Klausur zur Vorlesung
„Konzepte und Methoden der Systemsoftware“
Uni Paderborn SS 00
Prof. Dr. H.-U. Heiß
12. Juli 2000

Name, Vorname	
Matrikelnummer	
Fachbereich	
Studiengang	
Prüfungsordnung	
Prüfung oder Leistungsnachweis?	

- *Schreiben Sie zunächst sofort Ihren Namen und Ihre Matrikelnummer auf jedes Blatt der Klausur!*
- *Blätter ohne Namen werden nicht gewertet!*
- *Lassen Sie die Klausur zusammengeheftet!*
- *Die Klausur dauert 120 Minuten und umfasst 5 Aufgaben auf 18 Seiten.*
- *Die Klausur ist bestanden, wenn mindestens 50% der Punkte erreicht wurden.*
- *Es sind keine Hilfsmittel zugelassen (insbesondere keine Taschenrechner und Handys)!*
- *Abschreiben und abschreiben lassen führt zum Nichtbestehen der Klausur!*
- *Benutzen Sie kein eigenes Konzeptpapier bzw. Schmierpapier. Sie bekommen bei Bedarf Papier von der Klausuraufsicht!*
- *Wenn Sie Konzeptpapier abgeben wollen, dann nur mit Namen und Matrikelnummer!*
- *Kennzeichnen Sie Ihre Lösung eindeutig. Es wird keine Lösung gewertet, wenn Sie zu einer Aufgabe mehr als eine Lösung abgeben.*

Aufgabe	1	2	3	4	5	Σ
Punkte	20	25	18	25	32	120
Erreicht						

Note	Bestanden

Aufgabe 1: Koordination nebenläufiger Prozesse (3+2+15=20 Punkte)

a) Wenn Semaphore für die Synchronisation des exklusiven Zugriffs auf Systemresourcen (gegenseitiger Ausschluss) verwendet werden, so werden Sie im Normalfall mit der Anzahl der verfügbaren Ressourcen dieses Betriebsmitteltyps initialisiert. Wann ist es sinnvoll, einen Semaphor mit Null zu initialisieren? Beschreiben Sie das Szenario, wenn in diesem Fall als erstes eine p()-Operation aufgerufen wird!

Synchronisation von Prozessen (Signalisierung)

- **bei der *Synchronisation von Prozessen* bedeutet die p()-Operation die Waiting-for-signal-Operation**
- **initial existiert kein Signal, deshalb wird der Semaphor mit 0 initialisiert**
- **der erste Prozess, der die p()-Operation aufruft, ohne dass zuvor bereits die v()-Operation (bedeutet Signal-Operation) aufgerufen wurde, muss warten, bis das Signal ankommt**

Zur Erläuterung des Unterschieds (nicht für Lösung erforderlich):

- beim gegenseitigen Ausschluss bedeutet die p()-Operation das Anzeigen eines Wunsches zum Betreten des kritischen Abschnitts
- der kritische Abschnitt wird normalerweise als frei initialisiert (analog für Betriebsmittel), deshalb erhält der Semaphor einen Wert N ≥ 1; der erste Prozess, der die p()-Operation ausführt, darf den kritischen Abschnitt betreten bzw. die ersten N Prozesse, die die p()-Operation ausführen, dürfen eine Ressource des zugehörigen Betriebsmitteltyps belegen

(diese Erkenntnis sollte der Aufgabe 5 (Blatt2): Barber-Shop entnommen worden sein; vgl. Musterlösung)

b) Was ist der semantische Unterschied von signal_c/wait_c gegenüber signal/wait?

Bei der Signalisierung mit **signal/wait** wird eine einfache *Reihenfolgebeziehung zwischen Anweisungsfolgen* verschiedener Prozesse hergestellt. Es wird eine gemeinsame binäre Variable benutzt.
(Ein Abschnitt B in einem Prozess P2 soll nach einem Abschnitt A in einem Prozess P1 ausgeführt werden: Vor B in P2 wird die wait-Operation aufgerufen, nach A in P1 wird die signal-Operation aufgerufen.)
Realisierung sowohl als aktives Warten als auch als Signalisieren mit Wartezustand möglich.

signal_c/wait_c wird für die Bedingungssynchronisation (in Monitoren) eingesetzt. Während ein Prozess auf das Wahrwerden einer Bedingungsvariablen wartet, muss der Prozessor für andere Prozesse freigegeben werden. Ein Prozess kann somit seine eigene Ausführung blokkieren, bis (gemeinsam genutzte) Daten einen bestimmten Zustand erreicht haben. Verwendung für grobe Granularität (ggs. gegenseitiger Ausschluss).
wait_c(cond): Prozess gibt Monitor frei und wartet auf das nachfolgende signal_c(cond); danach setzt er im Monitor fort. Der Prozess wird in jedem Fall blockiert.
signal_c(cond): Ein wartender Prozess wird freigegeben; der Monitor ist wieder belegt. Gibt es keinen wartenden Prozess, so hat die Prozedur keinen Effekt.

c) In (Hoare, 1974)wurde vorgeschlagen, dass das Scheduling von Prozessen, die auf das Wahrwerden einer Bedingungsvariablen in einem Monitor warten, auf Basis von Prioritäten durchgeführt werden könnte. Verwenden Sie eine Syntax der Form WAIT_C(condition-name, priority), um die Reihenfolge der wartenden Prozesse in einer Bedingungswarteschlange anzuzeigen. Die Priorität bestimmt die Reihenfolge der Zeitpunkte, in der Prozesse reaktiviert werden. Die Priorität kann benutzt werden, um Prozesse in einer geeigneten Reihenfolge zu reaktivieren, die auf die Zylinder einer Festplatte schreiben wollen.

Nehmen Sie an, dass ein Sweep-Algorithmus die Plattenköpfe steuert: Die Köpfe werden in einer Richtung über die Oberfläche zum äußersten benötigten Zylinder bewegt und anschließend zurück in die entgegengesetzte Richtung. Die Köpfe stoppen auf ihrem Weg bei Zylindern, um Prozessen den Datentransfer zu ermöglichen.

Schreiben Sie Prozeduren für einen Monitor:

- eine Prozedur, die anfordert, die Plattenköpfe zu einem bestimmten Zylinder zu bewegen und die den aufrufenden Prozess blockiert, bis der angeforderte Zylinder erreicht wird, und
- eine Prozedur, die von einem Prozess aufgerufen wird, wenn er den Datentransfer mit einem angegebenen Zylinder abgeschlossen hat.

Verwenden Sie hierzu den auf der nächsten Seite angegebenen Programmrahmen.

Hinweise:

- Die Aufgabe der Monitorprozeduren besteht in der Synchronisation der Datentransfers für die anfordernden Prozesse. Anweisungen zum *Bewegen* der Köpfe und für den eigentlichen *Datentransfer* brauchen Sie nicht anzugeben.
- Die Variable *headpos* speichert die aktuelle Position der Köpfe.
- Die boolesche Variable *busy* speichert, ob ein Prozess aktuell einen Datentransfer ausführt.
- Es gelte: type cylinder = 0 ..cylmax;
- Verwenden Sie eine Operation QUEUE (condition-name), die genau dann *true* als Ergebnis liefert, wenn irgendein Prozess in der Warteschlange wartet, andernfalls liefert sie *false*.

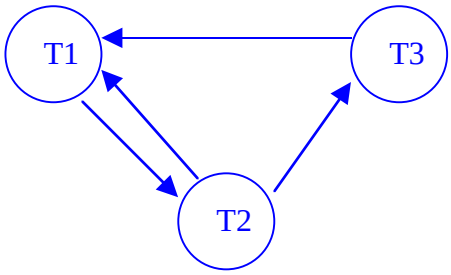
```
monitor module diskhead;
entry REQUEST, RELEASE;
import WAIT_C, SIGNAL_C, QUEUE;
var   headpos: cylinder := 0;
      direction: (up, down)   := up;
      busy: boolean      := false;
      upsweep, downsweep : condition;

procedure REQUEST (dest : cylinder);
          begin
              if busy then
              begin
                  if ((headpos<dest or headpos=dest) and direction = up)
                         then WAIT_C (upsweep, dest)
                         else WAIT_C (downsweep, cylmax-dest);
                  end;
              busy := true; headpos := dest;
          end;
procedure RELEASE;
          begin
              busy := false;
              if (direction = up) then
              begin
                   if QUEUE(upsweep)
                   then SIGNAL (upsweep)
                   else
                         begin
                               direction := down; SIGNAL (downsweep)
                         end;
                  else if QUEUE (downsweep)
                      then SIGNAL (downsweep)
                         else
                            begin
                               direction := up; SIGNAL (upsweep)
                            end;
              end;
          end;
end diskhead;
```

Aufgabe 2: Transaktionen (5+15+3+2=25 Punkte)

Gegeben sei die folgende Ausführungsfolge von drei nebenläufigen Transaktionen

Op	T1	T2	T3
1.		read(z)	
2.		read(y)	
3.		write(y)	
4.			read(y)
5.			read(z)
6.	read(x)		
7.	write(x)		
8.			write(y)
9.			write(z)
10.	read(y)		
11.	write(y)		
12.		write(x)	
13.		commit	
14.			commit
15.	commit		

a) Zeichnen Sie den Serialisierbarkeitsgraphen!



b) Geben Sie an, welche Eigenschaft der durch diese Ausführungsreihenfolge definierte Plan besitzt und begründen Sie kurz Ihre Antworten.

i) serialisierbar ❑ ja ☒ nein

Begründung:

Serialisierungsgraph nicht zyklenfrei: (T1,T2), (T1,T2,T3)

ii) rücksetzbar ☒ ja ❑ nein

Begründung:

T1 liest von und bestätigt nach T2

T3 liest von und bestätigt nach T2

T1 liest von und bestätigt nach T3

iii) kaskadierenden Abbruch vermeidend ❑ ja ☒nein

Begründung:

Es werden unbestätigte Daten gelesen (Op 4 und 10)

iv) strikt ❑ ja ☒ nein

Begründung:

siehe iii)

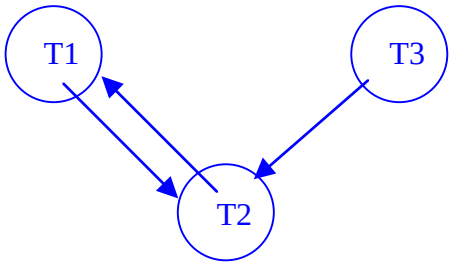
c) Nehmen Sie an, die Operationen 1-15 werden in der angegebenen Reihenfolge nicht direkt ausgeführt, sondern einem 2PL-Scheduler (Zweiphasen-Sperrung) als Eingabe übergeben. Dadurch kann eine andere Ausführungsreihenfolge entstehen.

Tritt eine Verklemmung auf?

Wenn ***ja***, zeichnen Sie den Wartegraphen und geben sie die Operationsnummer an, bei der die Verklemmung auftritt.

Wenn ***nein***, geben Sie die der Ausführung äquivalente serielle Reihenfolge der Transaktionen an.

Es tritt eine Verklemmung bei Operation 12 auf



d) Wie groß ist der Aufwand für eine Verklemmungsentdeckung (im O-Kalkül), wenn der Wartegraph *n* Transaktionen und *m* Wartekanten enthält ?

Tiefensuche O(n+m)

Aufgabe 3: Betriebsmittelverwaltung (2+7+9=18 Punkte)

a) Verklemmung (2 Punkte)

Gegeben sei ein System mit $m = 3$ Prozessen ($i = 1,..,3$) und $n = 2$ Betriebsmitteltypen ($j = 1,2$). Außerdem seien alle aktuellen Anforderungen der Prozesse bekannt. Zur Zeit sind keine Betriebsmittel frei.

Prüfen Sie ob eine Verklemmung vorliegt! Die Situation sei durch die folgenden Größen beschrieben:

Belegungen:

$$B = \begin{matrix} \cancel{0} & 1 \\ 2 & 0 \\ 3 & 3 \end{matrix}$$

Aktuelle Anforderungen:

$$A = \begin{matrix} \cancel{0} & 0 \\ 0 & 2 \\ 1 & 0 \end{matrix}$$

Eine Menge von Prozessen heißt **verklemmt**, wenn es eine Teilmenge gibt, deren **aktuelle** Anforderungen (insgesamt) nicht erfüllbar sind durch den Vorrat der nicht von diesen Prozessen belegt ist.

Zur Verklemmungsentdeckung setzt man den Banker-Algorithmus mit den aktuellen Anforderungen statt den Restanforderungen ein:

DP:= {p1, p2, p3}, f = (0 0)

1. Schritt: a1 = (0 0) ≤ f daraus folgt: DP := {p2, p3} und f = (0 0) + (0 1)

Es gibt keine weiteren Möglichkeiten.

DP ist also ungleich der leeren Menge und damit liegt eine **Verklemmung** vor,

d.h. die Menge {p1, p2, p3}ist verklemmt, nicht nur p2, p3.

Betriebsmittelverwaltung (4 + 3 = 7 Punkte)

Gegeben sei ein System mit $m = 3$ Prozessen ($i = 1,..,3$) und $n = 2$ Betriebsmitteltypen ($j = 1,2$). Ein Teil der vorhandenen Betriebsmittel sei belegt, ein Teil sei noch frei. Außerdem seien die Restanforderungen der Prozesse bekannt. Die Situation sei durch die folgenden Größen beschrieben:

Belegungen: Restanforderungen: frei:

$$B = \begin{matrix} \cancel{2} & 2 \\ 1 & 0 \\ 0 & 4 \end{matrix} \qquad R = \begin{matrix} \cancel{0} & 1 \\ 3 & 0 \\ 2 & 1 \end{matrix} \qquad f = (1 \quad 1)$$

i) Wenden Sie den Banker-Algorithmus an, um zu überprüfen, ob das System sicher ist!

1. Die Restanforderungen von p1 können erfüllt werden, f = (1 1) + (2 2) = (3 3).
2. Die Restanforderungen von p2 können erfüllt werden, f = (3 3) + (1 0) = (4 3).
3. Die Restanforderungen von p3 können erfüllt werden, f = (4 3) + (0 4) = (4 7).

Das System ist sicher, weil eine Reihenfolge gefunden wurde, in der die Prozesse beendet werden können, wobei die Restanforderungen auf einmal zugeteilt werden können.

ii) Wenn das System *sicher* ist, überlegen Sie, welche und wie viele Betriebsmittel man mindestens entfernen muss, damit das System unsicher wird.

Wenn das System *unsicher* ist, geben Sie an, wie viele freie Betriebsmittel ab dem oben beschriebenen Zustand gebraucht werden, damit das System sicher wird.

Da das System sicher ist, minimieren wir f.

Es gibt 2 mögliche Lösungen, wovon eine für die Klausur reichte.

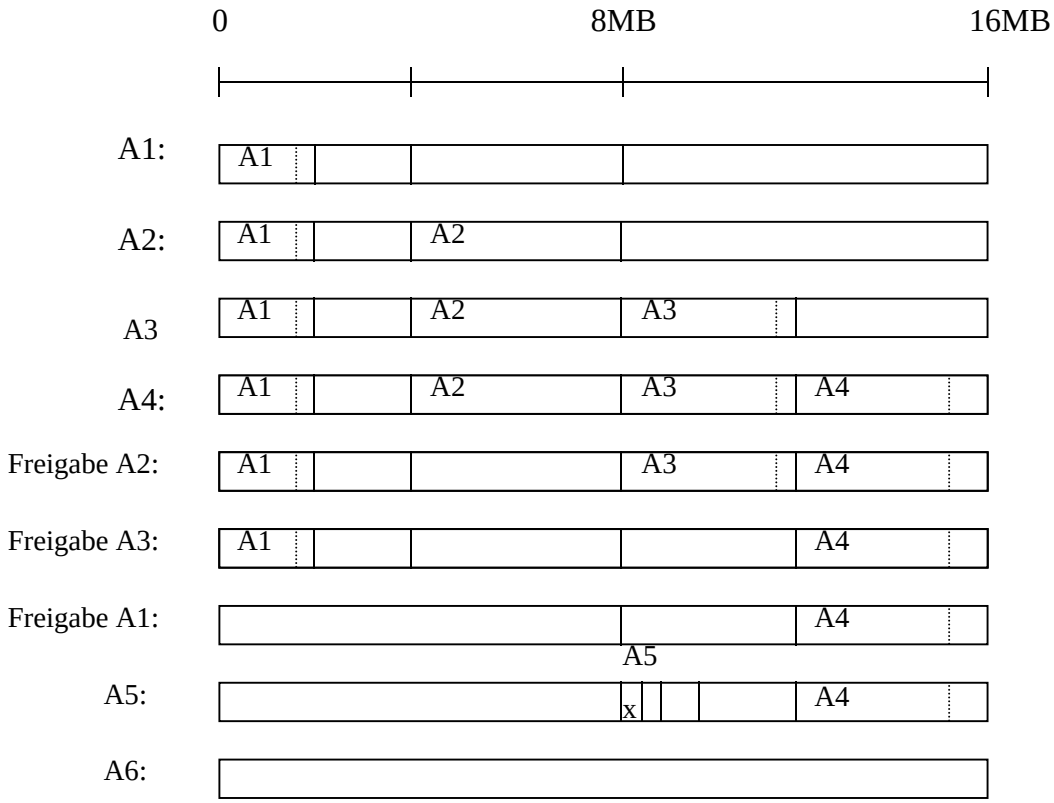
1. Angenommen, f = (0 1), dann Reihenfolge p1: f = (2 3), p3: f = (2 4), damit können aber die Restanforderungen von p2 aber nicht erfüllt werden, damit ist das System unsicher.
2. Mit f = (1 0) kann keiner der Prozesse beendet werden, auch damit ist das System unsicher.

c) **Buddy-Verfahren (5 + 2 + 2 = 9 Punkte)**

Ein 16 MByte großer Speicher werde nach dem Buddy-Verfahren verwaltet. Auf dem zu Beginn leeren Speicher werden der Reihe nach die folgenden Operationen durchgeführt:

1. Anforderung 1: 1,5 MB
2. Anforderung 2: 4 MB
3. Anforderung 3: 3,5 MB
4. Anforderung 4: 3 MB
5. Freigabe Anforderung 2 (4 MB)
6. Freigabe Anforderung 3 (3,5 MB)
7. Freigabe Anforderung 1 (1,5 MB)
8. Anforderung 5: 0,5 MB
9. Anforderung 6: 9 MB

i) Können alle Anforderungen erfüllt werden ? Geben Sie die Anordnung im Speicher an! Nutzen Sie dafür die unten angegebenen rechteckigen Felder.



A 6 kann nicht erfüllt werden.

ii) Was bedeutet interner Verschnitt ?

Meistens wird Speicher in Vielfachen von festen Elementargrößen vergeben. Anforderungen werden daher auf das nächste Vielfache gerundet. Dadurch entsteht Speicherplatz, der als belegt gekennzeichnet ist, aber nicht benutzt wird. Man nennt solchen Speicherplatz internen Verschnitt.

Formal ist er definiert als Verhältnis der Erwartungswerte der Anzahl belegter aber ungenutzter zur Anzahl der belegten Speichereinheiten.

iii) Wie groß ist der interne Verschnitt zu dem Zeitpunkt bevor die letzte Anforderung eingegangen ist ?

Der konkrete Verschnitt ist das Verhältnis der Anzahl belegter aber ungenutzter zur Anzahl der belegten Speichereinheiten.

Verschnitt gibt es bei Anforderung 4, nicht aber bei Anforderung 5, denn es wird weiter geteilt:

1 MB / (4 MB + 0,5 MB) = 1 / 4,5 = 2/9 = 22,2 %

Aufgabe 4: Speicherverwaltung (8+12+5=25 Punkte)

a) In dieser Aufgabe werden 8 Aussagen gemacht. Ordnen Sie jeder Aussage eine der angegebenen Auswahlstrategien durch Ankreuzen in das entsprechende Feld zu. Für jede richtige Zuordnung gibt es einen Punkt, für eine falsche Zuordnung wird ein Punkt abgezogen. Mehrfachnennungen können möglich sein.

Ausgelagert wird...	FIFO	LFU	LRU	RNU	Keine
...die Seite, die schon am längsten im Speicher ist.	☒	❑	❑	❑	❑
...eine Seite, die zufällig ausgewählt wurde.	❑	❑	❑	❑	☒
...eine Seite, auf die nur lesend zugegriffen wurde.	❑	❑	❑	❑	☒
...eine Seite, die innerhalb eines vorgegebenen Zeitraums nicht mehr referenziert wurde.	❑	❑	❑	☒	❑
...die Seite, die am wenigsten häufig referenziert wurde.	❑	☒	❑	❑	❑
...die Seite, die am längsten nicht mehr benötigt wird.	❑	❑	❑	❑	☒
...die Seite, die am längsten nicht mehr referenziert wurde.	❑	❑	☒	❑	❑
...die Seite, die am längsten nicht geschrieben wurde.	❑	❑	❑	❑	☒

b) Betrachten Sie die folgende Tabelle. Ein Thread hat vier Speicherseiten allokiert. Die folgenden Nummern sind dezimal, und wir beginnen bei Null zu zählen. In der Tabelle wird der Zeitpunkt des Einlagerns und der Zeitpunkt der letzten Referenz protokolliert, weiterhin gibt es ein Feld für das „referenziert"-Bit (R) und ein Feld für das „modifiziert"-Bit (M) für jede Seite. Die Zeiten sind Clock-Ticks seit dem Start des Threads zur Zeit 0. Es handelt sich **nicht** um die Anzahl der Ticks seit Auftreten des jeweiligen Ereignisses.

Virtual Page Number	Page Frame	Time loaded	Time referenced	R Bit	M Bit
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

Ein Seitenfehler auf der virtuellen Seite 4 tritt auf. Der Inhalt welches Page Frame wird unter Berücksichtigung der nachfolgenden Strategien auf der nächsten Seite ersetzt. Begründen Sie Ihre Entscheidung.

i) FIFO

Page Frame 3 wird ausgelagert, da seine Einlagerung zum Zeitpunkt 20 am längsten zurückliegt.

ii) LRU

Page Frame 1 wird ausgelagert, da seine letzte Referenzierung zum Zeitpunkt 160 am längsten zurückliegt.

iii) Clock

Page Frame 0 wird ausgelagert, da sie die drittälteste Seite ist und das Referenzbit auf 0 gesetzt ist.

iv) Belady's optimaler Algorithmus mit diesem Referenzstring: 4,0,0,0,2,4,2,1,0,3,2

Page Frame 3 wird ausgelagert, da die Seite 3 erst in der entferntesten Zukunft wieder benötigt wird (zuvor werden die Seiten 4, 0, 2, und 1 benötigt),

v) Betrachten Sie diesen Referenzstring ausgehend vom Zustand in obenstehender Tabelle direkt vor dem Seitenfehler: 4,0,0,0,2,4,2,1,0,3,2.

Wie viele Seitenfehler treten auf, wenn die Seitenauswahl durch ein Rückwärtsfenster (working set) der Größe 4 beschränkt wird. Erklären Sie, warum und wann jeder Seitenfehler auftritt.

Referenzstring		4	0	0	0	2	4	2	1	0	3	2
Seiten im Speicher in LRU-Reihenfolge	3	4	0	0	0	2	4	2	1	0	3	2
	0	3	4	4	4	0	2	4	2	1	0	3
6 Seitenfehler (rot)	2	0	3	3			0	0	4	2	1	0
Blau: Ersetzte Seiten	1	2								4	2	1

c) Betrachten Sie im folgenden ein Rechensystem mit byteweise adressierbaren 256 MByte Hauptspeicher (physikalischer Adressraum) und maximal 4 GByte virtuell adressierbaren Speicher. Die maximale Segmentgröße betrage 8 MByte.

