VO 182.022 20. Oktober 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 Synchronisation (25)

In einem Computersystem, das zur Vereinfachung nur ein relevantes Register besitzt, soll eine Folge von Instruktionen mit möglichst hoher Parallelität ausgeführt werden. Es gibt zwei Arten von Operationen auf dem Register: Operationen, die das Register lesen und dann eine Funktion auf dem Registerwert ausführen (kurz Leseoperationen genannt), und Operationen, die einen neuen Wert in das Register schreiben (Schreiboperationen).

Ein Programm für das Computersystem besteht aus einer (unendlichen) Folge von Lese- bzw. Schreiboperationen, die vom Computersystem unter folgenden Einschränkungen abgearbeitet werden.

- Verschiedene Leseoperationen des Programms (vom Typ `READ_INSTRUCTION`) können in beliebiger Reihenfolge und insbesondere auch gleichzeitig abgearbeitet werden.
- Eine Schreiboperation (Typ `WRITE_INSTRUCTION`) und eine beliebige andere Instruktion (Lese- oder Schreiboperation) müssen immer in der Reihenfolge abgearbeitet werden, in der sie im Programmcode stehen (d.h., steht eine Schreiboperation S1 im Code an irgendeiner Stelle vor einer anderen Operation O2, so muss S1 auch vor O2 abgearbeitet werden; genauso muss jede Operation O1, die im Code irgendwo vor einer Schreiboperation S2 steht, vor S2 abgearbeitet werden).

a) Ergänzung von Synchronisationskonstrukten

Das Computersystem arbeitet das Programm unter höchst möglicher Parallelität ab, indem es für jede Instruktion einen eigenen Prozess erzeugt. Dieser Prozess liest die Instruktion ein, bestimmt, ob es sich um eine Lese- oder Schreiboperation handelt, und führt die Operation schliesslich unter Beachtung der oben genannten Einschränkungen mit Hilfe der Funktionen `execute_read_instruction()` bzw. `execute_write_instruction()` aus.

Ergänzen Sie das folgende Stück Pseudocode für die Prozesse zur Instruktionsausführung so mit geeigneten Synchronisationskonstrukten, dass die korrekte Instruktionsabarbeitung laut der oben angegebenen Regeln gesichert ist. Geben Sie geeignete Initialisierungen der Synchronisationskonstrukte an.

Code fuer Prozess zur Instruktionsausfuehrung:

```
    instruktion = get_next_instruction_from_file()

    if (instruction_type(instruktion) == READ_INSTRUCTION)
    {   Behandlung fuer Leseinstruktion

        execute_read_instruction(instruktion)

    } else {    Behandlung fuer Schreibinstruktion (WRITE_INSTRUCTION)

        execute_write_instruction(instruktion)
```

```
}
```

Initialisierungen:

```
```

2 Scheduling (25)

2.1 Round-Robin Scheduling 10

Schedulen Sie das folgende Taskset mittels Round-Robin und einer Zeitscheibenlänge (time-slice) gleich 1. Der Overhead für den Taskwechsel ist vernachlässigbar.

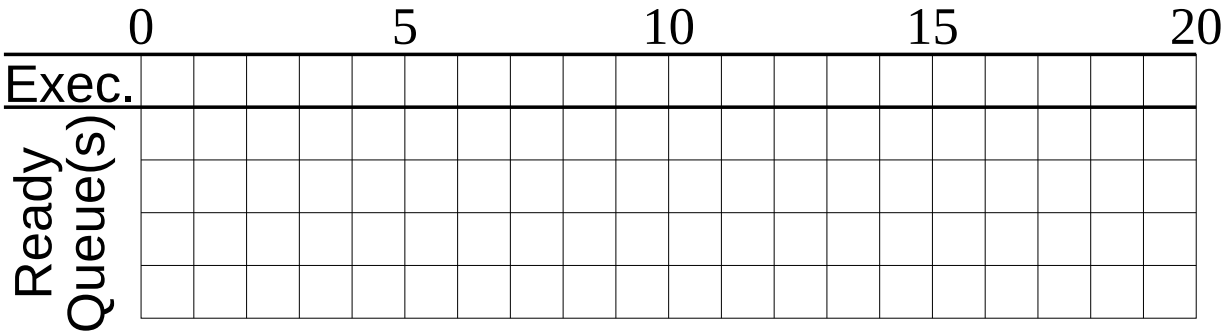
Das Scheduling soll in **folgender Reihenfolge** ablaufen (Algorithmus):

