

KNr.

MNr.

Zuname, Vorname

(Ges.)(100)

1.)(30)

2.)(25)

3.)(25)

4.)(20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

Die Werkstätte der Firma 'Ho Ruck' hat zusätzlich zu ihren Arbeitsgeräten eine neue Drehbank bekommen. Jedes Gerät kann zu jeder beliebigen Zeit gebraucht werden. Unglücklicherweise ist der Stromverbrauch, wenn alle elektrischen Geräte eingeschaltet sind, jetzt grösser als die Zuleitung zur Werkstätte verkraftet. Als Provisorium installiert der Elektriker ein System, in dem Geräte nur eingeschaltet werden können, solange der Strom in der Leitung nicht zu gross wird.

a) (4)

Welche Art von Synchronisation wird dafür benötigt?

- ☐ keinerlei Synchronisation
 ☐ wechselseitiger Ausschluss
☐ Bedingungssynchronization
 ☐ Mutual & Condition Synchronisation

Die Nennleistung der Zubringerleitung ist **16 kW**.

Stk	Maschinenart	Leistungsbedarf / Stk
3	Hebebühnen	3.5kW
1	Bestoßmaschine	4.5kW
1	neue Drehbank	6.0kW

Tabelle 1: Auflistung der Verbraucher

b) (4)

Der Elektriker installiert folgende Routinen für ein Gerät mit einer Leistung von r kW.
Achtung: Eine Leistungseinheit stellt **0.5 kW** dar.

```
vor_dem_Einschalten(Leistungsbedarf r){
    for(int i=0;i<(2*r);i++){
        P(A);
    }
}
```

```
nach_dem_Ausschalten(Leistungsbedarf r){
    for(int i=0;i<(2*r);i++){
        V(A);
    }
}
```

Das Semaphor A wurde mit der maximal verfügbaren Anzahl an Leistungseinheiten initialisiert.

```
initS(A,32);
```

Das schlaue Lehrmädchen meint dazu: „Mir scheint, das ist nicht der Weisheit letzter Schluss!“ Welche(s) Problem(e) kann/können bei dieser Lösung auftreten?

- | | |
|----------------------------------|---|
| <input type="radio"/> Deadlock | <input type="radio"/> Busy Waiting |
| <input type="radio"/> Starvation | <input type="radio"/> Überlastung der Zuleitung |

c) (22)

Glücklicherweise stellt das verwendete System neben Semaphoren auch Routinen für Eventcounter zur Verfügung:

- `initEC(E,Zahl)` legt einen Eventcounter an und initialisiert ihn mit `Zahl`.
- `await(E,Zahl)` wartet solange bis der Eventcounter `E` grösser gleich dem Wert von `Zahl` ist.
- `read(E)` liest den Wert des Eventcounters aus.
- `advance(E,Zahl)` erhöht den Eventcounter um den Wert `Zahl`.

Gemeinsam erstellen die beiden nun ein Programm, welches einen Semaphor und zwei Eventcounter verwendet.

Der erste Eventcounter (`ECused`) entspricht der Anzahl der verbrauchten Ressourcen im System. Der zweite Eventcounter (`ECavail`) entspricht der Anzahl der zurückgegebenen Ressourcen plus der noch verfügbaren Ressourcen im System.

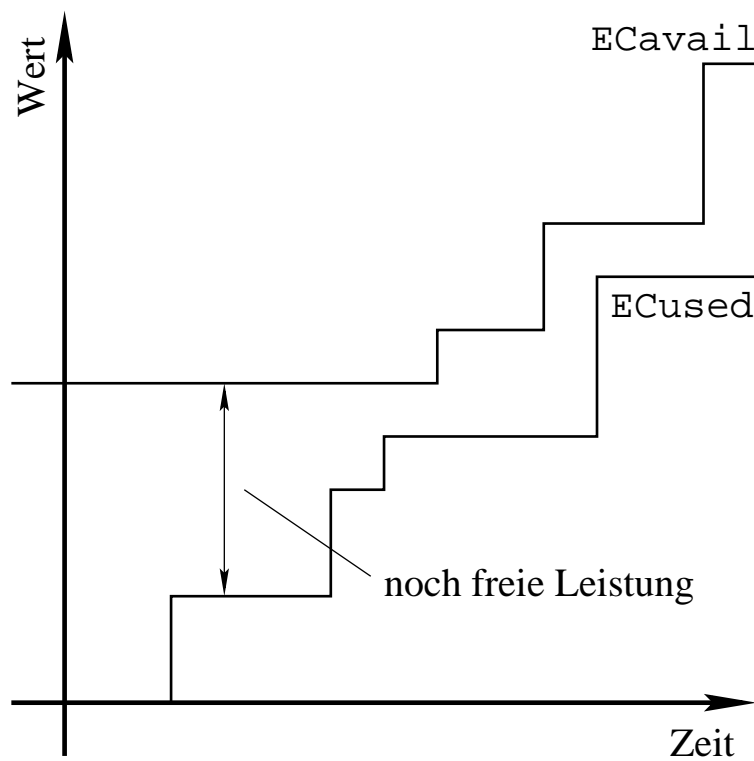


Abbildung 1: Bild eines Leistungsbedarfes

1. Ein Gerät wartet zunächst bis EC_{avail} mindestens den Wert Leistungsbedarf des Gerätes plus dem Wert von EC_{used} erreicht hat.
2. Danach testet das Gerät, ob inzwischen ein anderes Gerät möglicherweise weitere Ressourcen verbraucht hat. Dazu greift es auf beide EC zu und prüft, ob die Differenz mindestens dem Leistungsbedarf des Gerätes entspricht. Ist die Differenz kleiner als der Leistungsbedarf des Gerätes, so muss es den Vorgang bei Schritt 1 wiederholen. Ansonsten erhöht das Gerät EC_{used} um den Wert seines Leistungsbedarfes und schaltet sich ein.
3. Wenn das Gerät ausgeschaltet wird, erhöht es EC_{avail} um den Wert seines Leistungsbedarfes.

Beachten Sie, dass manche Programmteile möglicherweise exklusiv ausgeführt werden müssen. Weiters darf kein `goto` verwendet werden (umwandeln in eine Schleife). Wie kann eine solche Lösung aussehen?

Initialisierung:

```
vor_dem_Einschalten(Leistungsbedarf r){
```

```
}
```

```
nach_dem_Ausschalten(Leistungsbedarf r){
```

```
}
```

2 Speicherverwaltung (25)

a) Lokalität von Prozessen – Working Sets (11)

Das Working Set Modell beschreibt das Speicherzugriffsverhalten von Prozessen. In diesem Modell ist das Working Set $W(t, \Delta)$ definiert als die Menge der Speicherseiten, die in den letzten Δ Zeiteinheiten vor dem Zeitpunkt t referenziert wurden.

Die folgende Tabelle beschreibt eine Folge von Speicherseitenzugriffen eines Prozesses. In Spalte 2 jeder Zeile steht die Nummer der Seite, auf die in der Zeitscheibe $(t - 1, t]$ zugegriffen wird. Geben Sie in den leeren Feldern die Working Sets $W(t, \Delta)$ für Δ gleich 2, bzw. 5 an.

t	Seitennr.	$W(t, 2)$	$W(t, 5)$
1	29		
2	12		
3	13		
4	22		
5	29		
6	13		
7	12		
8	45		
9	29		
10	12		
11	12		

b) Kombination aus Segmentierung und Paging (14)

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 in der Folge betrachtete Speicherverwaltungssystem verwendet zur Adressierung 20-bit Adressen. Für das Paging sind alle Seitenrahmen 256 Bytes (sedezimal (hexadezimal) 0x00100) groß. Das verwendete Adressformat ist folgendes:

Seg# (8 bit)	Page# (4 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 Sedezimalzahlen (Hexadezimalzahlen) angegeben):

Segmenttabelle		
Seg#	Base	Length
0x03	0x0AFFE	0x3
0xAC	0x7D00F	0x1
0x3D	0x1CE00	0x7
0x00	0x00000	0xD

Seitentabelle	
Address	Frame#
0x00000	0x123
0x00001	0x124
0x00002	0x376
...	...
0x0AFFE	0xEEE
0x0AFFF	0xABC
0x0B000	0x000
0x0B001	0xDDD
...	...
0x1CDFF	0x666
0x1CE00	0x7DA
...	...
0x1CE05	0x5A7
0x1CE06	0x5AC
...	...
0x7D00E	0x471
0x7D00F	0xAAA
0x7D010	0x815
...	...

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):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)		
0xAC2FF			
0x01111			
0x002FF			
0x3D6D7			
0xFF000			
0x0328A			
0x00D33			

3 Scheduling (25)

a) Rate-Monotonic Scheduling (12)

Task	Laufzeit (ms)	Periode (ms)
T1	2	20
T2	2	10
T3	3	13
T4	2	12
T5	3	23

Folgendes wird angenommen:

- Alle Tasks sind periodisch
- Deadline = Periode
- Scheduling Overhead wird vernachlässigt
- Alle Tasks sind unabhängig

Ist dieses Taskset nach der notwendigen Scheduling-Bedingung unter Verwendung von Rate-Monotonic Scheduling (RMS) schedulbar?

