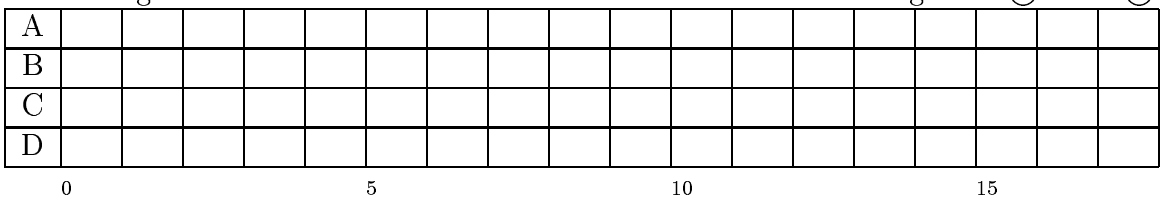
Ist die notwendige Bedingung erfüllt? ◯ Ja ◯ Nein
Ist die hinreichende Bedingung erfüllt? ◯ Ja ◯ Nein

Versuchen Sie das Taskset einmal mit dem RMS und einmal mit dem *Earliest-Deadline-First* (EDF) Verfahren zu schedulen. Verwenden Sie dazu die nachstehenden Vorlagen. Tragen Sie bei jeder Vorlage die aktiven Taskzeiten ein und bezeichnen Sie deutlich eventuelle Deadlineverletzungen. Kreuzen Sie jeweils an ob das Scheduling erfolgreich war. Eine Vorlage dient als Ersatz, streichen Sie gegebenenfalls eine falsch ausgefüllte Vorlage deutlich durch.

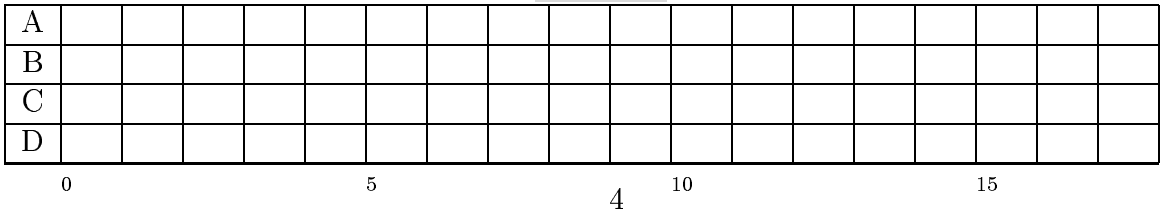
Scheduling nach dem **RMS**-Verfahren: Erfolgreich: ◯ Ja ◯ Nein



Scheduling nach dem **EDF**-Verfahren: Erfolgreich: ◯ Ja ◯ Nein



Ersatzvorlage: Scheduling nach dem -Verfahren: Erfolgreich: ◯ Ja ◯ Nein



2.2 Verständnisfragen (12)

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

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

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

◯ Ja ◯ Nein Beim Scheduling nach dem Round-Robin-Verfahren werden I/O-intensive Prozesse benachteiligt.

◯ Ja ◯ Nein Wenn in einem Single-Prozessor-System bei allen Tasks die Deadline gleich ihrer Periode ist, dann liefert RMS Scheduling dasselbe Ergebnis wie EDF Scheduling.

◯ Ja ◯ Nein Eine für das EDF Scheduling hinreichende Bedingung stellt auch eine hinreichende Bedingung für das RMS Scheduling dar.

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

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

◯ Ja ◯ Nein Beim Highest-Response-Ratio-Next-Verfahren kann das Problem der Starvation auftreten.

◯ Ja ◯ Nein Das Shortest-Remaining-Time-Verfahren ist immer preemptive.

◯ Ja ◯ Nein Das Shortest-Process-Next-Verfahren ist immer preemptive.

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

◯ Ja ◯ Nein Ein Scheduler nach dem Feedback Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Shortest-Process-Next-Verfahren.

◯ Ja ◯ Nein Earliest Deadline First Scheduling ist optimal in Multi-Prozessor Systemen.

◯ Ja ◯ Nein Das Round-Robin-Verfahren ist optimal, wenn die Periodendauer aller Tasks gleich ist.

◯ Ja ◯ Nein Ein Scheduler nach dem RMS Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Round-Robin-Verfahren.

3 Deadlock (25)

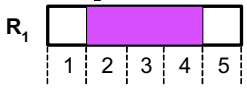
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *start* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehles.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Roussource bereits ab Start der Anweisung `get()` als allokiert gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt, wie im folgenden Beispiel verdeutlicht ist:

   ```
   2: get(R1)
   3: ...
   4: free(R1)
   ```

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht gehen kann.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.

5. Beschriften Sie jeweils ein Kästchen im Diagramm

   mit 'A', von welchem aus der Punkt *end* erreichbar ist

   mit 'B', welches unweigerlich zu einen Deadlock führt

   mit 'C', welches ein nicht erreichbarer Zustand ist

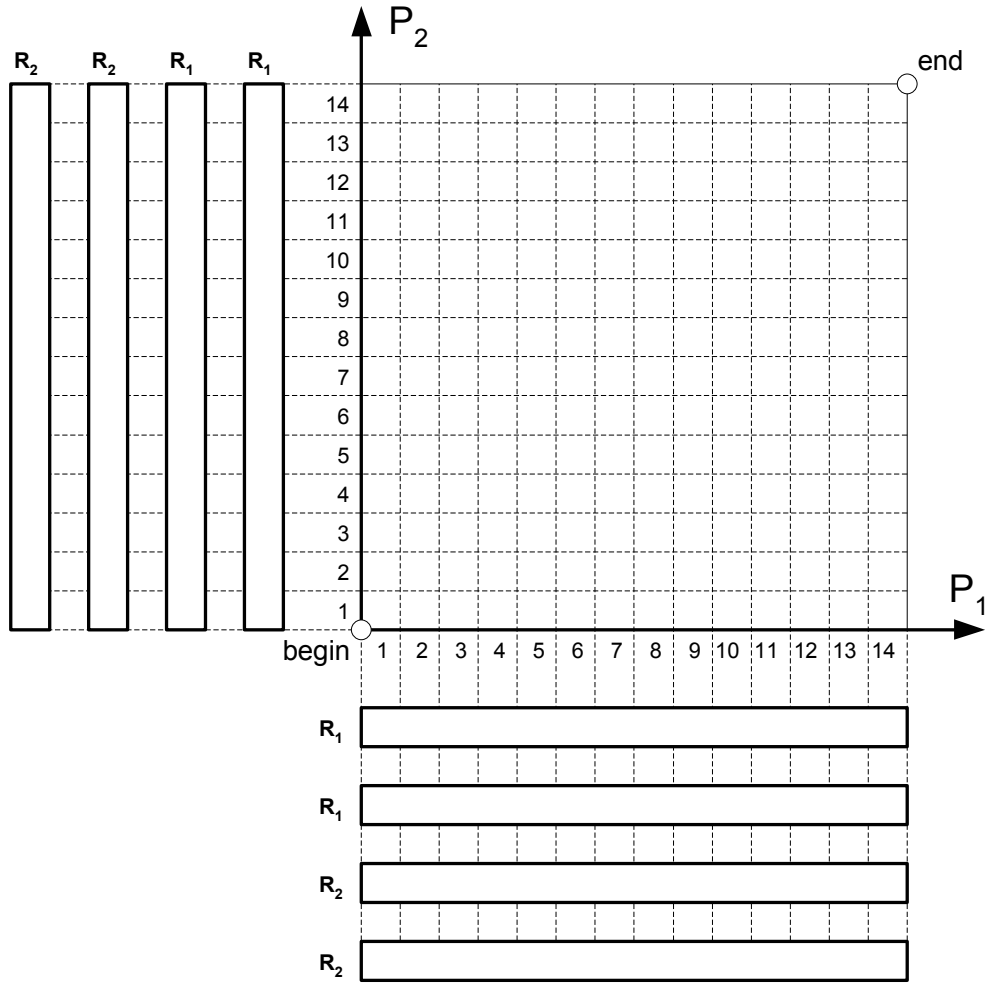
Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=5;
 2: get(R2);
 3: get(R2);
 4: b=a+6;
 5: a=a+4;
 6: get(R1);
 7: c=b+a;
 8: free(R2);
 9: a=0;
10: c=c/2;
11: b=0;
12: free(R1);
13: free(R2);
14: return;
```

Program P_2:

```
 1: a=0;
 2: get(R1);
 3: get(R1);
 4: b=a+2;
 5: get(R2);
 6: get(R2);
 7: free(R1);
 8: c=a+2;
 9: free(R1);
10: free(R2);
11: d=c;
12: free(R2);
13: a=d;
14: return;
```



4 Memory Management (25)

Bei einem Einprogrammsystem (Single Task System) kann man den Hauptspeicher in zwei Bereiche einteilen, den Speicher für das Betriebssystem (Kernel) und für das gerade ausgeführte Programm. Bei einem Mehrprogrammsystem (Multi Task System) ist der Hauptspeicher in mehrere Bereiche unterteilt. Die Verwaltung dieser Bereiche übernimmt eine sogenannte Speicherverwaltung.

Welche typischen Anforderungen stellt man an eine solche Speicherverwaltung?(3 Punkte)

Erklären Sie den Begriff *virtueller Speicher*. (3 Punkte)

Die zwei Grundtechniken um virtuellen Speicher zu realisieren sind *Segmentierung* und *Paging*. Erklären Sie die Gemeinsamkeiten, Unterschiede, Probleme, ... diese beiden Techniken. (8 Punkte)

Erklären Sie assoziative Zuordnung (associative mapping) und direkte Zuordnung (direct mapping). Geben Sie dazu jeweils ein Beispiel an. (5 Punkte)

Erklären Sie für ein einstufiges Paging-System die genauen Vorgänge bei einem TLB-Hit und TLB-Miss. Auf welche Speicherseiten wird zugegriffen, wie wird entschieden welcher Eintrag ausgewählt wird? (6 Punkte)