1. Neue Tasks an das Ende der Ready Queue stellen
2. Den zuletzt ausgeführten Task an das Ende der Ready Queue stellen
3. Den vordersten Task der Ready Queue ausführen (`Exec.`)

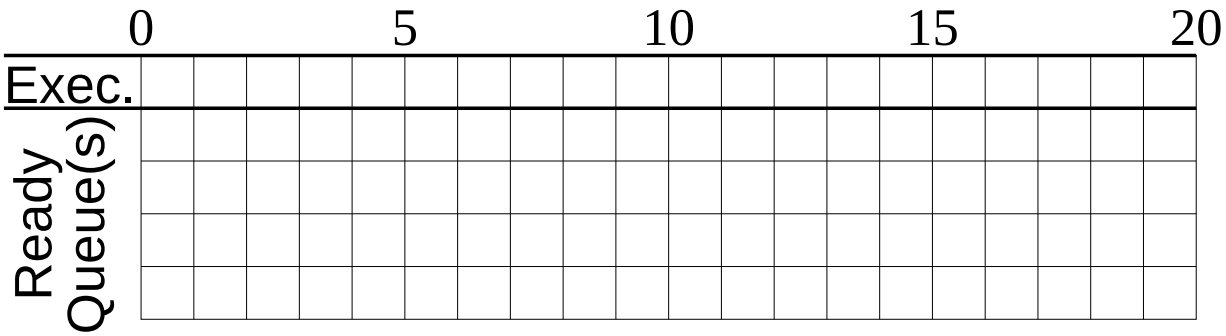
Task	Arrival Time	Execution Time
A	0	3
B	1	6
C	3	3
D	5	5
E	9	1
F	15	2

Ein Task kann also zum Zeitpunkt des Arrivals bereits ausgeführt werden (Beispiel siehe Task A: arrival time= 0; Zuteilung = 0).

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.



Ersatzvorlage:



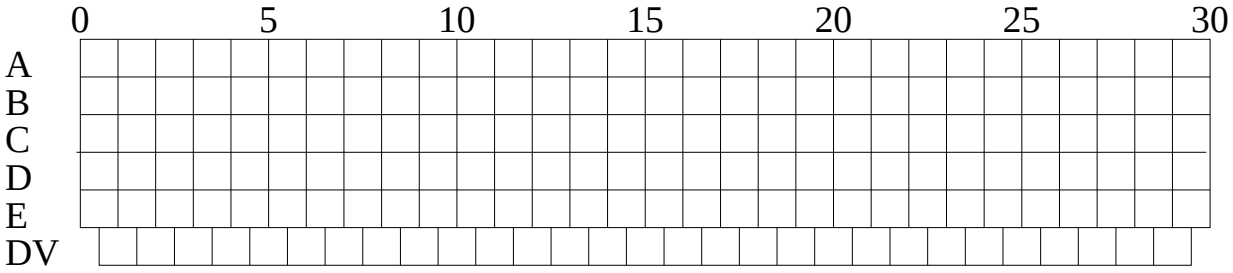
2.2 Earliest Deadline First Scheduling 12

Schedulen Sie das nebenstehende Taskset mittels Earliest Deadline First (EDF). Der Overhead für den Taskwechsel ist vernachlässigbar.

