LVA: Betriebssysteme Preis: 0,00 € Seiten: 17

Vorlesungsprüfung aus Betriebssysteme 2003-01-22

KNr. | MNr. | Zuname, Vorname

(00) | 1.)(25) | 2.)(30) | 3.)(23) | 4.)(22) | Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

Deadlock (25)

egeben sind zwei Prozesse P_1, P_2 und zwei Ressourcen R_1, R_2. Die Ressourceallokationen r Prozesse in Abhängigkeit des jeweiligen Prozessfortschritts sind in Tabelle 1 eingetra-

Zeit	P1	P2
t=0		
t=1	P(R1)	P(R1)
t=2	P(R1)	P(R1)
t=3	P(R1)	
t=4		P(R2)
t=5		
t=6	V(R1)	
t=7	V(R1)	V(R1)
t=8		V(R1)
t=9	V(R1)	
t=10		
t=11	P(R2)	P(R1)
t=12	P(R2)	P(R1)
t=13		P(R1)
t=14	V(R2)	V(R1)
t=15	V(R2)	V(R1)
t=16		V(R1)
t=17		
t=18		V(R2)
t=19		
t=20	Termination	Termination

Tabelle 1: Prozess P1 und P2

gen. 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. Von Ressource R_1 sind 3 Einheiten vorhanden, von R_2 sind 2 Einheiten vorhanden.

1.1 Abarbeitungs Diagramm

Abbildung 1 stellt ein Abarbeitungs Diagramm für die Prozesse P_1 und P_2 dar. 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. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.(3P. pro Flaeche. 1P Abzug pro falsches Kästchen)

2. 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". (4P. f. oben, 2P. f. unten, 1P Abzug pro falsches Kästchen, alles falsch - keine Punkte)

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. (1P)

4. Stellt der Punkt P einen Deadlock dar? Begründen Sie Ihre Antwort! (2P, 1P f. Ja/Nein, 1P f. Begruendung)

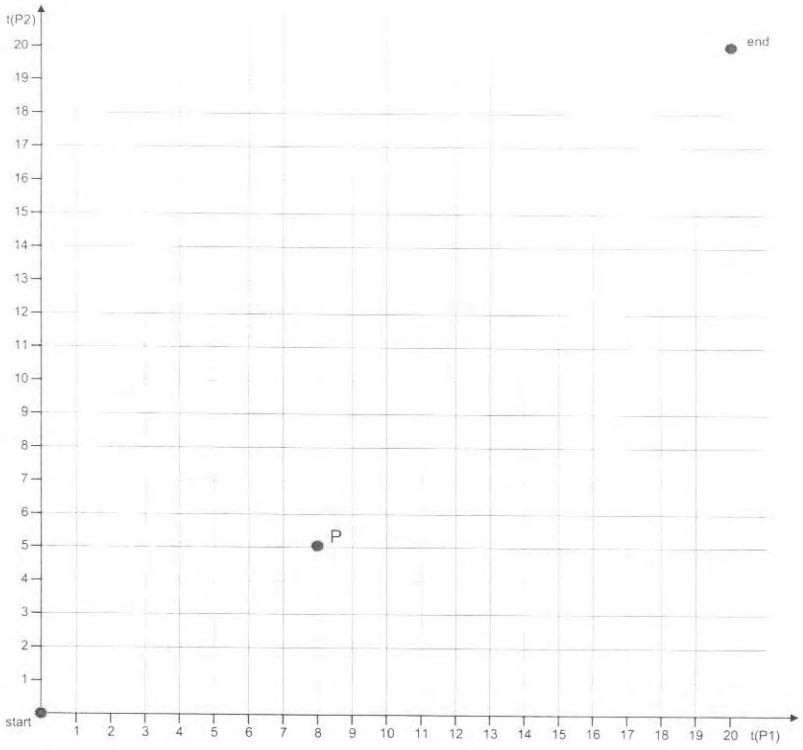


Abbildung 1: Abarbeitungs Diagramm

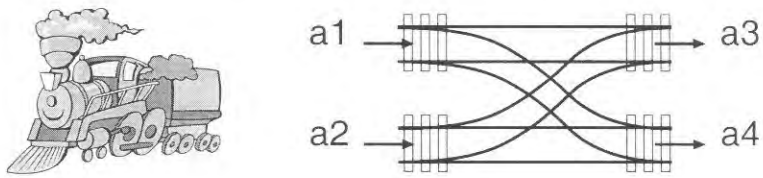
1.2 Deadlock Avoidance

Bestimmen Sie die Claim-Matrix C der Prozesse, sowie die Allokationsmatrix A im Punkt P. (Claim: 2 P, Alloc: 2P)

Ist P ein Safestate? Wenn ja, geben Sie mit dem Banker's Algorithm eine Ableitung an. Wenn nein, begründen Sie warum P kein Safestate ist. (3P: 1P Ja/Nein, 2P Begruendung)

2 Synchronisation (30)

Der Verschubbahnhof *Innovativa* hat eine relativ einfache Gleisstruktur, wie in folgender Abbildung dargestellt ist:



Der Verschubbahnhof hat nur ein einziges Weichensystem, mit dem es möglich ist, von dem einen Gleis auf das andere zu wechseln. Die Anschlusspunkte des Weichensystems sind mit **a1, a2, a3** und **a4** benannt. Wichtig ist, dass entsprechend den eingezeichneten Pfeilen ein Zug nur von **a1** oder **a2** kommend nach **a3** oder **a4** fahren darf.

Für den Verschubbahnhof ist eine Synchronisations-Software zu entwickeln, um Zugkollisionen während des Passierens des Weichensystems zu vermeiden. Das Weichensystem stellt **drei Betriebsarten** zur Verfügung:

KEINE: Keine Züge dürfen passieren.

GERADE: Züge dürfen nur geradeaus passieren.

ALLE: Züge dürfen sowohl geradeaus als auch diagonal passieren.