- Berechnen Sie die Länge eines Eintrags in der Segmenttabelle und einer virtuellen Adresse.

Eintrag der Segmenttabelle

Segment-anfang	Segment-länge
28 Bit	23 Bit

virtuelle Adresse

Segment	Offset
9 Bit	23 Bits

- Welche physikalische Adresse entspricht der virtuellen Adresse

111010	101111

mit dem folgenden Eintrag Nr. 111010 in der Segmenttabelle.

1111111111110000	101100100000

Hinweis: Die angegebenen Zahlen sind Binärzahlen. Die führenden Nullen wurden weggelassen.

Grundlegend ist anzumerken, dass im Gegensatz zur Seitenadressierung (paging), bei der Segmentadressierung mit Speicherbereichen unterschiedlicher Länge gearbeitet wird. Daher Kann ein Segment an einer beliebigen physikalischen Adresse beginnen, und es ist erforderlich, die Länge eines Segmentes zu kennen, um Zugriffsverletzungen zu erkennen.

Die Segmentanfangsadresse ist daher eine vollständige physikalische Adresse der Länge *lg(256MB) = 28 Bit.*

Zur Angabe der Segmentlänge benötigen wir *lg(8MB) = 23 Bit.*

Da das System maximal *4GB / 8MB = 512* Segmente verwalten soll, ist die Segmentnummer *lg(512) = 9 Bit* lang.

Der Offset adressiert innerhalb eines Segmentes und ist daher *lg(8MB) = 23 Bit* lang.

Die physikalische Adresse ergibt sich durch Addition des Offset der virtuellen Adresse zur Segmentanfangsadresse desjenigen Eintrages in der Segmenttabelle, der durch das Segmentfeld der virtuellen Adresse referenziert wird, d.h.

1111 1111 1111 0000

+ 10 1111

= 1 0000 0000 0001 1111

(hexadezimal: FFF0 + 2F = 1 001F)

Aufgabe 5: Scheduling (4+10+4+4+10=32 Punkte)

a) Erklären Sie den Begriff Prioritätsinvertierung. Geben Sie dazu ein Beispiel.

Ein Prozess hoher Priorität wartet auf ein Betriebsmittel, dass ein Prozess niedriger Priorität besitzt, während ein dritter Prozess mittlerer Priorität läuft und damit verhindert, dass der Prozess mit der niedrigen Priorität Rechenzeit erhält und damit das Betriebsmittel freigeben kann.

b) Es sind vier Prozesse mit ihren Startzeitpunkten und der benötigten Rechenzeit gegeben:

Prozess	Startzeitpunkt	Benötigte Rechenzeit
P1	0	5
P2	3	4
P3	5	2
P4	6	1

Geben sie die Reihenfolge der Prozesse an, indem Sie die Prozessnummern in die entsprechenden Kästchen eintragen. Beim RR-Verfahren wird angenommen, daß sich ein Prozeß zu seinem Startzeitpunkt bereits in der Ready-Warteschlange befindet.

- nach dem FCFS Verfahren

P1	*P1*	*P1*	*P1*	*P1*	*P2*	*P2*	*P2*	*P2*	*P3*	*P3*	*P4*

- nach dem RR -Verfahren mit einer Zeitscheibengröße von 1

P1	*P1*	*P1* *[P2]*	*P2* *P1*	*P1* *P2,[P3]*	*P2* *P3,P1,[P4]*	*P3* *P1,P4,P2*	*P1* *P4,P2,P3*	*P4* *P2,P3*	*P2* *P3*	*P3* *P2*	*P2*

- nach dem EDF Verfahren mit den Sollzeitpunkten d1=7, d2=11, d3=7, d4=8

P1	*P1*	*P1*	*P1*	*P1*	*P2*	*P2*	*P3*	*P3*	*P4*	*P2*	*P2*

c) Erläutern Sie die Begriffe „Eingangstor“ und „Ausgangstor“. Geben Sie je ein praktisches Beispiel an.