☐ Ja ☐ Nein

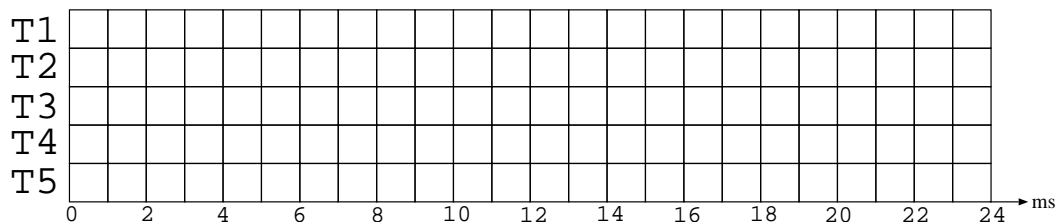
Bitte begründen Sie Ihre Antwort:

Ist dieses Taskset nach der hinreichenden Scheduling-Bedingung unter Verwendung von RMS schedulbar?

☐ Ja ☐ Nein

Bitte begründen Sie Ihre Antwort:

Das oben angegebene Taskset soll mit einem Rate-Monotonic Scheduling Algorithmus für den Worst Case¹ gescheduled werden. Bitte vervollständigen Sie den gesamten Zeitbereich der folgenden Skizze bzw. kennzeichnen Sie eine Verletzung einer Deadline.



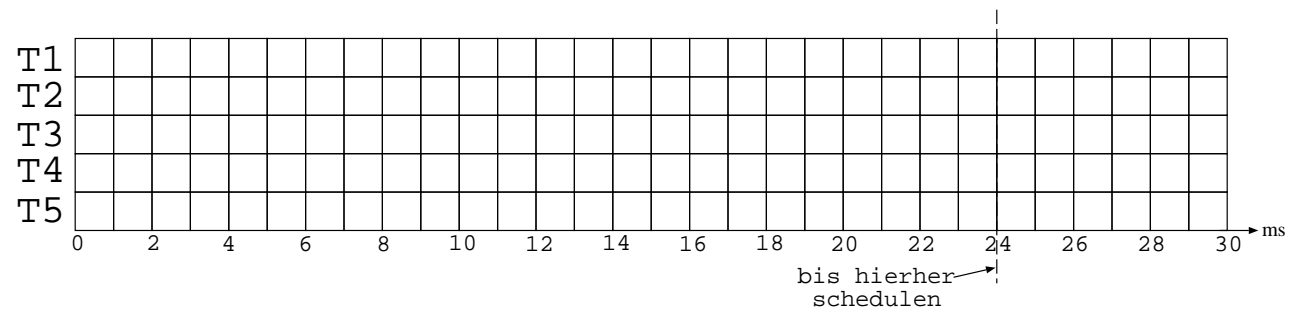
¹Im Worst Case beginnen alle Perioden bei 0.

b) Earliest Deadline First Scheduling (EDF) (8)

Schedulen Sie das gegebene Task Set nach dem Earliest Deadline First Verfahren *mit Pre-emption* bis zu $t = 24$ ms. Als Arrival Time können Sie für alle Tasks den Zeitpunkt 0 annehmen, der Scheduling Overhead ist wiederum 0. Weiters sind die angegebenen Deadlines zu beachten. Sollte das Taskset nicht gescheduled werden können, so zeichnen Sie bitte die Verletzung der Deadline eindeutig ein.

Die Deadlines für die Tasks sind:

Task	Laufzeit (ms)	Periode (ms)	Deadline (ms)
T1	2	20	12
T2	2	10	10
T3	3	13	13
T4	2	12	7
T5	3	23	8



c) Fragen zu Scheduling (5)

Wenn ein Test unter Verwendung von Rate-Monotonic Scheduling aufgrund der notwendigen Scheduling-Bedingung ergibt, dass ein Taskset schedulbar ist, aber ein Test aufgrund der hinreichenden Scheduling-Bedingung ergibt, dass ein Taskset *nicht* schedulbar ist, könnte es trotzdem sein, dass das Taskset schedulbar ist oder ist das unmöglich?

- ☐ Ja, ein solches Taskset könnte schedulbar sein.
- ☐ Nein, es ist unmöglich, dass ein solches Taskset schedulbar ist

Kreuzen Sie bei der folgenden Aussage bitte an, ob diese richtig oder falsch ist.

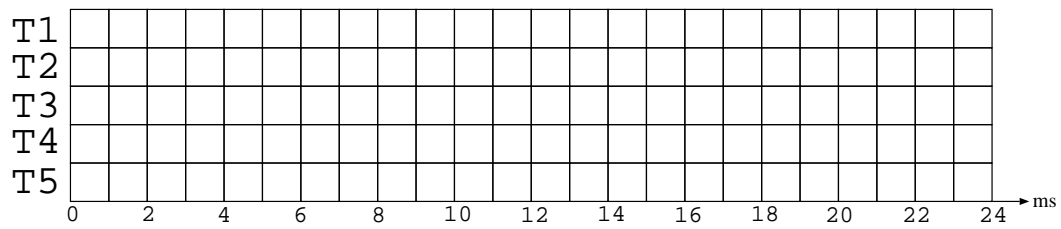
Bei Scheduling nach dem Round-Robin-Verfahren werden I/O-intensive Prozesse gegenüber rechenzeit-intensiven (mit wenig I/O) benachteiligt.

- ☐ richtig ☐ falsch

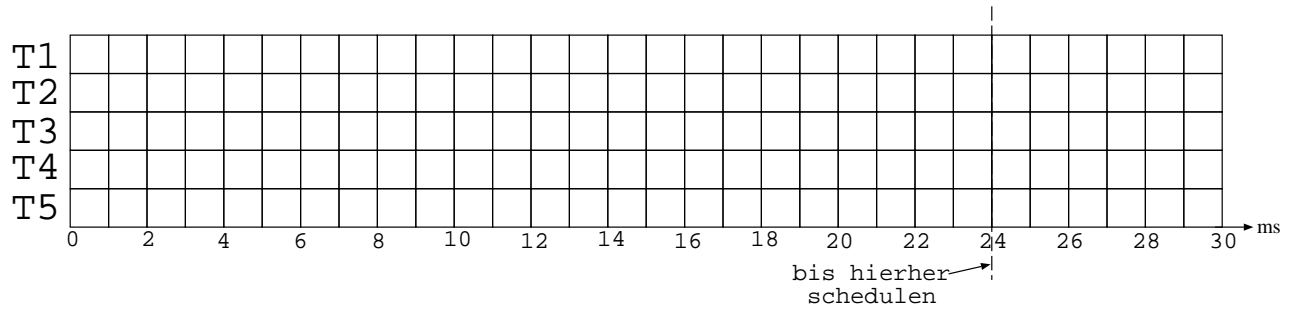
Ersatzvorlagen

Bitte streichen Sie die anderen Vorlagen durch, wenn Sie diese Vorlagen verwenden.

Ersatz a)



Ersatz b)



4 Security (20)

a) (10)

Was versteht man unter einem *aktiven* Security Threat?

Welche *aktiven* Security Threats gibt es?

Was versteht man unter einem *passiven* Security Threat?

Welche *passiven* Security Threats gibt es?

b) (4)

Was versteht man unter *Audit Records*?

Wozu dienen *Audit Records*?

c) (6)

Wann nennt man einen Verschlüsselungsalgorithmus *symmetrisch*?

Wann nennt man einen Verschlüsselungsalgorithmus *asymmetrisch*?

Welchen Vorteil hat die symmetrische gegenüber der asymmetrischen Verschlüsselung?

Welchen Vorteil hat die asymmetrische gegenüber der symmetrischen Verschlüsselung?

KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (25)

2.) (20)

3.) (30)

4.) (25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen A , B und C benötigen. Jede der drei Ressourcen ist nur einmal vorhanden und kann immer nur von einem Prozess belegt werden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt ihrer Abarbeitung sie jeweils die einzelnen Ressourcen benötigen. Die Anforderungen von Prozess P_1 sind entlang der x -Achse, die Anforderungen von Prozess P_2 entlang der y -Achse aufgetragen.