Es sind folgende Funktionen zu implementieren:

zug_kontrolle(von, nach) zur Kontrolle von Zügen. Diese Funktion steuert einen Zug, fahrend von **von** nach **nach**. Der Zug soll entsprechend der aktuellen Weichenbetriebsart und anderen Zügen synchronisiert werden. Sollte **von**$\notin${**a1,a2**} oder **nach**$\notin${**a3,a4**} gelten (d.h. eine nicht im System vorgesehende Fahrtrichtung), so ist die unten beschriebene Funktion **setze_stop(von)** aufzurufen, wobei **von** die Richtung ist, aus der dieser Zug kommt (der Zug selbst bleibt in diesem Fall einfach stehen).
Wenn das Weichensystem passierbar ist (d.h., kein Konflikt mit der aktuellen Weichenbetriebsart und anderen Zügen), ist die Funktion **stelle_weichen(von, nach)** aufzurufen, um die Weiche korrekt zu stellen. Anschließend ist zum eigentlichen Durchfahren der Weichenanlage die Funktion **passiere()** aufzurufen.

setze_stop(von) zum Sichern der Weichenanlage. Der Parameter **von** bezeichnet die aktuelle Position eines Zuges (hier **von**$\in${**a1, a2, a3, a4**}). Die Funktion soll für die Position **von** die Strecken zu den beiden gegenüberliegenden Weichenanschlusspunkten sperren. Ein Zug in gleicher Fahrtrichtung am Parallelgleis geradeaus fahrend, soll jedoch weiterhin passieren dürfen.

Beispiel: Angenommen, ein Zug kommt fälschlicherweise von (**von**=**a4**). Somit sind alle weiteren Züge kommend aus **a1** oder **a2** und nach **a4** fahrend, zu blockieren. Ein Zug, kommend von **a1** oder **a2** und nach **a3** fahrend, darf jedoch weiterhin passieren.

weichen_betriebsart_uebernahme() Diese Prozedur ermittelt in einer Endlosschleife durch **p = hole_perm()** die aktuelle Weichenbetriebsart **p** $\in$ {**KEINE, GERADE, ALLE**} und setzt entsprechend die im Programm benötigten Synchronisationskonstrukte.

Implementieren Sie alle Funktionen sowie Initialisierungen derart, sodass **defaultmäßig Weichenbetriebsart GERADE** aktiv ist (z.B.: wegen mechanischem Fehler in der Weichenumschaltung). Das heißt, Züge dürfen defaultmäßig nur geradeaus passieren.

Hinweis: Der Fall, dass hinter einem wartenden Zug ein neuer Zug nachkommt, braucht nicht berücksichtigt zu werden.

Die Verwendung von globalen Variablen bzw. Busy-Waiting zur Synchronisation ist verboten!

a) Initialisierungen (4)

Die Synchronisation ist mit (*möglichst wenig!*) Semaphoren durchzuführen wobei unnötige Einschränkungen der Parallelität zu vermeiden sind. Verwenden Sie zur Initialisierung der Semaphoren die Funktion **init(s,v)**, welche als ersten Parameter den Semaphor und als zweiten Parameter den entsprechenden Initialisierungswert erhält. Danach können die Funktionen **P(s)** und **V(s)** auf den Semaphor angewendet werden.

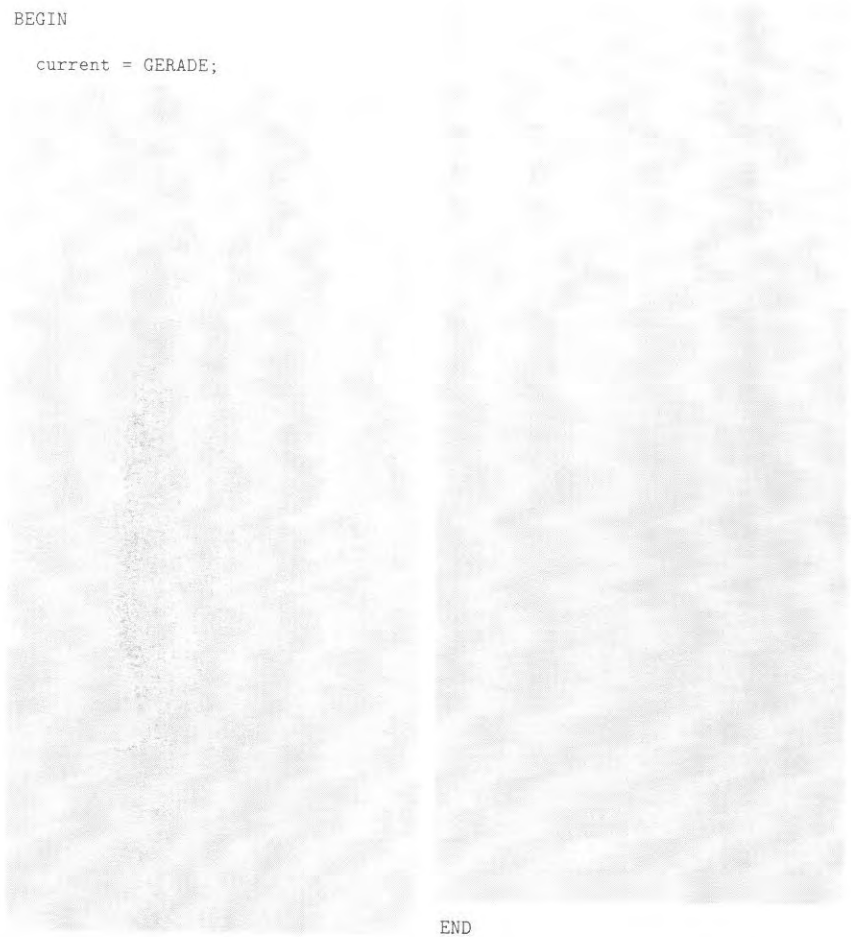
Geben Sie hier die notwendigen Initialisierungen von Semaphoren an.

b) Setzen einer neuen Weichenbetriebsart (8)