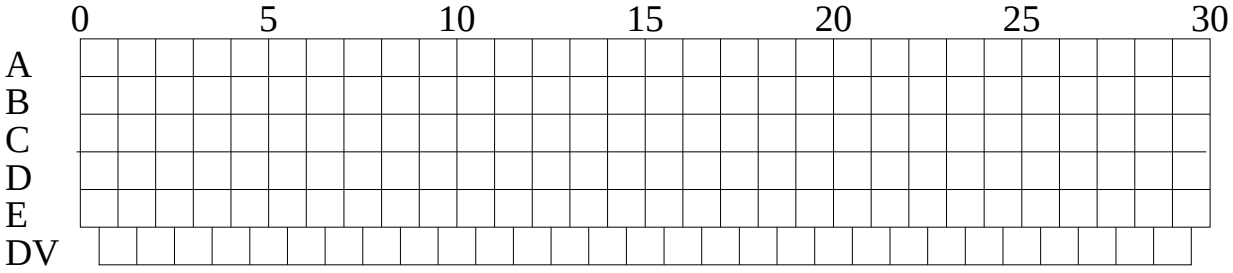
Task	Arrival Time (ms)	Execution Time (ms)	Deadline (ms)
A	0	10	20
B	2	3	10
C	3	4	9
D	6	1	7
E	16	3	19

Ein Task kann bereits zum Zeitpunkt des Arrivals ausgeführt werden (Beispiel siehe Task A: arrival time = 0; Zuteilung = 0).

Verfollständigen Sie folgendes Diagramm bis alle Tasks gescheduled worden sind bzw. bis es zu einer Deadline-Verletzung kommt. Tragen Sie im Falle einer Deadline-Verletzung zum betreffenden Zeitpunkt den jeweiligen Task in der Zeile `DV` ein.



Ersatzvorlage:



Bitte beantworten Sie diese Frage unabhängig von Ihrer Lösung oberhalb: Nehmen Sie an, dass das obige Taskset gescheduled werden kann. Bis zu welchem Zeitpunkt müssen Sie das angegebene Taskset *mindestens* schedulen, um sicher gehen zu können, dass es wirklich schedulbar ist. Bitte richtige Antworten ankreuzen:

◯ 10 ms ◯ 16 ms ◯ 18 ms ◯ 19 ms ◯ 20 ms ◯ 21 ms ◯ 22 ms
◯ 23 ms ◯ 38 ms ◯ einen Zeitpunkt der in der Liste nicht aufscheint

2.3 Allgemeine Fragen zu Scheduling 3

Bewertung: falsche Antwort: -2, keine Antwort: -1

Die Verwendung von Preemtion bei Scheduling erhöht den Durchsatz des Systems:

◯ korrekt ◯ inkorrekt

Die Verwendung von First-Come-First-Served (FCFS) kann im ungünstigen Fall zu Starvation führen:

◯ korrekt ◯ inkorrekt

Die Verwendung von Round-Robin kann es zu keiner Starvation kommen:

◯ korrekt ◯ inkorrekt

3 Deadlock (25)

Bedingungen für Deadlock (6)

Erklären Sie die für das Auftreten eines Deadlocks notwendigen und hinreichenden Bedingungen.

Deadlock-Vermeidung (19)

Die Firma InTheRed hat unter ihren Angestellten 5 Ingenieure (`I`), 2 Tester (`T`) und 3 Patentanwälte (`P`) und bearbeitet derzeit drei Projekte, FIZZLE, T&E und HOTAIR.

Projektdaten:

Projektname:	FIZZLE
Benötigte Ressourcen:	4 Ingenieure, 2 Tester, 2 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur
Projektname:	T&E
Benötigte Ressourcen:	4 Ingenieure, 2 Tester, 0 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur, 1 Tester
Projektname:	HOTAIR
Benötigte Ressourcen:	2 Ingenieure, 1 Tester, 3 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur, 1 Tester, 1 Patentanwalt

Tragen Sie den Resource-Vektor, die Claim-Matrix und die Allocation-Matrix ein. Berechnen Sie auch den Availability-Vektor.

$$Resource = \begin{pmatrix} \texttt{I} & \\ \texttt{T} & \\ \texttt{P} & \end{pmatrix} \qquad Claim = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

$$Available = \begin{pmatrix} \texttt{I} & \\ \texttt{T} & \\ \texttt{P} & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Geben Sie in Ihrer Lösung die einzelnen Schritte der Ausführung des Deadlockerkennungsalgorithmus an.

