VO 182.022 17. November 2006

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

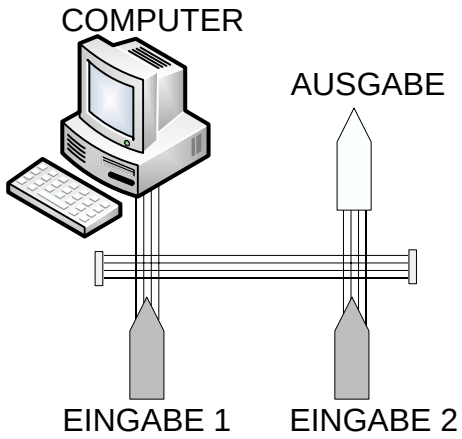
Ges.)(100)	1.)(30)	2.)(25)	3.)(23)	4.)(22)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (25)

Ein Computerprogramm simuliert das Verhalten eines Systems, das aus den folgenden Einheiten besteht:

- ein Computer
- zwei Eingabegeräte (Input),
- ein Ausgabegerät (Output), und
- ein Bus-Netzwerk, das den Computer mit den Eingabe- und Ausgabegeräten verbindet.



Das Programm simuliert die Funktionalität des Computersystems mittels verschiedener Prozesse, wobei ein Prozess das Verhalten des Computers, ein Prozess das Verhalten eines Eingabegerätes und ein Prozess das Verhalten des Ausgabegerätes simuliert. Diese Prozesse sollen folgende Funktionalität implementieren:

- Der Eingabeprozess liest die Daten mittels Funktion *read_ input_ data()* aus dem Eingabegerät und schickt die Daten mittels Funktion *send_ input_ data()* zum Computer sobald das Netwerk frei von Nachrichten ist.
- Der Computerprozess liest die emfangenen Daten mittels Funktion *PC_ read_ input_ data()*.

- Der Computer bearbeitet die gelesenen Daten mittels Funktion *process_ data()* und sendet die abgearbeiteten Daten durch das Netzwerk mittels Funktion *PC_ send_ output_ data()* zum Ausgabeprozess.
- Der Ausgabeprozess liest die empfangenen Daten mittels Funktion *rcv_ output_ data()* und verwendet sie mittels Funktion *apply_ output_ data()* am Ausgabegerät.
- Der Computer und der Ausgabeprozess können die Daten der Eingabeprozesse hintereinander bearbeiten.
- Der Computerprozess und die Eingabeprozesse dürfen das Netzwerk zum Senden der Daten nicht gleichzeitig benutzen.
- Die zwei Eingabegeprozesse sind identisch und arbeiten unabhängig voneinander.

Passen Sie auf, dass folgende Reihenfolge eingehalten wird:

read_ input_ data -> send_ input_ data ->

PC_ read_ input_ data -> process_ data -> PC_ send_ output_ data ->

rcv_ output_ data -> apply_ output_ data

Synchronisieren Sie den Arbeitsablauf der Prozesse mit Semaphoren. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Verwenden Sie folgende Funktionen für Operationen auf Semaphoren:

`initS(`***Sem,init***`)` legt einen Semaphor mit dem angegebenen Namen *Sem* an und initialisiert ihn mit der Zahl *init*.

P(*Sem*) implementiert ein *wait* auf dem Samaphore, und

V(*Sem*) implementiert ein *signal* auf dem Samaphore.

a) Initialisierungen

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** ist wie folgt:

- keine Daten im Eigabengerät,
- keine Daten im Ausgabegerät,
- das Netzwerk und ist frei von Nachrichten, und
- der Computer hat keine Daten zu bearbeiten.

Entwerfen Sie je einen Prozess für *Eingabegerät 1* und *2*

Prozess *Eingabegerät 1*:

```
do forever() {

}
```

Prozess *Eingabegerät 2*:

```
do forever() {

}
```

Entwerfen Sie die Prozesse für den *Computer* und das *Ausgabegerät*

Prozess *Computer*:

```
do forever() {

}
```

Prozess *Ausgabegerät*:

```
do forever() {

}
```

2 Uniprocessor Scheduling (20)

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.

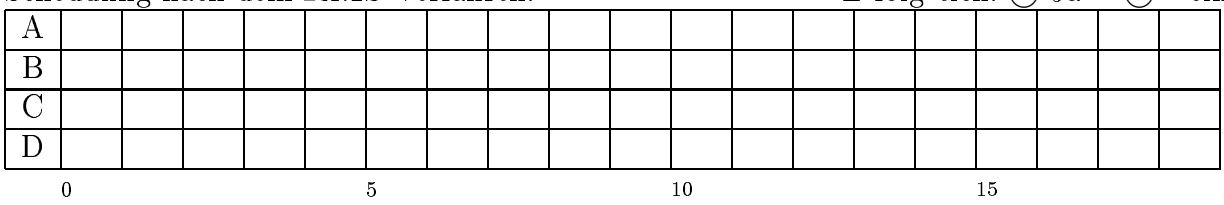
Task	Ausführungszeit	Periodendauer
A	2	4
B	1	6
C	1	7
D	1	9

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)!

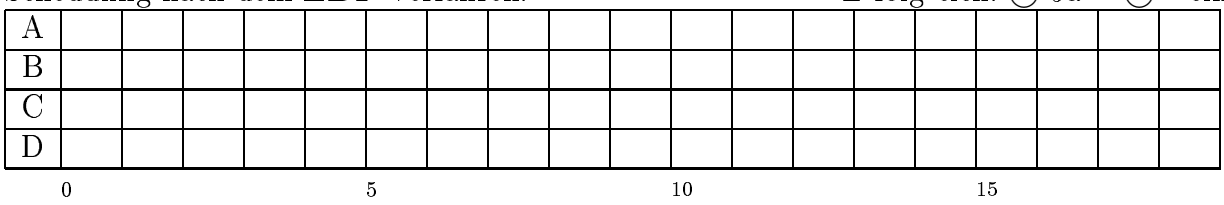
Ist die notwendige Bedingung erfüllt? ◯ Ja ◯ Nein
Ist die hinreichende Bedingung erfüllt? ◯ Ja ◯ Nein

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.

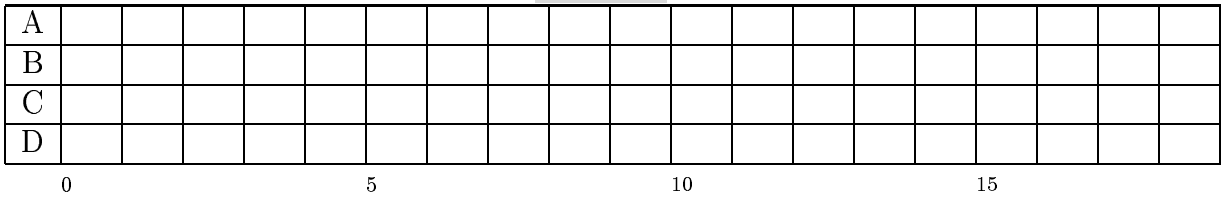
Scheduling nach dem **RMS**-Verfahren: Erfolgreich: ◯ Ja ◯ Nein



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



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



3 Security (25)

3.1 Nennen und erklären Sie die Threats of Security. Geben Sie jeweils ein konkretes Beispiel an! (5)

3.2 Erklären Sie die Funktionsweise von Kerberos! (5)

3.3 Wie funktioniert Public Key Encryption? (5)

3.4 Beschreiben Sie das UNIX Passwort Scheme! (4)

3.5 Erläutern Sie zwei Ansätze zum Angriff auf ein konventionelles Encryption Scheme! (6)

4 Speicherverwaltung (22)

Das betrachtete Speicherverwaltungssystem verwendet zur Adressierung 16-Bit Adressen. Für die angegebenen virtuellen Speicheradressen sind, in Abhängigkeit von der Adressierungsart, die entsprechenden physikalischen Adressen zu ermitteln. Von den angebenen Adressen sind die niederwertigen 8 Bit der Offset der Adresse und die höherwertigen 8 Bit die Segmentnummer bzw. die Seitennummer. Bei Paging sind alle Seiten 256 Bytes (Hexadezimal 0x100) lang. Alle Werte mit führendem **0x** sind als Hexadezimalzahl angegeben, Werte mit abschließendem **b** sind als Binärzahl zu interpretieren. Ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld.

Es werden folgende Begriffe (englische Notation) verwendet:

Base	Basisadresse des Segmentes
Entry	Eintrag in der Adressübersetzungstabelle
Frame#	Seitenrahmennummer (im physischen Speicher)
Length	Länge des Segmentes
Page#	Seitennummer (im virtuellen Speicher)

a) Paging — Assoziativer Zugriff (associative mapping)

Adressübersetzungstabelle:

Page#	Frame#
00000101b	0x11
00100011b	0x01
00000001b	0x30
00110000b	0x41
00010001b	0x02

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x1541	
0x0110	
0x3105	
0x0502	

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:

Entry	Frame#
0	0x13
1	0x08
2	0x34
3	0x31
4	0x01
5	0x68
6	0xA0

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x1306	
0x0511	
0x028A	
0x0401	

c) Segmentierung — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:

Entry	Base	Length
0	0x2840	0x004
1	0x0563	0x0FF
2	0x72AB	0x100
3	0x0300	0x010
4	0x11FD	0x0F0
5	0x4444	0x030
6	0x3112	0x060
7	0x4220	0x010

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x04A0	
0x4222	
0x0010	
0x0305	
0x01F2	
0x0650	

d) Verständnisfragen

Kreuzen Sie bitte die richtigen Antworten an! Achtung! Falsche Antworten werden negativ gewertet!

- Welche Aussagen treffen beim *Fixed Partitioning* zu?
  - ◯ Die Partitionen sind immer von gleicher Größe
  - ◯ Der vorhandene Speicher wird uneffizient ausgenutzt
  - ◯ Das Problem der Internal Fragmentation tritt auf

- In einem fehlerfreien System können zwei oder mehr virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
  - ◯ Richtig
  - ◯ Falsch

- Zu welchen Effekten kann es bei Paging kommen?
  - ◯ Unterschiedliche Längen der Pages
  - ◯ Internal Fragmentation
  - ◯ External Fragmentation

- Bei Speicherverwaltung mittels Segmentierung berechnet sich die physikalische Adresse aus der virtuellen Adresse durch:
  - ◯ Addition der physikalischen Segmentstartadresse mit der virtuellen Adresse
  - ◯ Multiplikation der physikalischen Segmentstartadresse mit dem Offset-Teil der virtuellen Adresse
  - ◯ Addition der physikalischen Segmentstartadresse mit dem Offset-Teil der virtuellen Adresse
  - ◯ Ersetzen der Seitennummer in der virtuellen Adresse durch die physikalische Segmentstartadresse