Programm zum Setzen der Weichenbetriebsart für Wartungsarbeiten.
`weichen_betriebsart_uebernahme()`

```
BEGIN

  current = GERADE;
```

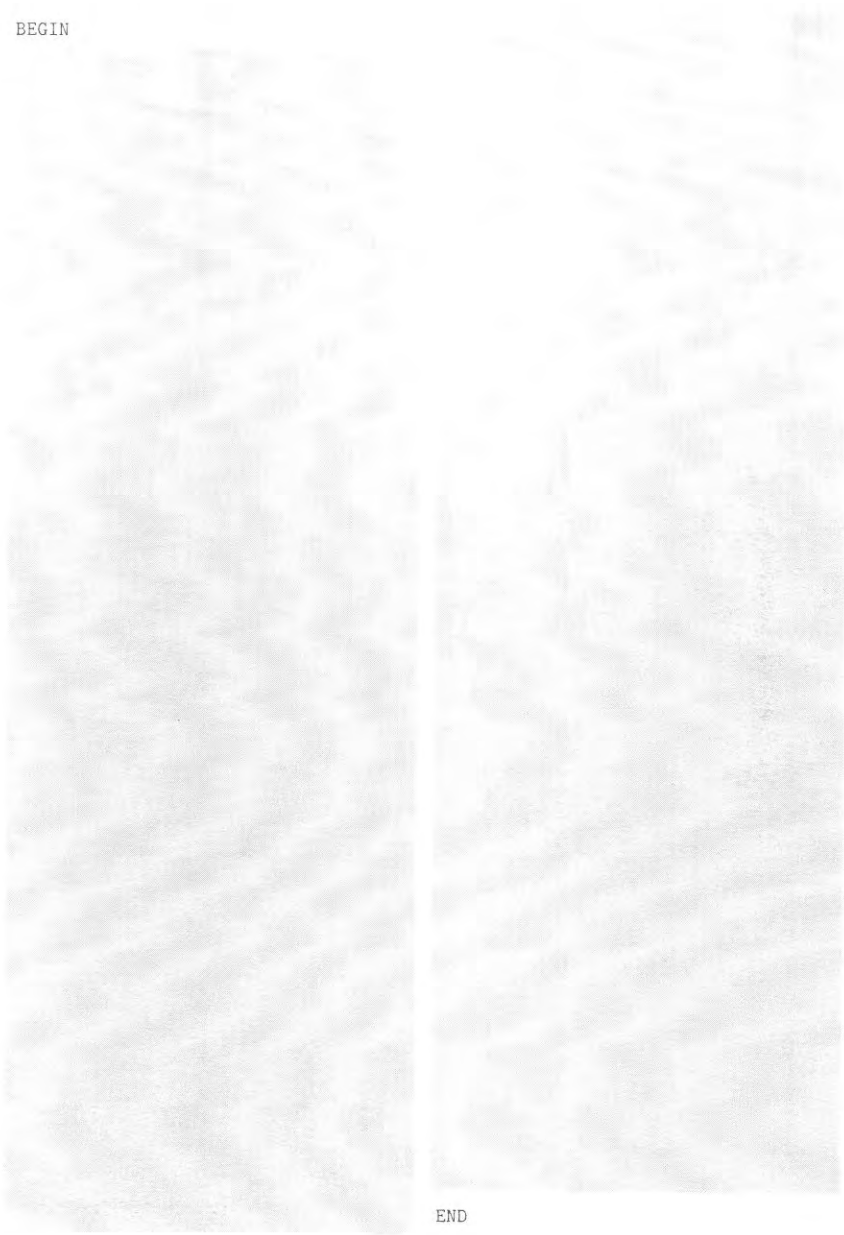


```
END
```

c) Zugkontrolle (14)

Programm zur Kontrolle eines Zuges:
`zug_kontrolle(von, nach)`


```
BEGIN
```



```
END
```

d) Stopsignal setzen (4)

- Programm zum Stoppen der für einen Zug stehend auf von kritischen entgegenkommenden Züge:
`setze_stop(von)`

```
BEGIN

END
```


3 Speicherverwaltung (23)

a) Kombination aus Segmentierung und Paging

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

Das im Folgenden betrachtete Speicherverwaltungssystem verwendet zur Adressierung 32-bit Adressen (virtuell und physikalisch). Für das Paging sind alle Seitenrahmen 256 Bytes groß. Das verwendete Adressformat ist folgendes:

Seg# (12 bit)	Page# (12 bit)	Offset (8 bit)

Hierbei wird assoziativer Zugriff (associative mapping) auf die Segmenttabelle und direkter Zugriff (direct mapping) auf die Seitentabelle verwendet.

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle und Seitentabelle (alle Werte sind als Hexadezimalzahlen angegeben):

Segmenttabelle		
Seg#	Base	Length
0x000	0x34500234	0x0FF
0x39A	0x2A2AA342	0x005
0x723	0x2EE23323	0x002
0xCD3	0xB0010010	0x001

Seitentabelle	
Address	Frame#
0x2A2AA342	0x456D43
0x2A2AA343	0x123499
0x2A2AA344	0x19D453
0x2A2AA345	0x4F5D1D
...	...
0x2EE23323	0x453AA1
0x2EE23324	0x434DAA
0x2EE23325	0x421111
...	...
0x345002E7	0x12432A
0x345002E8	0xABDDAD
0x345002E9	0xDF45F4
0x345002EA	0x45DAEE
...	...
0xB0010010	0x761010
0xB0010011	0x859631
0xB0010012	0x5A64D2
0xB0010013	0x5A42DF
0xB0010014	0x4AF5DA
...	...

Ermitteln Sie unter Benützung obiger Tabellen die physikalischen Adressen zu folgenden virtuellen Adressen (ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld) (13P, pro Fehler -2):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0x39A00123	
0xCD30016A	
0x39A002A2	
0x72300023	
0x39A00023	
0x72300100	
0xCD300101	
0x0000B502	
0x0000B4FF	

b) Verständnisfragen

Kreuzen Sie bitte die richtigen Antworten an. Bzw., geben Sie an, ob die Aussage richtig oder falsch ist oder beantworten Sie die Frage. (Pro Frage 1P(ausser 7.frage), 7. Frage: pro kreuz ein punkt - falsches Kreuz -1 Punkt

- Eine Austauschstrategie, die auf alle Seiten des Hauptspeichers angewandt wird, nennt man
  - ◯ lokale Ersetzungsstrategie
  - ◯ globale Ersetzungsstrategie
- Beim *Fixed Partitioning* sind die Partionen *immer* von gleicher Größe.
  - ◯ richtig
  - ◯ falsch
- Beim *Fixed Partitioning* können sich Partitionen im Hauptspeicher überlappen.
  - ◯ richtig
  - ◯ falsch
- Bei einem Buddy System sind die Blockgrößen das Produkt einer Zweierpotenz mit der kleinsten Blockgröße.
  - ◯ richtig
  - ◯ falsch
- Beim *Fixed Partitioning* ist die Hauptspeichernutzung extrem effizient.
  - ◯ richtig
  - ◯ falsch
- Welche Replacement-Policy ist am einfachsten zu implementieren?
  - ◯ Least recently used
  - ◯ First in - first out
  - ◯ Optimale Strategie
- Zu welchen Effekten (mehrere möglich) kann es bei Segmentierung kommen?
  - ◯ Unterschiedliche Segmentlängen
  - ◯ Internal Fragmentation
  - ◯ External Fragmentation
- Was ist ein *Working Set* im Speichermanagement?
- Wenn alle Seiten eines Working Sets im Hauptspeicher sind, kann ein Prozess effizient abgearbeitet werden.
  - ◯ richtig
  - ◯ falsch

4 Scheduling (22)

4.1 Single Processor 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.

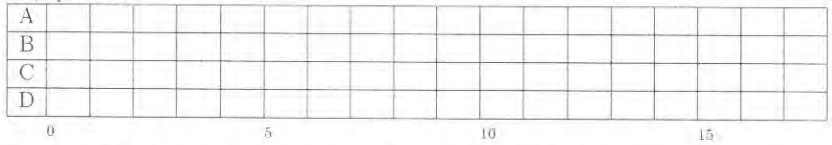
Task	Ausführungszeit	Periodendauer
A	2	8
B	1	4
C	2	7
D	1	5

Ermitteln Sie für dieses Taskset die *notwendige* und die *hinreichende* Bedingung für das *Rate Monotonic Scheduling* (RMS) Verfahren. Berechnen Sie die Zahlenwerte auf mindestens eine Kommastelle genau.

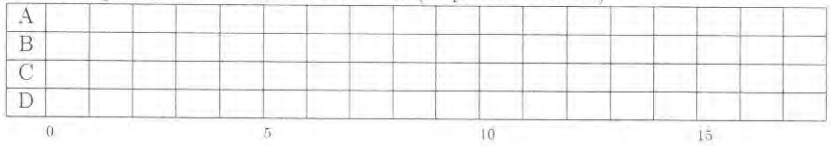
Ist die notwendige Bedingung erfüllt? 1P ◯ Ja ◯ Nein
Ist die hinreichende Bedingung erfüllt? 1P ◯ Ja ◯ Nein

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. Eine Vorlage dient als Ersatz, streichen Sie gegebenenfalls eine falsch ausgefüllte Vorlage deutlich durch.

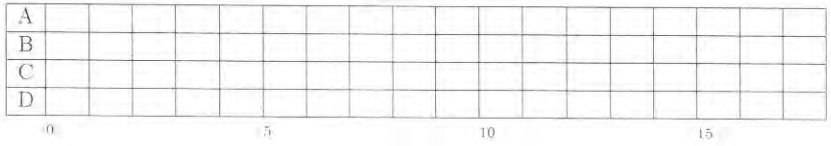
Scheduling nach dem **RMS**-Verfahren: ersten 3 spalten: 1 Punkt, spalte 4,5:1P; spalte 6,7: 1P; spalte 8,9:1P



Scheduling nach dem **EDF**-Verfahren: 5x (3 spalten: 1Punkt)



Ersatzvorlage: Scheduling nach dem -Verfahren:



4.2 Verständnisfragen (9)

Pro falsche oder fehlende Antwort -1P. Welche Scheduling-Strategie ähnelt einer Round Robin-Strategie mit sehr langen Zeitscheiben?

Beurteilen Sie die folgenden Aussagen! Falsche Antworten werden negativ gewertet!

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

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

◯ Ja ◯ Nein Earliest Deadline First Scheduling findet in Single-Prozessor Systemen immer eine Lösung.

◯ 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.

◯ 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 nicht 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 Die Prioritäten beim RMS Scheduling ergeben sich durch die *Processor Utilization* der Tasks.

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

Vorlesungsprüfung aus Systemprogrammierung 2002-05-29

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (35)

In der Produktionsanlage eines Zulieferbetriebs werden Komponenten aus Aluminiumzylindern und Kunststoffringen von Industrieroboter hergestellt. Die Aluminiumzylinder und Kunststoffringe stehen in einem Materiallager bereit. Fertige Teile werden in einem Endlager verstaut. Die Abfolge der Arbeitsschritte bei der Produktion ist in Abbildung 1 veranschaulicht.