Verwenden Sie nun den *Bankier-Algorithmus* (Banker's Algorithm) und geben Sie für *jeden* Schritt die Claim- und Allocationmatrix, sowie den Availability Vector und das als nächstes durchzuführende Projekt an. Wenn kein Projekt mehr auszuführen ist, dann schreiben sie 'fertig', falls ein Deadlock auftritt, schreiben Sie 'Deadlock' in den nächsten Schritt.

`Nächstes Projekt/fertig/Deadlock:`

`Alle für das Projekt benötigten Ressourcen sind in Verwendung:`

$$Claim = \begin{pmatrix} \texttt{I} & & & \\ \texttt{T} & & & \\ \texttt{P} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

`Nach der Ausführung des letzten Projektes:`

$$Claim = \begin{pmatrix} \texttt{I} & & & \\ \texttt{T} & & & \\ \texttt{P} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Projekt/fertig/Deadlock:

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{I} \\ \texttt{T} \\ \texttt{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

Nach der Ausführung des letzten Projektes:

$$Claim = \begin{pmatrix} \texttt{I} \\ \texttt{T} \\ \texttt{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

Nächstes Projekt/fertig/Deadlock:

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{I} \\ \texttt{T} \\ \texttt{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

Nach der Ausführung des letzten Projektes:

$$Claim = \begin{pmatrix} \texttt{I} \\ \texttt{T} \\ \texttt{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

4 Memory Management (25)

a) Virtueller Speicher mit Kombination aus Segmentierung und Paging 14

In folgendem Beispiel sollen Sie ausgehend von virtuellen Adressen die physikalischen Adressen bestimmen. Es werden folgende Begriffe (englische Notation) aus dem Buch zur Vorlesung verwendet:

Base	Basisadresse Seitentabelle des Segmentes
Length	Länge des Segmentes (Anzahl der Seiten des Segmentes)
Virt.Addr.	Virtuelle Adresse
Frame#	Seitenrahmennummer (im physischen Speicher)
Page#	Seitennummer (im virtuellen Speicher)
Seg#	Segmentnummer

Es handelt sich bei dem System um ein System mit 32-Bit-Adressen, wobei die niederwertigen 16 Bit immer den Offset einer Adresse bilden und die höherwertigen 16 Bit Segment- und Seitennummer bilden:

Seg# (8 bit)	Page# (8 bit)	Offset (16 bit)

Es wird dabei direkter Zugriff (direct mapping) sowohl auf die Segmenttabelle als auch auf die Seitentabelle verwendet.

Bei Paging sind alle Seiten 65536 Bytes (hexadezimal `0x0001 0000`) lang. Alle Werte sind als Hexadezimalzahlen angegeben. Ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld.

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle:

Segmenttabelle		
Seg#	Base	Length
0x00	0x0827 7234	0x03
0x01	0x1043 6073	0x01
0x02	0x4074 8054	0x02
0x03	0x5322 23AA	0x10
0x04	0x3012 B578	0x0A

und folgende Seitentabelle:

Seitentabelle	
Address	Frame#
...	...
0x0827 7234	0x6752
0x0827 7235	0x7613
0x0827 7236	0x258A
0x0827 7237	0xB3CA
...	...
0x1043 6073	0x56EF
0x1043 6074	0x4567
...	...
0x3012 B57F	0x5014
0x3012 B580	0x5109
0x3012 B581	0xA145
0x3012 B582	0x2000
0x3012 B583	0x3234
...	...
0x4074 8054	0x7353
0x4074 8055	0xBABA
0x4074 8056	0x9789
0x4074 8057	0xAFFE
...	...
0x5322 23B7	0x5400
0x5322 23B8	0x5401
0x5322 23B9	0x0945
0x5322 23BA	0x1313
...	...