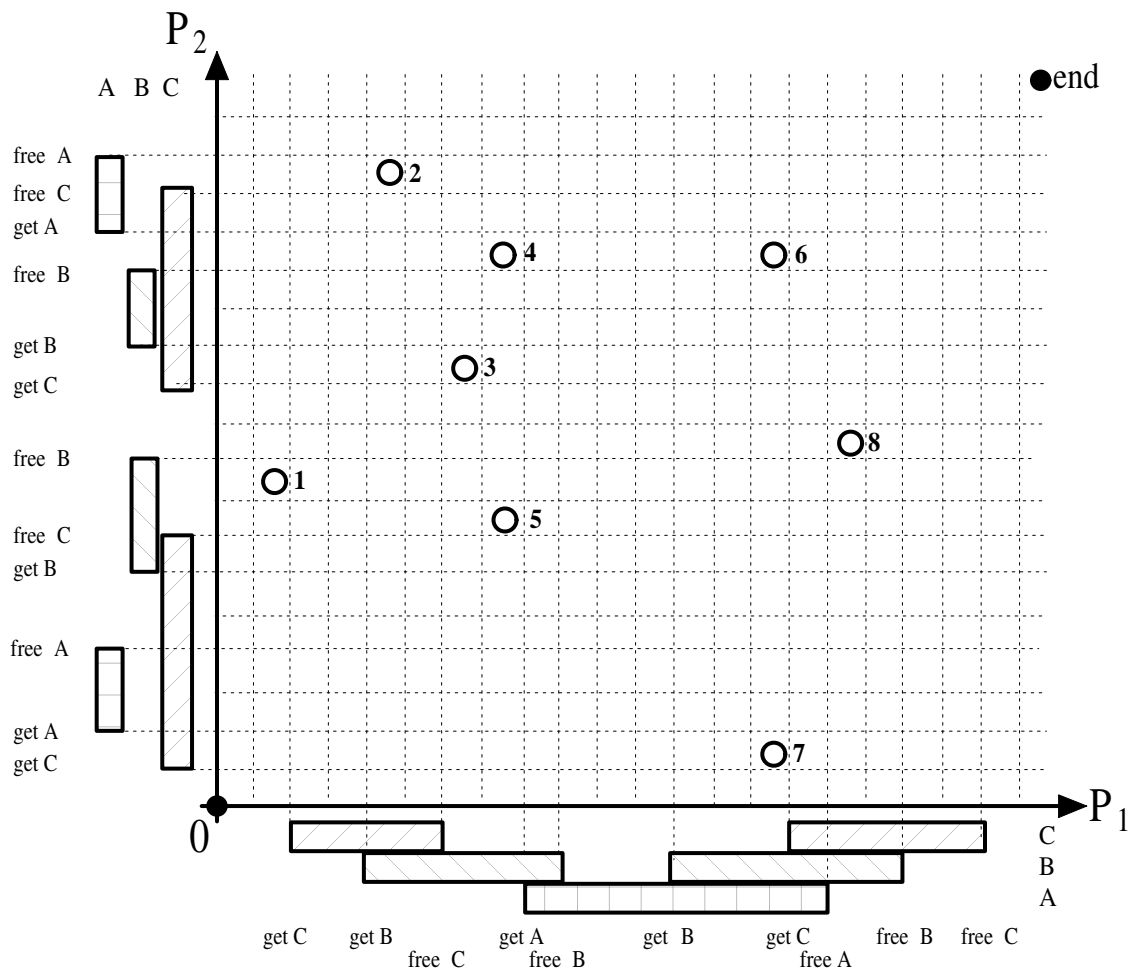
Der Fortschritt der Prozesse P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen dem Punkt θ und dem Punkt end in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

a) (8)

Umranden Sie in der Grafik jene Bereiche deutlich, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

b) (9)

Umranden und schraffieren Sie in der Grafik die Bereiche, die ein Kantenzug nicht passieren darf, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche zusätzlich deutlich mit einem "D".



c) (8)

In der Grafik sind acht Punkte durch einen Kreis gekennzeichnet und nummeriert. Tragen Sie in der folgenden Tabelle für jeden Punkt ein, ob eine durch diesen Punkt gehende Abarbeitung der beiden Prozesse ab dem Punkt (a) immer deadlockfrei erfolgt, (b) zu einem Deadlock führen kann aber nicht muss, (c) immer zu einem Deadlock führt, oder (d) gar nicht möglich ist.

	1	2	3	4	5	6	7	8
(a) immer deadlockfrei	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
(b) Deadlock möglich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
(c) sicherer Deadlock	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
(d) Abarbeitung unmöglich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2 Scheduling (20)

a) Earliest Deadline First Scheduling (EDF) (10)

Schedulen Sie das gegebene Task Set nach dem Earliest Deadline First Verfahren *mit Preemption* bis zum Zeitpunkt $t = 26$ ms unter Beachtung der angegebenen Deadlines.

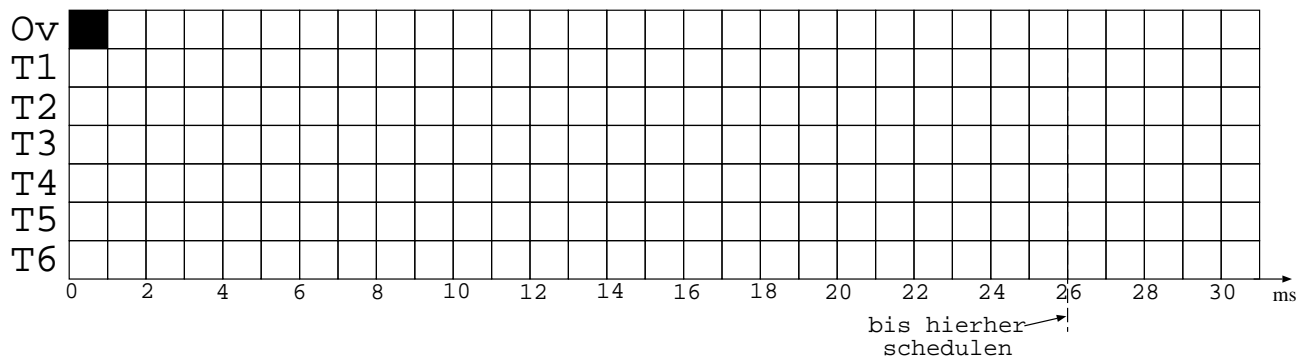
Sollte das Taskset nicht gescheduled werden können, so zeichnen Sie das gültige Schedule bitte bis zum Zeitpunkt der Verletzung der Deadline und kennzeichnen Sie diesen Zeitpunkt.

Das Taskset:

Task	Laufzeit (ms)	Periode (ms)	Deadline (ms)
T1	1	11	9
T2	2	13	10
T3	1	16	16
T4	1	12	7
T5	2	25	14
T6	3	13	12

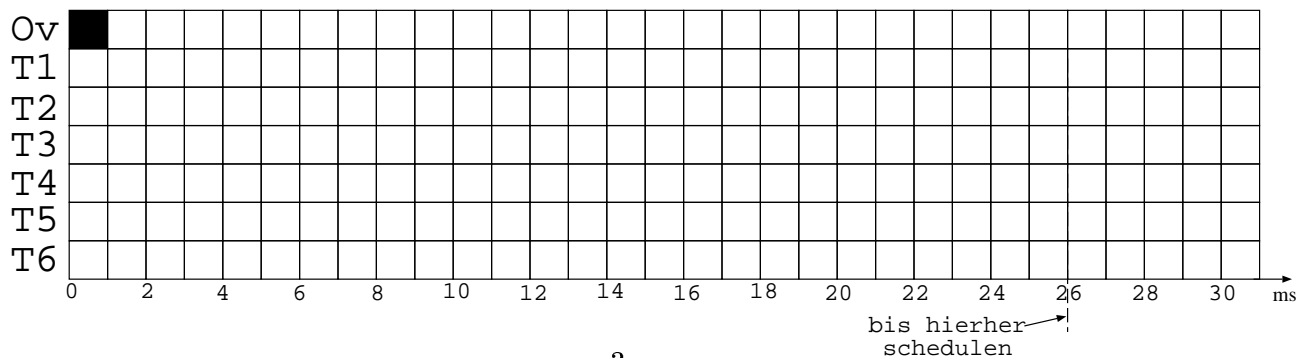
- Alle Tasks sind periodisch.
- Der Scheduling Overhead beträgt 1 ms.
- Alle Tasks sind unabhängig.
- Die erste Ankunftszeit (arrival time) ist für alle Tasks der Zeitpunkt 0.

Bitte schedulen Sie das obige Taskset.



Ersatzvorlage

Bitte streichen Sie die andere Vorlage durch, wenn Sie diese Vorlage verwenden.



b) Fragen zu Scheduling (10)

Werden bei Rate-Monotic Scheduling die Prioritäten den Tasks statisch oder dynamisch zugeordnet?

☐ dynamisch

☐ statisch

Begründung:

Was sind Vorteile von Scheduling-Mechanismen, die die Priorität statisch zuweisen?

Ist Scheduling nach dem First-In-First-Out Prinzip preemptive?

☐ ja

☐ nein

Begründung:

Fragen zu Scheduling nach dem Round-Roubin-Verfahren:

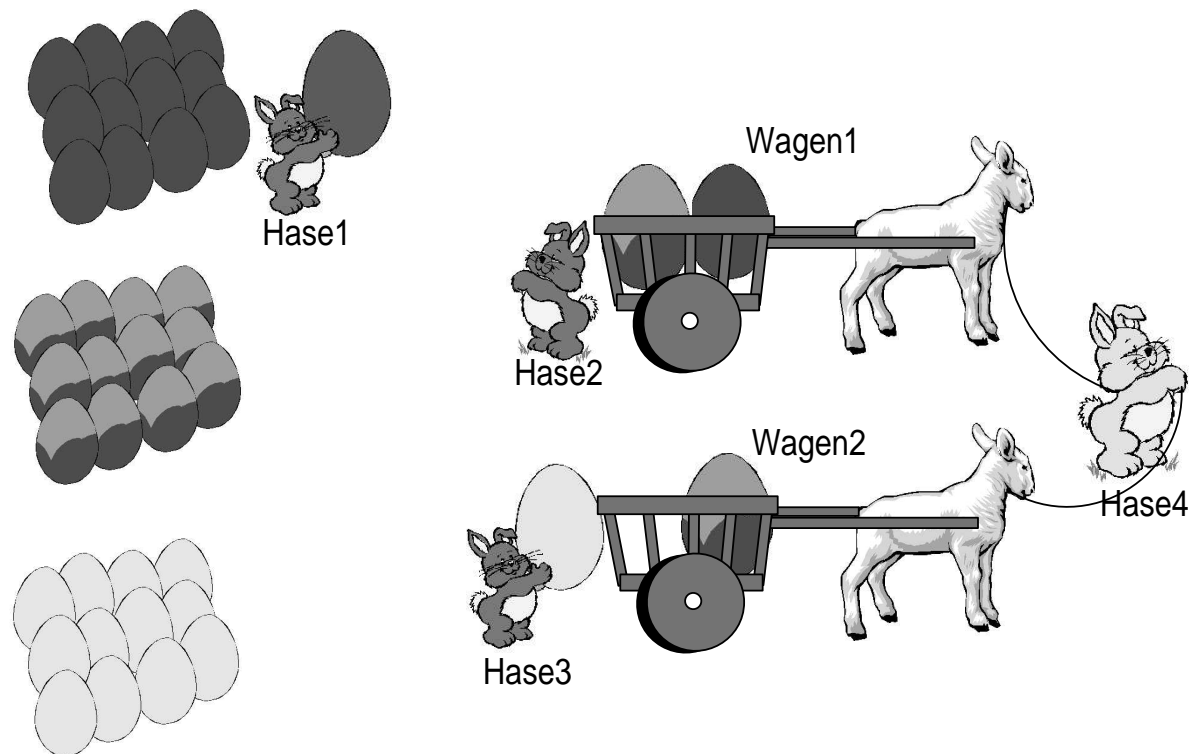
Was sind die Vorteile von großen Zeitquanten?

Was sind die Nachteile von großen Zeitquanten?

Was sind die Vorteile von kleinen Zeitquanten?

Was sind die Nachteile von kleinen Zeitquanten?

3 Synchronisation (30)



Immer wenn Ostern vor der Tür steht, herrscht bei den Osterhasen Hochkonjunktur. Eine Abteilung von vier Osterhasen ist mit dem Beladen und Versenden von bemalten Ostereiern beauftragt. Die Osterhasen sollen dabei nach folgendem Schema arbeiten:

- Jeder der drei Osterhasen *Hase1*, *Hase2* und *Hase3* hat einen eigenen unendlichen Eiervorrat, von dem er immer genau ein Ei entnehmen kann.
- Die Eier werden auf zwei Wägen geladen, von denen jeder genau zwei Eier fassen kann.
- *Hase1* belädt immer Wagen 1, *Hase3* belädt immer Wagen 2, *Hase2* belädt **abwechselnd** Wagen 1 und Wagen 2. *Hase4* beteiligt sich nicht am Beladen.
- Die beiden Wägen können unabhängig voneinander beladen werden, aber es kann immer nur **ein Ei nach dem anderen** auf einen Wagen geladen werden.
- Sind beide Wägen voll, so werden die Zuglämmer von *Hase4* zum Kunden geführt, die anderen Hasen müssen dann warten, bis *Hase4* mit den leeren Wägen wieder zurückkehrt.

Synchronisieren Sie den Arbeitsablauf der vier Hasen mittels **Semaphore**. Sie können davon ausgehen, dass am Anfang beide Wägen leer bereitstehen und die Beladehasen unbeladen bei ihren Stapeln stehen. *Hase2* belädt zuerst Wagen 1. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`gehezu(Ziel)` Bewegt den Hasen zum *Ziel*. Als Ziel kann `Stapel1`, `Stapel2`, `Stapel3`, `Wagen1`, `Wagen2` und für *Hase4* `Beladestation` und `Kunde` angegeben werden.

`nimmEi()` Lässt den Hasen ein Ei aufnehmen. Funktioniert nur, wenn der Hase neben einem Stapel steht.

`beladeWagen(Wagen)` Der Hase belädt den Wagen mit dieser Funktion. Als Argument kann für *Wagen* `Wagen1` und `Wagen2` angegeben werden. Diese Funktionen beinhaltet keine Synchronisationskonstrukte!

`entladeWagen(Wagen)` Entlädt einen Wagen. Diese Arbeit muss leider von *Hase4* allein erledigt werden. Als Argument kann für *Wagen* `Wagen1` und `Wagen2` angegeben werden.

a) Initialisierungen (6)

Initialisieren Sie die notwendigen Semaphore:

b) (12)

Entwerfen Sie die Prozesse, die in den Hasen *Hase1* und *Hase2* ablaufen:

Prozess *Hase1*:

Prozess *Hase2*:

```
do forever() {
```

```
do forever() {
```

```
}
```

```
}
```

c) (12)

Entwerfen Sie die Prozesse, die in den Hasen *Hase3* und *Hase4* ablaufen:

Prozess *Hase3*:

```
do forever() {
```

Prozess *Hase4*:

```
do forever() {
```

```
}
```

```
}
```

4 Speicherverwaltung (25)

a) Segmentierung (6)

Das im folgenden beschriebene Speicherverwaltungssystem verwendet zur Adressierung 32-bit Adressen (virtuell & physikalisch). Für die angegebenen virtuellen Speicheradressen sind die entsprechenden physikalischen Adressen zu ermitteln. Von den angegebenen Adressen sind die niederwertigen 16 Bit der Offset der Adresse. Die Verwendung der höherwertigen 16 Bit ist aus dem angegebenen Adressformat ersichtlich.

Alle Werte sind als Hexadezimalzahlen angegeben. Ergibt sich bei der Umwandlung aufgrund einer ungültigen Segmentnummer eine ungültige Adresse, so schreiben Sie bitte **“ungültig/Nummer”** in das entsprechende Feld. Sollte sich aufgrund eines unzulässigen Offsets eine ungültige Adresse ergeben, so schreiben Sie bitte **“ungültig/Offset”** in das entsprechende Feld.

Der Zugriff auf die Segmenttabelle erfolgt assoziativ (associative mapping). Es werden folgende Begriffe (englische Notation) aus dem Buch zur Vorlesung verwendet:

Base	Basisadresse des Segmentes	Virt.Addr.	Virtuelle Adresse
Length	Länge des Segmentes	Seg#	Segmentnummer

Das verwendete Adressformat ist folgendes:

31	16	15	0
Seg# (16 bit)		Offset (16 bit)	

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle:

Segmenttabelle		
Seg#	Base	Length
0xFFFF	0x15FF B000	0xCCDD
0x1122	0x0808 B000	0x1001
0x1234	0x1266 0000	0x8000

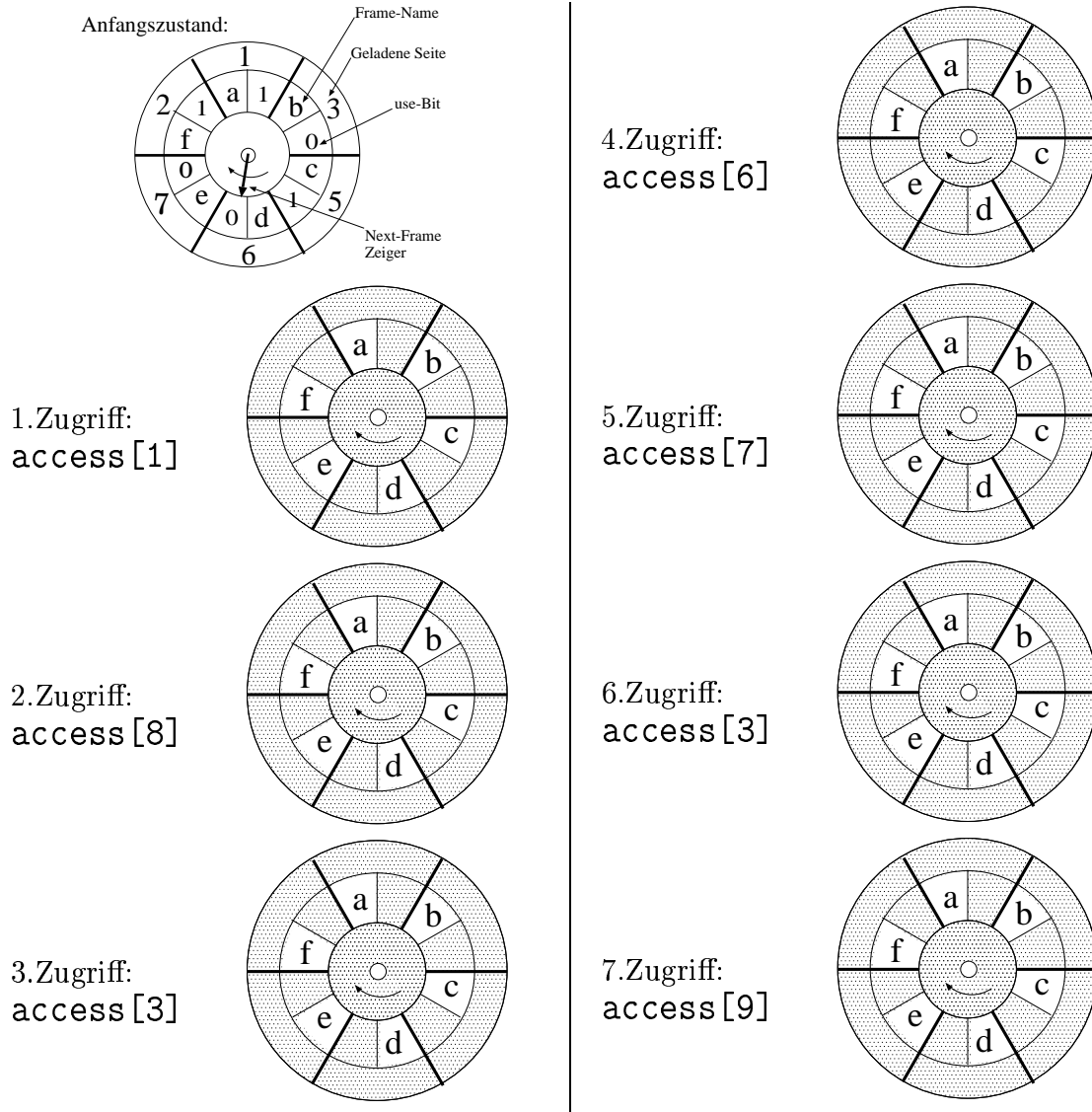
Ermitteln Sie unter Benützung obiger Segmenttabelle die physikalischen Adressen zu folgenden virtuellen Adressen:

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0xFFF1 3201	
0x1234 8000	
0x1234 8001	
0x1234 2001	
0x1121 1003	
0xFFFF 1503	

b) Replacement Policy (14)

Simulieren Sie das Verhalten einer *clock policy* zum Auslagern von Speicher-Seiten, wenn neue Seiten geladen werden müssen.

Für jede Seite im Speicher existiert ein *use*-Bit. Die einzelnen Plätze im Hauptspeicher sind in diesem Beispiel mit Buchstaben von a ... f benannt. Der *next frame*-Zeiger bewegt sich in den verwendeten Grafiken im Uhrzeigersinn weiter.



Zeichnen Sie nun in den obigen Grafiken

- die Position des *next frame*-Zeigers
- die Inhalte aller Frames (geladene Seiten)

für die Ausführung von Befehlen mit Speicherzugriffen ein. Ein Befehl `access[x]` steht für einen Speicherzugriff auf die Seite **x**. Geben Sie den gefragten Speicherzustand **nach** der Ausführung des jeweiligen Befehls an, wobei Sie für die Befehle von einer sequentiellen Reihenfolge ausgehen können.

c) (3)

Nennen Sie drei Szenarien für den fehlerfreien Fall, unter denen Programme von nicht privilegierten Benutzern die gleiche physikalische Speicheradresse benutzen:

1.

2.

3.

d) (2)

Um wieviel Prozent **vergrößert** (nicht vervielfacht!) sich der virtuelle Adressraum, wenn Sie statt 32 bit langen 34 bit lange virtuelle Adressen verwenden?

Der virtuelle Adressraum vergrößert sich um %.

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(25)

3.)(25)

4.)(20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

Bei der Studienbeihilfenbehörde: Es gibt **ein Büro mit zwei Sachbearbeitern**. Deshalb können sich im Büro maximal zwei Studenten aufhalten. Der Student kann durch die Glastüre sehen, ob es einen Sachbearbeiter gibt, der keinen Studenten betreut. Ein Sachbearbeiter wartet so lange, bis ein Student eintritt und auf sich aufmerksam macht. Ein freier Sachbearbeiter wird nun aufsehen und den Studenten bitten seine Matrikelnummer zu nennen, um den entsprechenden Akt zu holen.

Der Sachbearbeiter geht daraufhin zum Aktenschrank: Ist der Akt nicht im Schrank bedeutet dies, dass er gerade vom Kollegen bearbeitet wird. In diesem Fall wird dies dem Studenten mitgeteilt und er wird gebeten sich draußen am Ende der Reihe anzustellen (Das Programm Student terminiert und wird neu gestartet). Ist der Akt verfügbar, holt ihn der Sachbearbeiter vom Aktenschrank. **Der Student wartet bis der** seinen Fall bearbeitende **Sachbearbeiter beim Schreibtisch ankommt**. Dann bringt der Student sein Anliegen vor und nach der Klärung des Problems mit dem Sachbearbeiter verabschiedet er sich und verlässt das Büro.

a) Erstellen und initialisieren Sie alle benötigten Semaphore. (4)

Zum Anlegen eines Semaphores steht Ihnen das Statement `SemInit` zur Verfügung. Mit diesem kann man sowohl einfache Semaphore (`Seminit(SemaphorName, InitWert)`) als auch Semaphorearrays (`Seminit(SemaphorArrayName[Anzahl], InitWert1, InitWert2, ...)`) erzeugen und initialisieren. Der Aufruf `P(SemaphorName)` sperrt eine Semaphore und mit `V(SemaphorName)` gibt man es frei! (**Personalnummern der Sachbearbeiter:** $\in \{0, 1\}$.)

```
#define ANZ_SB 2
```

b) Vervollständigen Sie das Programm für einen Studenten. (14)

Es stehen Ihnen die Routinen `BueroBetreten` und `BueroVerlassen` zur Verfügung, welche keinerlei Rückgabewerte liefern, sowie die Funktion `SBAuswaehlen` welche die Personalnummer (0 oder 1) des aufsehenden Sachbearbeiters zurückgibt. Um dem Sachbearbeiter seine Matrikelnummer zu nennen und den Verbleib des Aktes zu erfahren verwende man `MatNrNennen(MatNr)`. Diese Funktion gibt einen negativen Wert zurück wenn der Akt gerade bearbeitet wird. (**C Syntax ist nicht erforderlich; es reicht Pseudocode**

Personalnummern der Sachbearbeiter: $\in \{0, 1\}$.)

```
/* Deklaration von Variablen. */
```

```
/* Warten, bis ein Platz im Buero frei ist. */
```

```
/* Buero betreten */
```

```
/* Begruessen und einen freien Sachbearbeiter *  
 * in seiner Beschaeftigung unterbrechen.      */  
Begruessen
```

```
/* Den SB auswaehlen und die Matrikelnummer    *  
 * nennen. Wenn der entsprechende Ordner        *  
 * nicht frei ist, das Buero wieder verlassen. */
```

```
if( ) {
```

```
    /* Der Akt ist nicht verfuegbar; Buero verlassen */
```



```

} else {
    /* Der Akt ist verfuegbar; warten bis ihn der *
    * Bearbeiter bei sich am Tisch hat.          */

```

```

AnliegenVortragen
AufSpaeterVertroestenLassen
MitDankVerabschieden
/* Das Buero Verlassen */

```

```

}

```

c) Vervollständigen Sie das Programm eines Sachbearbeiters (12)

Verwenden Sie die Routinen: `ZumAktenschrackGehen` und `ZumSchreibtischGehen`, sowie `AktHerausnehmen(Matrikelnummer)` und `AktZurueckstellen(Matrikelnummer)`. Weiters existiert eine Funktion mit Namen `WoIstDerAkt(Matrikelnummer)`, welche den Verbleib des Aktes signalisiert (`AKT_IST_BEIM_KOLLEGEN` bzw. `AKT_IST_VERFUEGBAR`). Um die Matrikelnummer des Studenten zu erhalten, verwende man `MatNrFragen`. Die Konstante `PERSONAL_NR` ist ebenfalls vordefiniert.

(Personalnummern der Sachbearbeiter: $\in \{0, 1\}$.)

```

/* Deklaration der Variablen. */

```

```

while((Zeit>=0900)&&(Zeit<=1200)){
    /* Warten bis ein Student kommt */

```

```

Begruessen
/* Anhand der Matrikelnummer entscheiden, wo der Akt ist. */

```

```

if(
                                     != AKT_BEIM_KOLLEGEN){

```

```
/* Der Akt ist verfuegbar! Na dann holen wir ihn. */
```

```
/* Dem Studenten signalisieren, dass man bereit zur Arbeit ist. */
```

```
DemAnliegenLauschen  
AufSpaeterVertroesten  
Verabschieden  
/* Den Akt zurueckstellen */
```

```
}  
}
```

2 Deadlock (25)

Am Institut für technische Informatik werden über den Sommer Praktika von vier Studenten (**A**ndreas, **B**eatrice, **C**hristine und **D**aniela) absolviert. Zur Abwicklung ihre Praktika brauchen die Leute folgende Ressourcen:

	A ndreas	B eatrice	C hristine	D aniela
L aserdrucker	1	1	1	0
W orkstation	1	4	1	2
K notenrechner	5	3	2	9

Abgesehen von diesen Ressourcen, die sich die Leute teilen müssen, hängen die einzelnen Arbeiten nicht voneinander ab, können also (bzw. sollen) parallel ausgeführt werden. Weiters werden Ressourcen von den Studenten erst dann wieder freigegeben, wenn ihr Praktikum beendet ist.

Insgesamt stehen am Institut 1 **L**aserdrucker, 4 **W**orkstations und 9 **K**notenrechner zur Verfügung.

Im Augenblick (Initialzustand) sieht die aktuelle Zuteilung der Ressourcen zu den Praktikanten folgendermaßen aus:

- **A**ndreas hat bereits den **L**aserdrucker und 3 **K**notenrechner belegt.
- **B**eatrice arbeitet gerade auf 2 **W**orkstations und 1 **K**notenrechner.
- **C**hristine hat zur Zeit 2 **K**notenrechner in Verwendung.
- **D**aniela benützt 1 **W**orkstation und 1 **K**notenrechner.

a) Initialzustand (5)

Ermitteln Sie *Ressource-Vektor* (RV) und *Available-Vektor* (AV) sowie *Claim-Matrix* (CM) und *Allocation-Matrix* (AM) für den obig beschriebenen Initialzustand.

$$\begin{aligned}
 RV &= \begin{pmatrix} L & W & K \\ \text{[box]} & \text{[box]} & \text{[box]} \end{pmatrix} & AV &= \begin{pmatrix} L & W & K \\ \text{[box]} & \text{[box]} & \text{[box]} \end{pmatrix} \\
 CM &= \begin{pmatrix} & L & W & K \\ A & \text{[box]} & \text{[box]} & \text{[box]} \\ B & \text{[box]} & \text{[box]} & \text{[box]} \\ C & \text{[box]} & \text{[box]} & \text{[box]} \\ D & \text{[box]} & \text{[box]} & \text{[box]} \end{pmatrix} & AM &= \begin{pmatrix} & L & W & K \\ A & \text{[box]} & \text{[box]} & \text{[box]} \\ B & \text{[box]} & \text{[box]} & \text{[box]} \\ C & \text{[box]} & \text{[box]} & \text{[box]} \\ D & \text{[box]} & \text{[box]} & \text{[box]} \end{pmatrix}
 \end{aligned}$$

b) Resource Allocation Denial (20)

Benützen Sie zur Abwicklung der weiteren Ressourcenzuteilung die Strategie des *Resource Allocation Denial*. Wenden Sie ausgehend vom Initialzustand den *Banker's Algorithmus* solange an, bis entweder alle Praktika beendet wurden, oder ein Deadlock auftritt.

Notieren Sie hierbei, welcher Praktikant gerade sein Praktikum beenden konnte (also aktiv war) bzw. wenn ein Deadlock entstanden ist, kreuzen Sie an, welche Praktikanten daran beteiligt sind. Es müssen nur die tatsächlich verwendeten Felder ausgefüllt werden.

☐ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:

☐ Andreas

☐ Beatrice

☐ Christine

☐ Daniela

☐ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} A & L & W & K \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} A & L & W & K \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \\ & & \end{pmatrix}$$

☐ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:

☐ Andreas

☐ Beatrice

☐ Christine

☐ Daniela

☐ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} A & L & W & K \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} A & L & W & K \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \\ & & \end{pmatrix}$$

☐ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:

☐ Andreas

☐ Beatrice

☐ Christine

☐ Daniela

☐ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & \text{shaded} & \text{shaded} & \text{shaded} \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & \text{shaded} & \text{shaded} & \text{shaded} \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} & L & W & K \\ & \text{shaded} & \text{shaded} & \text{shaded} \\ & & & \\ & & & \end{pmatrix}$$

☐ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:

☐ Andreas

☐ Beatrice

☐ Christine

☐ Daniela

☐ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & \text{shaded} & \text{shaded} & \text{shaded} \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & \text{shaded} & \text{shaded} & \text{shaded} \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} & L & W & K \\ & \text{shaded} & \text{shaded} & \text{shaded} \\ & & & \\ & & & \end{pmatrix}$$

c) Erklären Sie den Unterschied zwischen den Begriffen *Deadlock*, *Lifelong* und *Starvation* (5)

3 Speicherverwaltung (25)

a) Translation Lookaside Buffer (15)

Das gegebene Speicherverwaltungssystem basiert auf der Paging-Technik und verwendet einen *Translation Lookaside Buffer (TLB)* mit drei Einträgen. Der TLB wird mit der *Least Recently Used* Strategie verwaltet, d.h., wenn für eine neu einzutragende Seite kein Eintrag mehr frei ist, wird jener Eintrag ersetzt, der am längsten nicht benutzt wurde. Zu diesem Zweck enthält der TLB für jeden Eintrag ein Feld, das angibt seit wievielen TLB Zugriffen der entsprechende Eintrag nicht benutzt wurde. Bei jedem Zugriff werden die Zähler jener Einträge erhöht, die nicht gebraucht wurden, der Zähler des verwendeten oder neuen Eintrags wird auf 0 gesetzt.

Der TLB und die benötigte Page Table werden mittels **Associative Mapping** angesprochen. Die Pagegröße beträgt 4096 Bytes. Die Page-Table enthält für jeden Eintrag ein Feld *loaded*, das anzeigt, ob sich die entsprechende Seite im Hauptspeicher befindet.

Eine virtuelle Speicheradresse hat folgendes Format:

Page#(4 Bit)	Offset (12 Bit)
--------------	-----------------

Eine physikalische Speicheradresse hat folgendes Format:

Frame#(8 Bit)	Offset (12 Bit)
---------------	-----------------

Die Speicheradressen, Frame- und Pagenummern sind im Hexadezimalsystem (16er-System) angegeben.

- Auf dieser 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 der nächsten Seite.
- 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 dieser Seite ein und markieren Sie das entsprechende *Loaded*-Feld.
- 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).
- Geben Sie weiters jeweils den Inhalt des TLB **nach** dem Zugriff auf die Page an.

Ausgangssituation

Translation Lookaside Buffer

Page#	Frame#	Last Use
2	0x47	0
4	0x00	1

Page Table

Page#	Loaded	Frame#
0	<input checked="" type="checkbox"/>	0x2F
1	<input type="checkbox"/>	
2	<input checked="" type="checkbox"/>	0x47
3	<input type="checkbox"/>	
4	<input checked="" type="checkbox"/>	0x00

Virtuelle Adresse

0x0313

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	Last Use

Virtuelle Adresse

0x042F

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	Last Use

Virtuelle Adresse

0x1141

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	Last Use

Virtuelle Adresse

0x4711

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	Last Use

b) TLB-Zeitverhalten(4)

Man nehme an, ein Zugriff auf den TLB oder ein Zugriff auf den Cache-Speicher dauert 15ns, ein Zugriff auf den Hauptspeicher braucht 60ns und das Laden einer Seite vom Platte dauert 10ms. Gehen Sie davon aus, dass die Pagetable *zur Gänze im Hauptspeicher* gehalten wird und die durch den TLB referenzierten Seiten sowie der TLB vom Cache-Speicher unterstützt werden. Rechnen Sie den Quotienten minimale Zugriffszeit : maximale Zugriffszeit auf zwei signifikante Stellen genau aus.

Quotient $\frac{t_{min}}{t_{max}} =$

c) Verständnisfragen (6)

- 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
- Welche dieser Replacement-Policies ist am einfachsten zu implementieren?
 - ☐ Least recently used
 - ☐ Clock algorithmus
 - ☐ First in - first out
- Die Clock Replacement Strategie liefert weniger Page Faults als die Least Recently Used Replacement Strategy.
 - ☐ richtig
 - ☐ falsch
- Beim *Fixed Partitioning* können sich Partitionen im Hauptspeicher überlappen.
 - ☐ richtig
 - ☐ falsch
- Große Seitengrößen bei reinem Paging führen zu großen Seitentabellen (page tables).
 - ☐ richtig
 - ☐ falsch
- 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)

4 Security (20)

a) (3)

Welche drei Sicherheitsanforderungen gibt es?

b) (3)

Welche Sicherheitsanforderungen sind in den folgenden Beispielen verletzt? (bitte kurze **Begründung** angeben!)

- Ein Druckprogramm erlaubt fälschlicherweise den Ausdruck einer Datei mit geheimen Daten.

- Ein Benutzer ändert unerlaubterweise das Passwort eines autorisierten Benutzers.

c) (1)

Ein frustrierter Programmierer, der sich aufgrund seiner bevorstehenden Kündigung an seiner Firma rächen will, fügt in das Buchhaltungsprogramm eine Routine ein, die einen Tag nach seiner Kündigung alle Daten löscht. Geben Sie den englischen Fachbegriff für diese Routine an!

d) (1)

Ein unautorisierter Benutzer kann alle Files in einem Computersystem lesen. Wie bezeichnet man diesen *threat* und gegen welche Sicherheitsanforderung verstößt er?

e) (4)

Erklären Sie die *Grundzüge* des Public-Key Verschlüsselungsverfahrens, sowie die *Vor-* und *Nachteile* gegenüber konventionellen Verschlüsselungsverfahren.

Grundzüge:

Vorteile:

Nachteile:

f) (3)

Was für potenzielle Schwachstellen(mind. 3) weist das Passwort *password* auf?

g) (3)

Welche Regeln(mind. 3) sollten bei der Wahl eines "sicheren" Passwortes angewandt werden?

KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (30)

2.) (25)

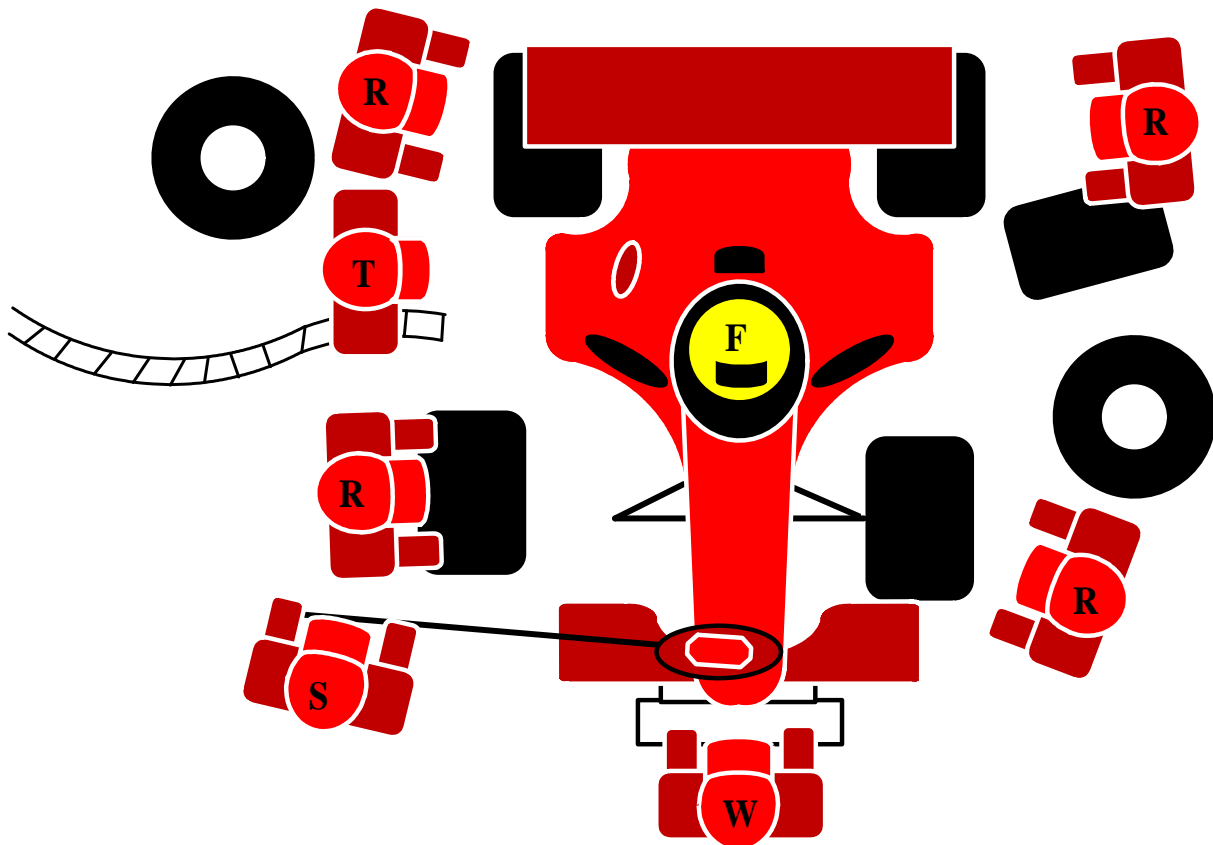
3.) (25)

4.) (20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Ein **Boxenstopp** in der Formel 1 läuft folgendermaßen ab: Sobald der Wagen eingefahren ist, beginnt der Tankwart (T) mit dem auffüllen des Tanks, während der Mann mit dem Wagenheber (W) den Wagen anhebt. Ist der Wagen in der Luft, beginnen die vier Reifenwechsler (R) mit dem Austausch der Reifen. Sind alle vier Reifen gewechselt, lässt der Mann mit dem Wagenheber das Fahrzeug wieder zu Boden und geht mit dem Wagenheber

zur Seite. Wenn der Wagenheber beiseite geräumt *und* der Tankvorgang beendet ist, gibt der Signalmann dem Fahrer das Signal zum Wegstarten.

Bedenken Sie bitte, dass es beim **Boxenstopp** auf optimale Synchronisation zwischen den Boxenmitarbeitern untereinander und zwischen Box und Fahrer ankommt, um keine wertvolle Zeit zu verlieren. Verwenden Sie außerdem nur unbedingt notwendige Synchronisationskonstrukte um dieses Ziel zu erreichen.

Zu verwendende Funktionen:

`initE(Name)` Legt einen Eventcounter mit dem angegebenen Namen *Name* an und initialisiert ihn mit der Zahl 0. Danach können die Funktionen **Advance(*Name*)** und **Wait(*Name*,*Wert*)** auf den Eventcounter angewendet werden.

`initS(Semaphor,init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`reFuel()` Initiiert einen Tankvorgang. Diese Funktion wird vom Tankwart ausgeführt und blockiert, bis der Tankvorgang beendet ist und der Tankwart aus dem Gefahrenbereich verschwunden ist.

`changeTire()` Wechselt den Reifen. Diese Funktion wird von den Reifenwechslern ausgeführt und blockiert, bis der Vorgang beendet ist und der Reifenwechsler aus dem Gefahrenbereich verschwunden ist.

`liftCar()` Hebt den Wagen an. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

`lowerCar()` Lässt den Wagen wieder sinken. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

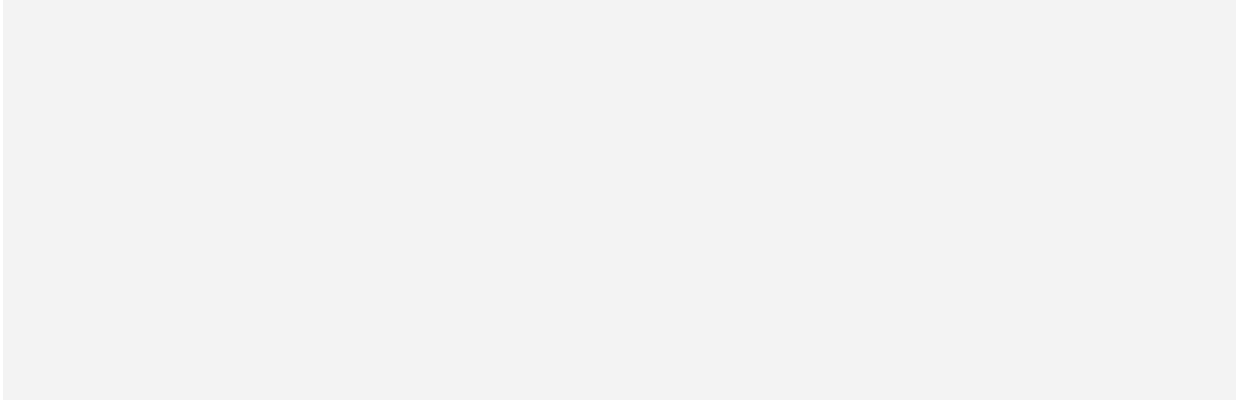
`removeJack()` Entfernt den Wagenheber aus dem Gefahrenbereich. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist und der bedienende Mann aus dem Gefahrenbereich verschwunden ist.

Synchronisieren Sie den Arbeitsablauf der sieben Mitarbeiter des Boxenteams für *einen einzigen* Boxenstopp mittels **Eventcounter**. Sie können davon ausgehen, dass am Anfang der Wagen zum Stillstand gekommen ist, der Wagenheber bereits unter dem Wagen positioniert ist, der Wagen aber noch nicht angehoben ist und weder Reifenwechsler noch Tankwart mit ihrer Arbeit begonnen haben.

Synchronisieren Sie weiters die Kommunikation zwischen Fahrer und Signalgeber mittels **Semphor(e)**.