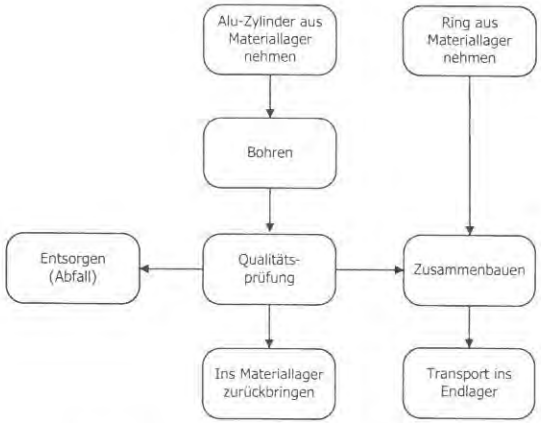


Abbildung 1: Abfolge der Arbeitsschritte

Für die Herstellung wird ein Kunststoffring an einem Aluminiumzylinder angebracht. Hierzu muss ein Aluminiumzylinder aus dem Materiallager geholt und gebohrt werden. Danach wird eine Materialprüfung durchgeführt, die den tatsächlich erzielten Bohrdurchmesser bestimmt. Ist der erzielte Bohrdurchmesser kleiner als 30 mm, so wird der Aluminiumzylinder

ins Materiallager retourniert, um später erneut gebohrt zu werden. Bei einem ermittelten Bohrdurchmesser über 31 mm muss der Aluminiumzylinder entsorgt werden. Nach erfolgreicher Materialprüfung, dh. der Bohrdurchmesser beträgt 30 mm oder 31 mm, kann ein Kunststoffring am Aluminiumzylinder angebracht und das fertige Teil ins Endlager abtransportiert werden.

Zur Durchführung dieser Verarbeitungsschritte stehen Industrieroboter bereit, die jeweils eine bestimmte Teilaufgabe ausführen können. Diese Roboter müssen außerdem Teile an andere Roboter weitergeben bzw. von diesen Teile übernehmen.

- Roboter 1 und Roboter 2 können Teile (Zylinder oder Ring) aus dem Materiallager holen.
- Roboter 3 führt Bohrungen am Zylinder durch und kann Teile ins Materiallager, sowie in die Entsorgungsstätte bringen.
- Roboter 4 ist mit der Qualitätsprüfung betraut und hat fehlerhafte Teile zu entsorgen.
- Roboter 5 kann zwei Teile zusammenbauen und fertige Teile ins Endlager transportieren.

Erstellen Sie die Programme der Industrieroboter und sorgen Sie für eine korrekte Abfolge der Arbeitsschritte durch Synchronisation mit Semaphoren!

Beachten Sie dabei folgende Hinweise:

- Die Verwendung von globalen Variablen bzw. Busy-Waiting zur Synchronisation ist verboten!
- Verhindern Sie das Auftreten von Deadlocks!
- Die Roboter 1, 2, 3 und 4 können zu einem bestimmten Zeitpunkt jeweils nur ein einziges Teil bearbeiten.
- Roboter 5 kann maximal zwei zum Zusammenbau bestimmte Teile aufnehmen.
- Achten Sie auf ein hohes Maß an Parallelismus bei der Verarbeitung der Teile durch die Roboter!
- Verwenden Sie möglichst wenige Semaphore!

a) Erstellen und initialisieren Sie alle benötigten Semaphore. (5)

Zum Anlegen eines Semaphores steht Ihnen das Statement `SemInit` zur Verfügung. Mit diesem kann man sowohl einfache Semaphore (`SemInit(SemaphorName,InitWert)`) als auch Semaphorarrays (`SemInit(SemaphorArrayName[Anzahl],InitWert1,InitWert2,...)`) erzeugen und initialisieren. Das Sperren eines Semaphors erfolgt durch den Aufruf von `P(SemaphorName)` bzw. `P(SemaphorArrayName[Index])`, mit `V(SemaphorName)` bzw. mit `V(SemaphorArrayName[Index])` gibt man es frei!

b) Vervollständigen Sie die Programme der Industrieroboter. (30)

Verwenden Sie hierzu die folgenden Routinen:

- *Nimm(roboter)* nimmt das Teil vom Industrieroboter *roboter* $\in$ *{roboter1,....,roboter5}* entgegen.
- *HoleAusLager(obj)* besorgt das Teil *obj* aus dem Materiallager. Diese Aktivität ist Roboter 1 und Roboter 2 vorbehalten.
  (obj $\in$ *{Aluzylinder,Kunststoffring}*).
- *Gib(ort)* reicht das Teil zum Standort *ort* weiter, wobei *ort* ein Industrieroboter *(roboter1,....,roboter5)*, das Endlager *(endlager)*, das Materiallager *(materiallager)*, oder die Entsorgungsstätte *(abfall)* ist. Wird das Teil an einen anderen Roboter übergeben, dann darf der Roboter nach dem Aufruf von *Gib(ort)* solange kein weiteres Teil aufnehmen, bis der andere Roboter das weitergereichte Teil mittels eines Aufrufs von *Nimm(roboter)* übernommen hat.
- *Bohren()* veranlasst einen Roboter zur Durchführung einer Bohrung am Teil. Diese Routine kann nur von Roboter 3 ausgeführt werden.
- Die Routine *int BestimmeDurchmesser()* ermittelt den erzielten Bohrdurchmesser in mm. Diese Routine kann nur von Roboter 4 ausgeführt werden.
- Zwei Teile werden mit der Routine *Zusammenbauen()* zu einem Teil verarbeitet. Diese Routine kann nur von Roboter 5 ausgeführt werden.

Roboter 1

Roboter 2

Roboter 3

Roboter 4

Roboter 5

2 Scheduling (20)

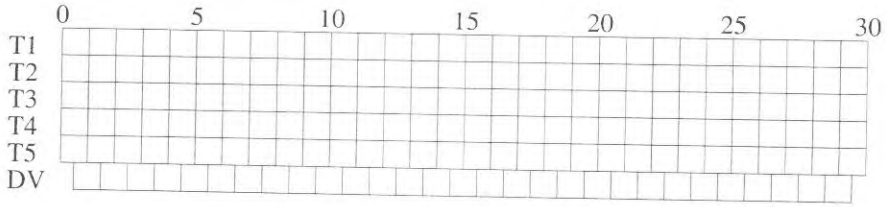
a) Rate-Monotonic Scheduling (10)

Task	Laufzeit (ms)	Periode (ms)
T1	5	12
T2	2	23
T3	2	22
T4	2	11
T5	3	13

Folgendes wird angenommen:

- Alle Tasks sind periodisch.
- Deadline = Periode.
- Laufzeit ist bekannt.
- Der Scheduling Overhead wird vernachlässigt.
- Alle Tasks sind unabhängig.

Dieses Taskset soll mit einem Rate-Montonic Scheduling Algorithmus für den Worst Case[1] gescheduled werden. Bitte vervollständigen Sie die folgende Skizze zur Gänze bzw. bis zur ersten Deadlineverletzung. Kennzeichnen Sie eine solche, indem Sie in der Zeile DV den Task eintragen, für den die Deadline nicht eingehalten wurde.



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:

◯ 11 ms ◯ 12 ms ◯ 13 ms ◯ 22 ms ◯ 23 ms
◯ 24 ms ◯ 26 ms ◯ 44 ms ◯ 46 ms
◯ einen Zeitpunkt der in der Liste nicht aufscheint

[1] Im Worst Case beginnen alle Perioden bei 0.

b) Fragen zu Rate-Monotonic Scheduling (6)

Gegeben seien $n = 4$ Tasks; mit den Ausführungszeiten C_1, C_2, C_3, C_4 und den Perioden T_1, T_2, T_3, T_4; der Scheduling-Overhead wird vernachlässigt:

Welche Bedingung muss erfüllt sein, dass Sie sicher **kein** mögliches Scheduling finden können?

Welche Bedingung müssen die 4 Tasks erfüllen, dass es sicherlich ein mögliches Scheduling gibt?

Können Sie mit Hilfe dieser beiden Bedingungen alle Fälle klassifizieren, oder gibt es Fälle welche sich nicht eindeutig entscheiden lassen?

◯ alle Fälle sind eindeutig entscheidbar
◯ es gibt Fälle, die nicht eindeutig entscheidbar sind

In einem System das RMS verwendet, messen Sie eine Prozessorauslastung von 0,65. Die Taskanzahl ist Ihnen unbekannt; weiters wird der Scheduling-Overhead vernachlässigt. Können Sie sicher sein, dass alle Deadlines eingehalten werden?

◯ ja ◯ nein

Wenn Sie die vorherige Frage mit nein beantwortet haben, geben Sie bitte an, welche Informationen Sie zusätzlich benötigen, um eine sichere Aussage machen zu können. Im Fall, dass Sie mit ja geantwortet haben, erklären Sie bitte, warum diese Information ausreicht?

c) Fragen zu Scheduling allgemein (4)

Kreuzen Sie bitte an ob die Aussagen korrekt sind

Bei First-Come-First-Served (FCFS) Scheduling kann es niemals zu Starvation kommen:

◯ korrekt ◯ inkorrekt

Round Robin (RR) Scheduling erfordert nur geringen Scheduling-Overhead:

◯ korrekt ◯ inkorrekt

Shortest-Remaining-Time (SRT) Scheduling benachteiligt Prozesse mit langen Ausführungszeiten:

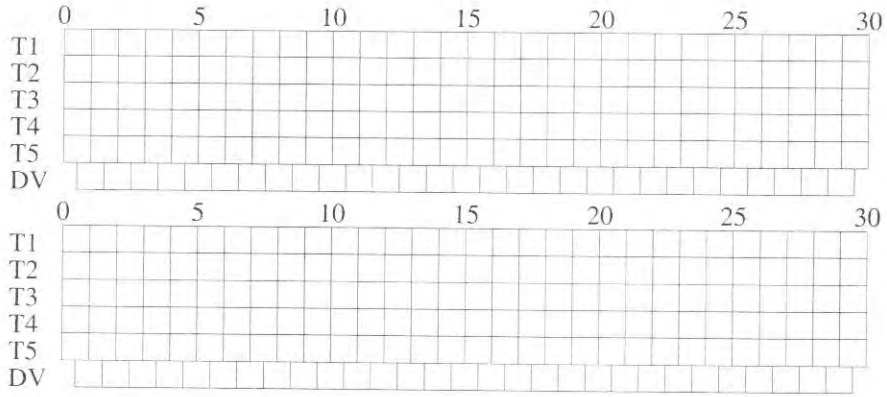
◯ korrekt ◯ inkorrekt

Bei Highest-Response-Ratio-Next (HRRN) Scheduling ist Starvation unmöglich:

◯ korrekt ◯ inkorrekt

Ersatzvorlagen zu a)

Streichen Sie ungültige Vorlagen deutlich durch, wenn Sie eine dieser Vorlagen verwenden!



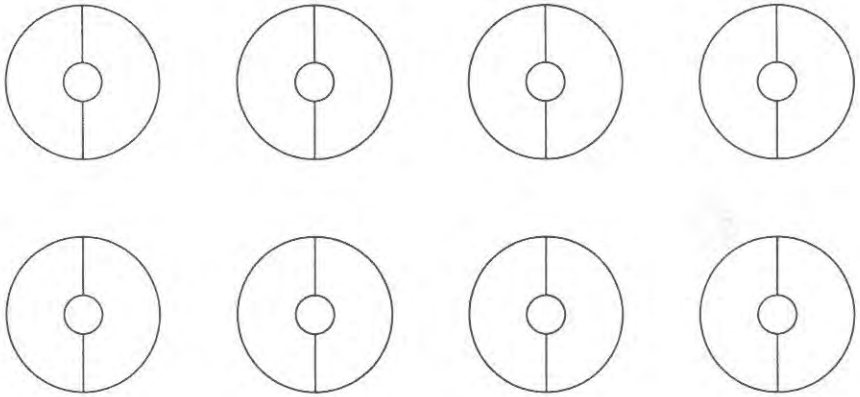
3 Speicherverwaltung - Replacement Policies(25)

a)(9)

Ein Prozess referenziert 5 pages A,B,C,D und E. Wie groß ist die Anzahl der Page Faults, die während dieser Zugriffe auftreten? Nehmen Sie jeweils an, dass der Speicher vor dem Zugriff leer ist.