Ein Eingangstor ist ein n:1 Kanal, welcher einem festen Empfänger zugeordnet ist. Unterschiedliche Sender können durch den Kanal diesem Empfänger Nachrichten schicken. Ein verbreitetes Beispiel hierfür ist die Hauptnachrichtenschlage in ereignisorientierten Systemen.

Ein Ausgangstor ist ein 1:n Kanal. Dort können verschiedene Empfänger Nachrichten abholen. Ein Beispiel hierfür ist die Auftragshaltung in einem Server mit mehreren Prozessen zur Bearbeitung. Dort können diese Prozesse ihre Aufträge aus so einem Kanal beziehen.

d) Erläutern Sie das „Fork/Join Prinzip“. Geben Sie dazu ein Beispiel an.

Manchmal können mehrere Teile einer Aufgabe gleichzeitig ausgeführt werden. Zu diesem Zweck werden zunächst die einzelnen Teile der Aufgabe an andere Prozesse zur Bearbeitung übergeben („Fork“), und danach werde die Ergebnisse wieder eingesammelt („Join“). Als ein Beispiel könnte man sich einen Primzahltest Vorstellen, der für n alle Zahlen t aus {2,3,..,Floor(sqrt(n))} probiert. Man könnte diese Rechnung mit m Prozessoren (unter der Annahme, dass der Test für jedes t gleich lange dauert) um den Faktor m beschleunigen, indem man m gleich grosse Intervalle auf den m Prozessoren prüft.

e) Ein Programm verarbeitet Videodaten, indem es die Videodaten zunächst digitalisiert, dann nachbearbeitet und zuletzt anzeigt. Die entsprechenden Phasen des zugehörigen Prozesses seien mit A, B, C bezeichnet. Auf dem Referenzsystem ergeben sich bei folgende Ausführungszeiten pro Bild:

Phase	Ausführungszeit
A	20 ms
B	80 ms
C	60 ms

Bei sequentieller Ausführung ergibt sich somit ein Zeitbedarf von 160ms. Der Durchsatz ist somit 6,25 Bilder/s.

i) Verbessern Sie diesen Durchsatz durch den Einsatz von mehr Prozessoren und geben Sie den resultierenden Durchsatz an. Phase 1 kann wegen der notwendigen Hardware nur einmal in dem Computersystem vorhanden sein!

Wenn man Phase B 4 mal repliziert kann im Mittel alle 20ms ein Bild Phase fertig bearbeitet verlassen. Geht man nun analog bei Phase C vor und repliziert diese 3 mal, so kann auch in Phase C im Mittel alle 20 ms ein Bild verarbeitet werden. Verschaltet man nun diese Phasen B und C mit Phase A zu einem Fließband, so kann mit genügend Prozessoren alle 20ms ein Bild verarbeitet werden, was 50 Bildern/s entsprich. Das ist wegen Phase A auch der maximal erzielbare Durchsatz.

ii) Welche Techniken haben Sie verwendet ?

Es wurde das Fließbandprinzip(Pipelining) und Replikation (Cloning) angewandt.

iii) Was ist die maximal sinnvolle Anzahl an Prozessoren ? Wie hoch ist der Durchsatz mit weniger als der maximalen Anzahl an Prozessoren? Wie hoch ist der Durchsatz mit mehr als der maximalen Anzahl an Prozessoren ?

Durch Zählen der verwendeten Prozesse in den Phasen ergibt sich die maximal auslastbare Zahl an Prozessoren mit 1+4+3=8. Stehen weniger Prozessoren zur Verfügung, so wird man linear langsamer, und es ergibt sich ein Durchsatz von 50*Prozessoren/8 Bildern/s. Man kann nicht mehr als die besagten 8 Prozessoren auslasten, weil ja Phase A nur 50 Bilder/s produziert. Somit kann man keinen Nutzen aus zusätzlichen Prozessoren ziehen.