Ermitteln Sie unter Benützung obiger Tabellen die physikalischen Adressen zu den folgenden virtuellen Adressen. Markieren Sie die den Eintrag mit "invalid segment nr." bzw. "invalid page nr." im Fehlerfall.

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0x030F 01CE	
0x0001 04B3	
0x0101 2019	
0x0409 1025	
0x0539 0715	
0x0407 197A	
0x0310 9788	

Bitte beachten Sie, dass bei diesem Beispiel falsche Antworten zu Punkteabzug führen.

b) Speicherfragmentierung 11

Geben Sie Beispiele von Speicherbelegungen für die jeweiligen Fragmentierungswerte an. Der Speicher besteht dabei aus durchnummerierten Speicherblöcken. Die allokierten Speicherbereiche sind immer auf die Grenzen von Speicherblöcken ausgerichtet. Beachten Sie dabei, dass die Beispiele so gewählt worden sind, dass die Aufteilung in Fragmentierungsbereiche ganzzahlig möglich ist. Nicht ganzzahlige Lösungen werden nicht gewertet!

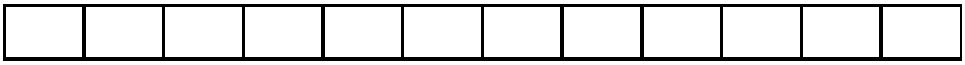
Markieren Sie die Speicherbereiche dabei folgendermaßen:

 ... Nutzdaten (Granularität: 1 Block)

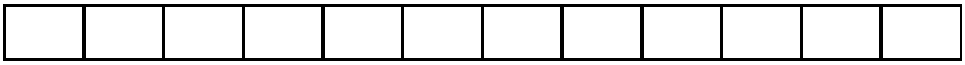
 ... Interne Fragmentierung (angegeben in % relativ zum allokierten Speicher)

 ... Externe Fragmentierung (angegeben in % relativ zum Gesamtspeicher)

internal fragmentation: 0%, **external fragmentation:** 50%, **Größe der Allokationseinheit: 3 Blöcke**



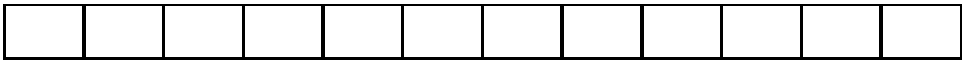
Ersatzvorlage:



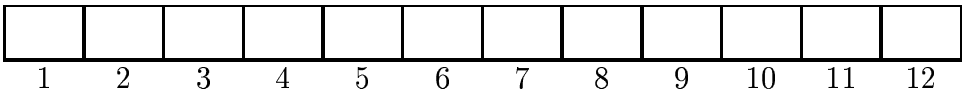
internal fragmentation: 33.3%, **external fragmentation:** 0%, **Größe der**

Allokationseinheit: **Blöcke (frei wählbar)**

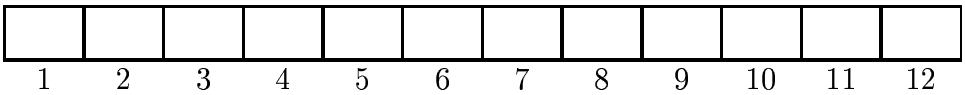
Ersatzvorlage:



internal fragmentation (if) = 50%, external fragmentation (ef) = 33.3%, Größe der Allokationseinheit: 4 Blöcke



Ersatzvorlage:



Nennen Sie eine Speicherverwaltungstechnik, bei der ausschließlich interne Fragmentierung auftreten kann:

Nennen Sie eine Speicherverwaltungstechnik, bei der ausschließlich externe Fragmentierung auftreten kann:

Nennen Sie eine Speicherverwaltungstechnik, bei der sowohl interne wie auch externe Fragmentierung auftreten kann: