VO 182.022 26. Jänner 2005

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Scheduling (25)

Round-Robin und Feedback Scheduling

Task	Arrival Time	Service Time
A	0	6
B	1	2
C	3	3
D	4	6
E	8	1
F	16	1

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= 0; Zuteilung = 1). Ein eintreffender Task wird immer ans Ende der Ready Queue gestellt (bei Feedback Scheduling immer ans Ende der höchst-prioren Queue).

Tragen Sie in der Zeile **Exec.** 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/den Ready Queue(s) 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). Der Overhead für den Taskwechsel ist vernachlässigbar.

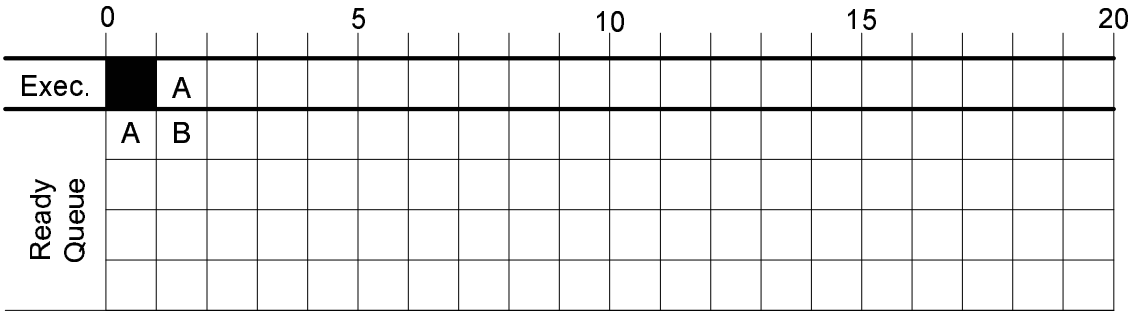


Abbildung 1: Round Robin Scheduling

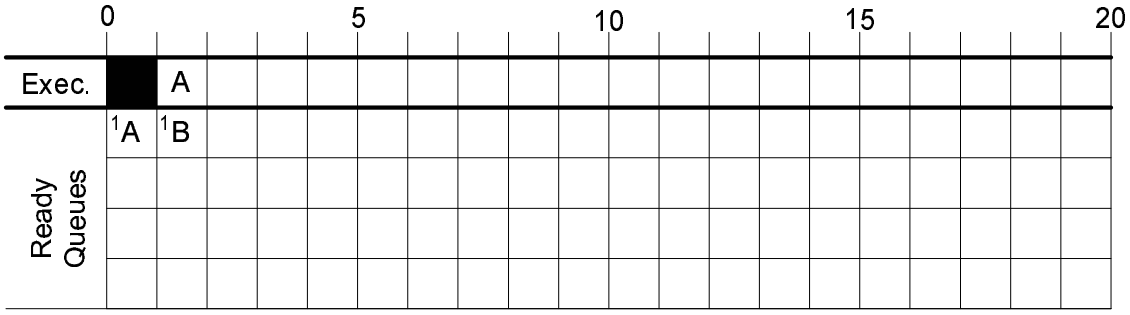


Abbildung 2: Feedback Scheduling

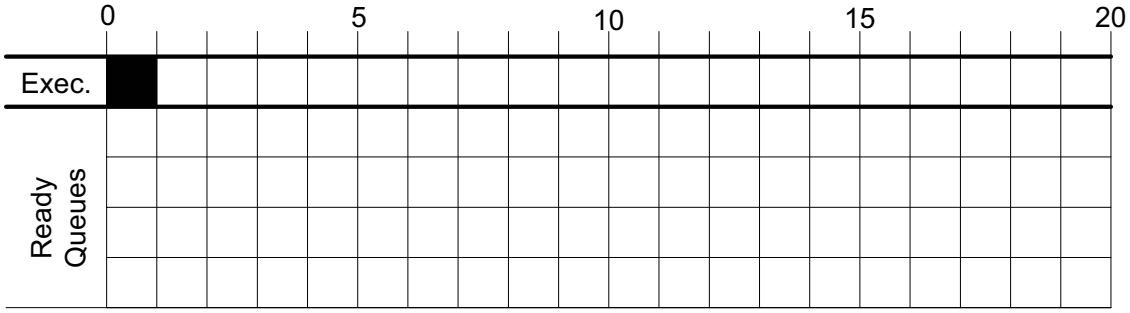


Abbildung 3: Ersatzraster

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. Gehen Sie davon aus, dass alle Tasks die erste Arrival Time 0 haben.

Task	Ausführungszeit	Periodendauer
A	1	4
B	3	10
C	2	5
D	1	16

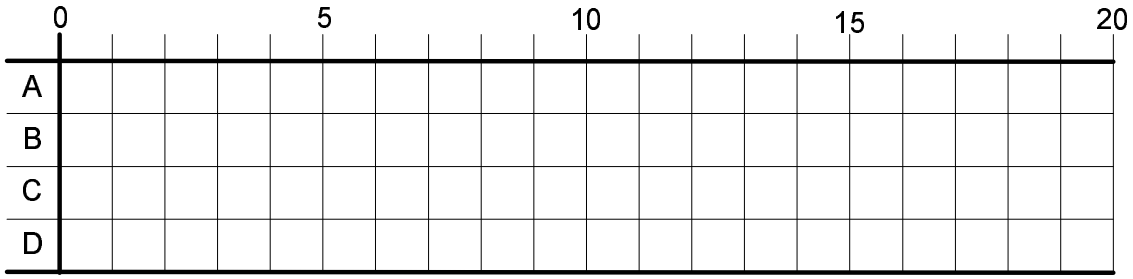


Abbildung 4: Rate-Monotonic Scheduling

2 Synchronisation (25)

Die folgenden Codestücke skizzieren eine Lösung des *Reader-Writer Problems*.