Access Order	Replacement Policies	Allocation Size in Number of Pages	Number of Page Faults
A;B;C;D;A;B;E;A	First In First Out	3	
A;B;A;B;C;A;D;E	Simple Clock	2	
E;B;A;B;C;A;C;B	Least Recently Used	3	

Für etwaige Skizzen zur Simple Clock Policy verwenden Sie bitte folgende Vorlagen:



b)(16)

Ein Prozess hat 4 Page Frames alloziert. Die Allocation Size in Number of Pages ist 4. Folgende Abkürzungen werden verwendet:

TL..time loaded: Der Zeitpunkt als die Page das letzte Mal geladen wurde

TR..time reference: Der Zeitpunkt als auf die Page das letze Mal zugegriffen wurde

Die angegebenen Zeiten sind die Ticks vom Zeitpunkt des Prozessstarts (0) weggezählt.

Virtual Page Number	Page Frame	TL	TR	Clock Use Bit
2	0	60	161	1
1	1	130	160	1
0	2	26	162	1
3	3	20	163	1

Ein Page Fault der virtuellen Page 4 ist aufgetreten. Welcher Page Frame wird ausgetauscht werden, wenn die folgende Memory Managment Policy verwendet wird?

Policy	Page Frame	Warum wird der Page Frame ausgetauscht werden?
First In First Out		
Least Recently Used		
Simple Clock*		
Optimal**		

*Der Frame Allocation Pointer wird den Page Frame Nummern entsprechend weitergesetzt, die Sequenz ist also: 0,1,2,3,0,....

**Zukünftige Access Sequence nach der aktuellen Virtual Page 4: 3,0,1

Vorlesungsprüfung aus ○ Betriebssysteme ○ Systemprogrammierung 2001-01-17

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(25)	2.)(30)	3.)(25)	4.)(20)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. 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.

Das Diagramm in Abbildung 1 zeigt die Ressource-Anforderungen der beiden Prozesse. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

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. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.
2. 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".
3. 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.

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.

2 Synchronisation (30)

Zum 5. Dezember wollen Sysprog Studenten ein "Krampuskränzchen" feiern. Dazu beabsichtigen sie, Maroni zu braten sowie Glühwein zu kochen. Folgende Randbedingungen müssen berücksichtigt werden.

- Es gibt 5 Kochtöpfe (à 70 Portionen) ...
- sowie 3 Backbleche (à 120 Portionen).
- Der Ofen hat 3 Herdplatten und 1 Backrohr.
- Im Backrohr hat genau ein Backblech Platz.
- Wenn das Backrohr in Betrieb ist, darf nur 1 Herdplatte verwendet werden. Wird das Backrohr nicht verwendet, können alle 3 Herdplatten verwendet werden.
- Zu Beginn des "Krampuskränzchens" sind alle Backbleche und Töpfe leer.
- Erst wenn ein Topf bzw. Blech vollständig geleert ist, wird vom Nächsten entnommen.
- Die Gäste geben ein Backblech bzw. einen Topf nur vollständig entleert wieder heraus.
- Die Lösung soll ein hohes Maß an Parallelität besitzen und dabei keine globalen Variablen verwenden.

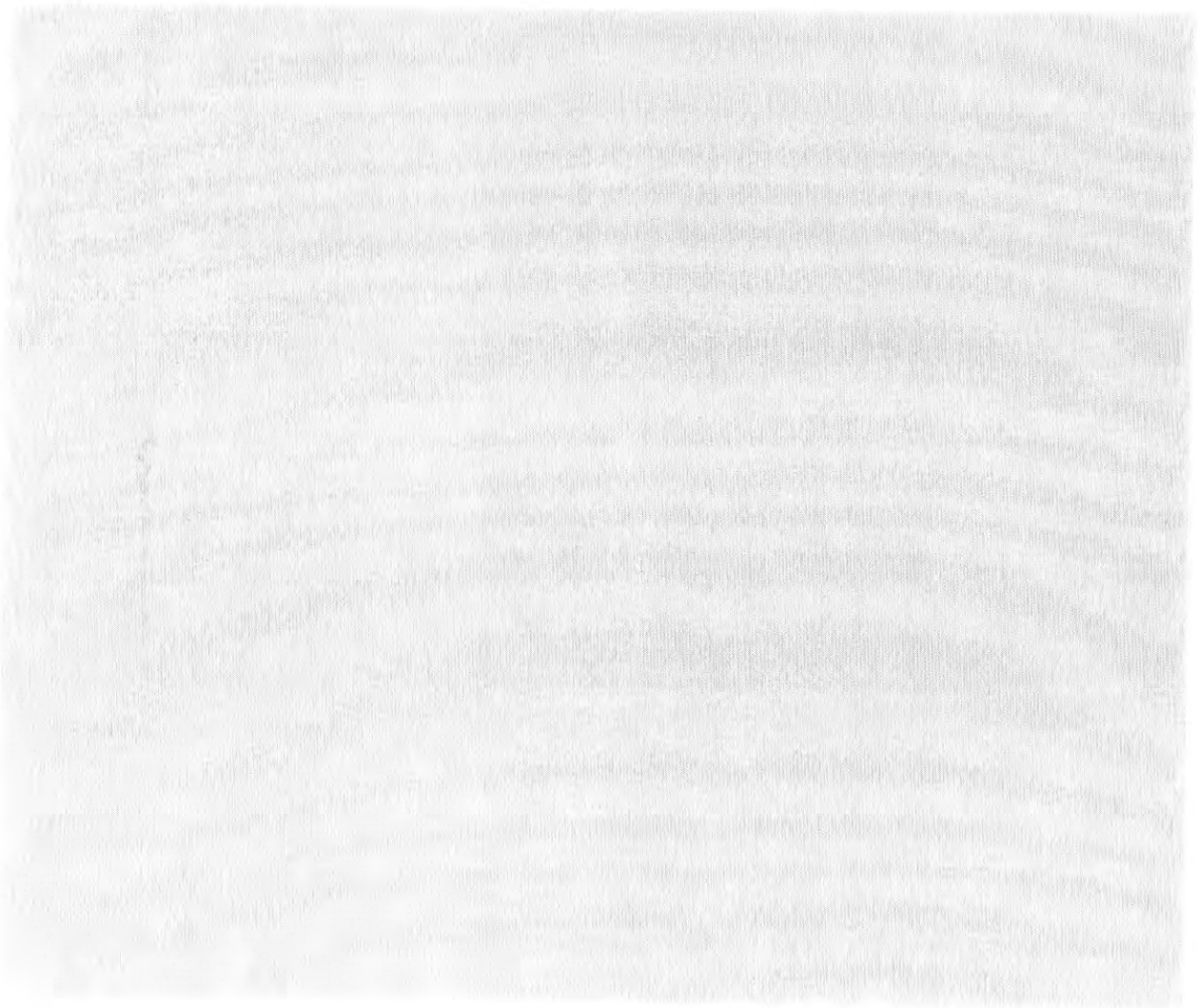
Es stehen Ihnen die folgenden Routinen für die Implementierung der drei Programme `hungriger_Student`, `Maroni_Brater` und `Gluehwein_Koch` zur Verfügung:

- `SInit(`SQ, val`)` Initialisiert den Sequencer SQ mit dem Wert val.
- `STicket(`SQ, val`)` Erhöht den Sequencer SQ um den Wert val und retourniert den **neuen** Wert.
- `EInit(`EC, val`)` Initialisiert den Eventcounter EC mit dem Wert val.
- `EWait(`EC, val`)` Ein Aufruf der Routine `EWait` kehrt zurück, sobald der Wert des Eventcounters $EC \geq val$ ist.
- `EAdvance(`EC, val`)` Erhöht den Wert des Eventcounters EC um val.
- `HoleGluehwein()` Ein durstiger Student kann durch Aufruf dieser Prozedur an Glühwein herankommen. Die Prozedur blockiert solange bis Glühwein vorhanden ist.
- `HoleMaroni()` Mit dieser Prozedur versorgt sich ein hungriger Student mit Maroni. Auch diese Prozedur blockiert bis Maroni verfügbar sind.
- `BlechHolen()` Ein Maronibrater nimmt durch diese Prozedur ein leeres Blech und stellt es bereit für den nächsten Arbeitsschritt.
- `MaroniAufsBlech()` Richtet **120 Portionen** Maroni auf einem bereitgestellten Backblech an.

2.3 Maronibrater (11)

Implementieren Sie das Programm `Maroni_Brater` das auf allen Maronibratern parallel exekutiert wird.

```
while(fest == IM_GANGE){
```



```
  GebtDenHungerden()
}
Aufraeumen()
```

3 Speicherverwaltung (25)

a) Translation Lookaside Buffer (12)

Das gegebene Speicherverwaltungssystem basiert auf der Paging-Technik und verwendet einen *Translation Lookaside Buffer (TLB)* mit drei Einträgen. Der TLB wird mit der **FIFO** Strategie verwaltet, d.h., wenn für eine neu einzutragende Seite kein Eintrag mehr frei ist, wird jener Eintrag ersetzt, der sich am längsten im TLB befindet. Zu diesem Zweck enthält der TLB für jeden Eintrag ein Feld (mit der Bezeichnung FIFO), das angibt, wie lange sich der entsprechende Eintrag in dem TLB befindet. Bitte beachten Sie, dass der niedrigste Wert dem am längsten im TLB befindlichen Eintrag zugeordnet ist. Der Zähler eines *neuen* Eintrags ist das um eins inkrementierte Maximum der bisherigen FIFO-Counters.

Der TLB und die benötigte Page Table werden mittels **Associative Mapping** angesprochen.

Eine virtuelle Speicheradresse hat folgendes Format:

Page#(8 Bit)	Offset (16 Bit)

Eine physikalische Speicheradresse hat folgendes Format:

Frame#(12 Bit)	Offset (16 Bit)

Die Speicheradressen, Frame- und Pagenummern sind im Hexadezimalsystem angegeben.

Auf der nächsten Seite ist ein Translation Lookaside Buffer und eine Page Table vorgegeben. Simulieren Sie ausgehend von diesen Daten den hintereinanderfolgenden Zugriff auf die virtuellen Speicheradressen auf den folgenden Seiten. Bitte beachten Sie dabei:

- Befindet sich eine Seite nicht im Hauptspeicher, so können Sie die Nummer des Page-Frames für die Page aus den nicht belegten frei wählen. Tragen Sie diese in der Page Table auf der nächsten Seite ein.
- Bestimmen Sie, ob es zu einem TLB Hit oder Miss kommt und ob es zu einem Main Memory Page Hit kommt oder nicht (Kreuzen Sie entsprechend **Ja** oder **Nein** an). (Hinweis: ein TLB Hit impliziert einen Page Table Hit)
- Geben Sie weiters jeweils den Inhalt des TLB **nach** dem Zugriff auf die Page an.

Virtuelle Adresse

0x223ABC

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x3921C3

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x031274

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x121100

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

FIFO

A	B	A	D	E	B	A	C	A	D	A	C

page fault?

Anzahl d. page faults:

c) Verständnisfragen (4)

- Welche dieser Replacement-Policies ist am einfachsten zu implementieren?
  - ◯ Least recently used
  - ◯ Clock algorithmus
  - ◯ First in - first out
- Um die interne Fragmentierung zu reduzieren, muss man die *Page Size*
  - ◯ verringern
  - ◯ vergrößern
- Wieviele Zugriffe auf die Pagetable benötigt ein System mit Translation Lookaside Buffer im Falle eines TLB Hits, um die virtuelle Adresse aufzulösen?

  Zugriff(e)
- Im Vergleich zu den anderen Replacement-Policies produziert die *optimal policy* bei beschränkter Buffergröße
  - ◯ keine page faults
  - ◯ die wenigsten page faults
  - ◯ die meisten page faults

d) (8)

Geben Sie die Schritte an, die zum Senden und beim Empfangen einer mit einem Public Key Verfahren verschlüsselten und signierten Nachricht notwendig sind. Welche Schlüssel braucht man dabei?