Initialisierungen

```
init(x,1); init(y,1); init(z,1);
init(wsem,1); init(rsem,1);
rc := 0; wc := 0;
```

Writer

```
 1   loop
 2       P(y);
 3       wc := wc + 1;
 4       if (wc == 1) then
 5           P(rsem);
 6       V(y);
 7
 8       write(...);
 9
10       P(y);
11       wc := wc - 1;
12       if (wc == 0) then
13           V(rsem);
14       V(y);
15   end loop;
```

Reader

```
21   loop
22       P(z);
23       P(rsem);
24       P(x);
25       rc := rc + 1;
26       if (rc == 1) then
27           P(wsem);
28       V(x);
29       V(rsem);
30       V(z);
31
32       read(...);
33
34       P(x);
35       rc := rc - 1;
36       if (rc == 0) then
37           V(wsem);
38       V(x);
39   end loop;
```

a) (8)

Beschreiben Sie kurz und prägnant für jeden Semaphor, welche Rolle er in der gegebenen Lösung spielt. Verwenden Sie, falls notwendig, die angegebenen Zeilennummern um Sich auf Stellen in den Codestücken zu beziehen.

x:

y:

z:

`rsem:`

`wsem:`

b) (17)

Wenn die maximale Anzahl von gleichzeitig aktiven *Reader*-Prozessen nach oben hin abgeschrankt wird, kann man eine Lösung für das *Reader-Writer Problem* angeben, die nur Semaphore (und keine globalen Variablen) zur Synchronisation verwendet. Schreiben Sie eine solche Lösung unter der Annahme, dass maximal K Reader gleichzeitig laufen. Sie brauchen in Ihrer Lösung nicht auf die Priorität von Lesern oder Schreibern zu achten.

Initialisierungen

Reader

Writer

3 Security (25)

3.1 Begriffe (10)

Was versteht man unter *Confidentiality (Secrecy)*, *Integrity* und *Availability*. Erklären Sie die drei Begriffe und geben Sie jeweils ein Beispiel einer Security Attacke an, welche die Eigenschaft verletzt.

- Confidentiality:

- Integrity:

- Availability:

Arten der Bedrohung (Types of Threats)

Füllen Sie folgende Tabelle derart aus, dass Sie jede angegebene Art der Bedrohung einem der Begriffe *Confidentiality (Secrecy)*, *Integrity* und *Availability* zuordnen (Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!):

Art der Bedrohung	**bedroht**
Interruption	
Interception	
Modification	
Fabrication	

3.2 Bedrohungen (8)

Erläutern Sie die folgenden Bedrohungen: *Logic Bomb*, *Trojan Horse*, *Virus* und *Worm*.

- Logic Bomb:

- Trojan Horse:

- Virus:

- Worm:

3.3 Verständnisfragen (7)

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

◯ Ja ◯ Nein *Network security* umfasst Maßnahmen zum Schutz der Daten während der Übertragung.

◯ Ja ◯ Nein Bei symmetrischen Crypto-Systemen werden zum Ver- und Entschlüsseln dieselben Keys verwendet.

◯ Ja ◯ Nein *Threshold detection* ist ein statistisches Verfahren zur Intrusion Detection.

◯ Ja ◯ Nein Das *Least Privilege* Prinzip besagt, dass jeder Benutzer alle Rechte per default hat.

◯ Ja ◯ Nein Bei *Masquerading* wird der Inhalt einer Message verändert.

◯ Ja ◯ Nein Eine *Trapdoor* ist ein geheimer Einstiegspunkt, der zur Umgehung der Zugriffskontrolle dient.

◯ Ja ◯ Nein Eine *denial-of-service* Attacke ist eine passive Attacke.

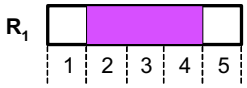
4 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. Ressource R_1 und R_2 sind einmal vorhanden. Ressource R_3 ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar.

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 (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

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 ein solcher Kantenzug 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. Dieser Kantenzug soll durch möglichst viele Punkte (S_1-S_6) führen.

5. Entscheiden Sie für jeden der 6 Punkte (S_1-S_6) im Diagramm, ob der Punkt erreicht werden kann, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	◯	◯
S_2	◯	◯
S_3	◯	◯
S_4	◯	◯
S_5	◯	◯
S_6	◯	◯

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=0;
 2: get(R2);
 3: b=a+2;
 4: a=a+3;
 5: get(R3);
 6: get(R1);
 7: c=b*a;
 8: free(R1);
 9: get(R3);
10: b=a+b;
11: free(R2);
12: free(R3);
13: free(R3);
14: return;
```

Program P_2:

```
 1: get(R3);
 2: a=0;
 3: get(R1);
 4: get(R3);
 5: b=0;
 6: get(R2);
 7: free(R3);
 8: c=a+b;
 9: free(R2);
10: d=c;
11: free(R3);
12: free(R1);
13: a=d+2;
14: return;
```