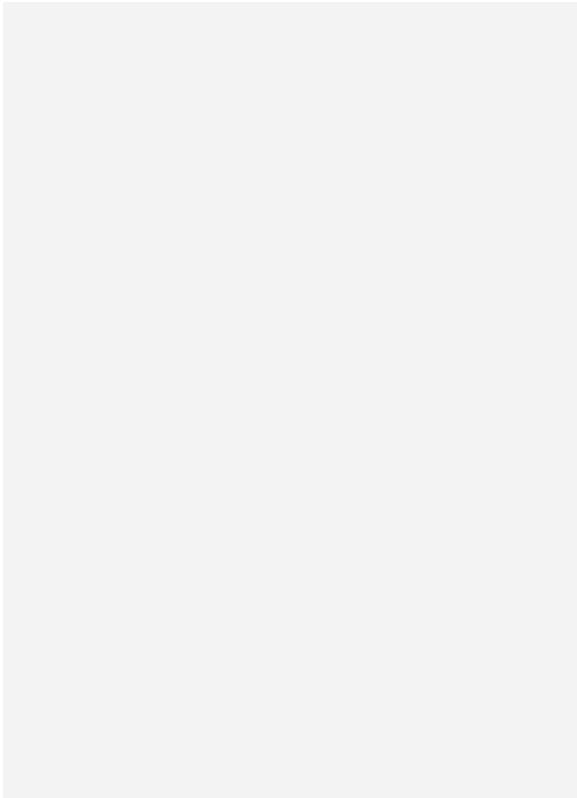
a) Initialisierungen (5)

Nehmen Sie hier die notwendigen Initialisierungen, welche beim Einfahren in die Box ausgeführt werden, vor:



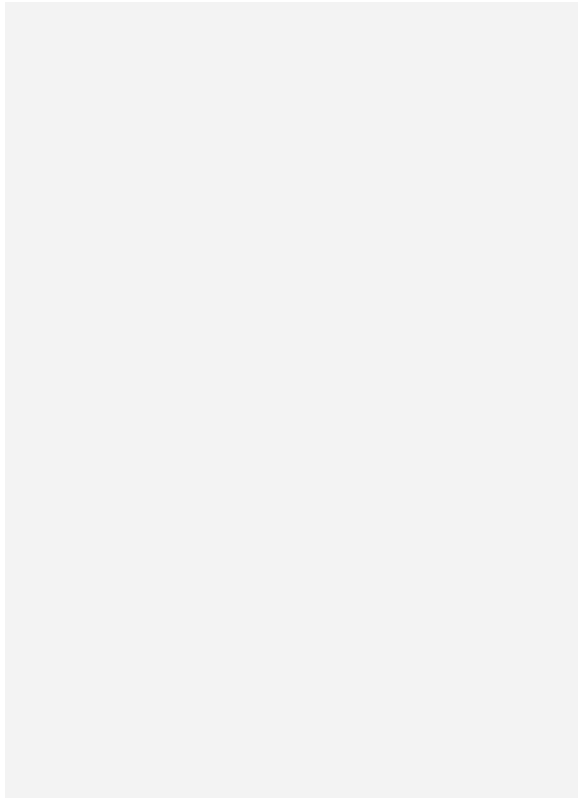
b) (5)

Entwerfen Sie hier das Programm für einen beliebigen Reifenwechsler (R):



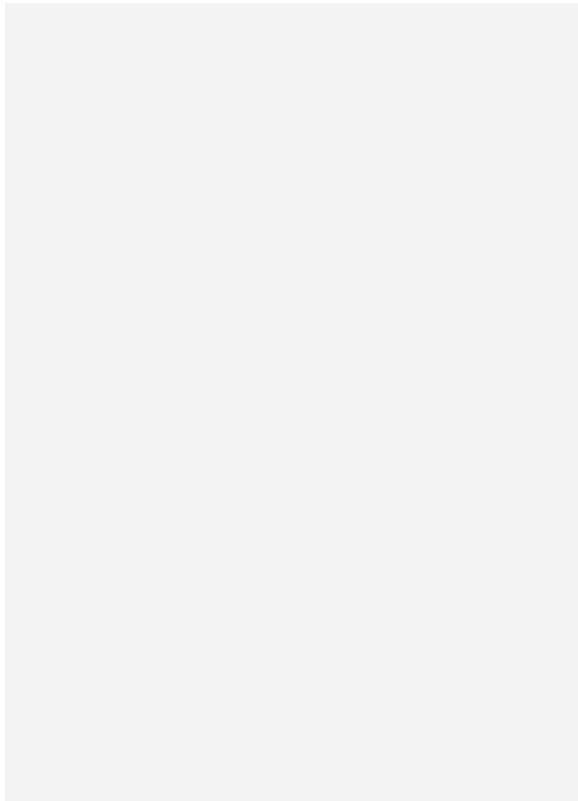
c) (5)

Entwerfen Sie hier das Programm für den Mann der den Wagenheber bedient (W):



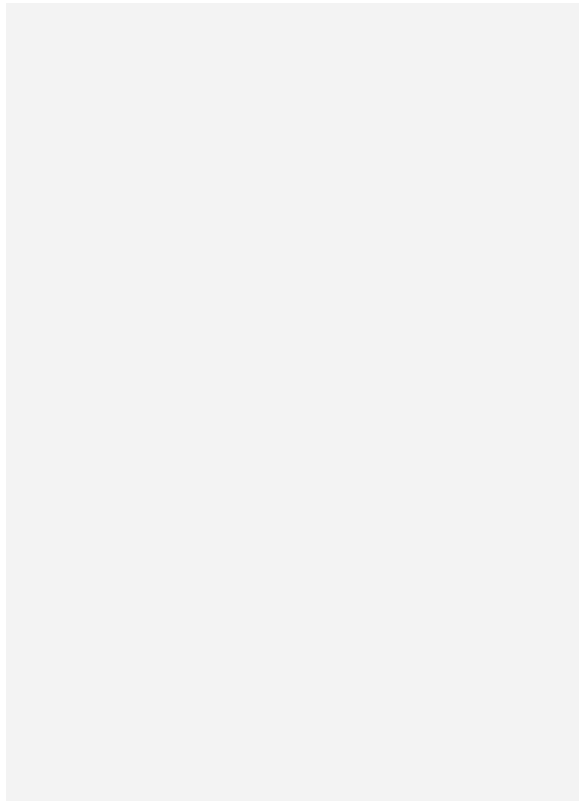
d) (5)

Entwerfen Sie hier das Programm für den
Signalgeber (S):



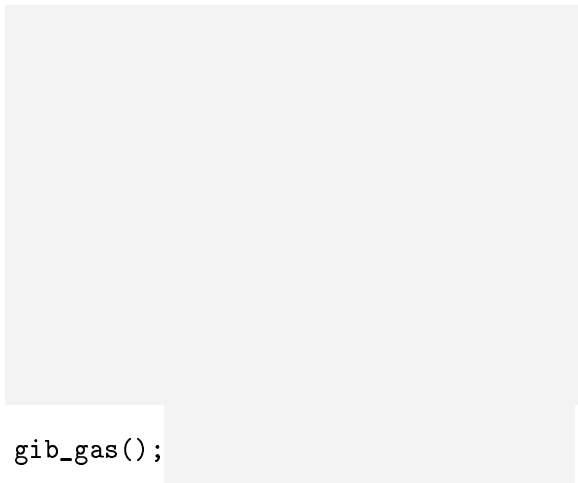
e) (5)

Entwerfen Sie hier das Programm für den
Tankwart (T):



f) (5)

Entwerfen Sie hier das Programm für den
Fahrer (F):



`gib_gas();`

2 Speicherverwaltung (25)

a) Segmentierung (13)

In einem Computersystem mit virtuellem Speicher gibt es drei Tasks, A, B, und C, welche die folgenden Ressourcen benötigen:

Task	Code (bytes)	Data (bytes)
A	1250	798
B	3423	3745
C	1024	0

Geben Sie den Verschnitt an Speicher (= Speicher, der reserviert, aber nicht tatsächlich benötigt wird) in Bezug zum reservierten Speicher für jedes der Programme für die zwei Fälle in Punkt 1) und 2) in Prozent an.

Nehmen Sie an, dass Code und Daten getrennt gespeichert werden. Geben Sie unbedingt das Ergebnis in Form einer *Dezimalzahl* mit einer Stelle hinter dem Komma (z.B. 13.1 %) an.

1) Verwendung von Paging, Seitengröße 2 KB (=2048 Bytes) (6)

Task A: Anzahl der Seiten = \Rightarrow Verschnitt = . %

Task B: Anzahl der Seiten = \Rightarrow Verschnitt = . %

Task C: Anzahl der Seiten = \Rightarrow Verschnitt = . %

2) Segmentierung des Speichers (3)

Verschnitt von Task A = . %

Verschnitt von Task B = . %

Verschnitt von Task C = . %

3) Erklärung (4)

Erklären Sie einerseits für Paging, andererseits für Speichersegmentierung, durch welchen Effekt es dazu kommen kann, dass der vorhandene physikalische Speicher nicht optimal genutzt wird. Bitte Namen der Effekte und Erklärung angeben.

Paging:

Segmentierung:

b) Kombination aus Segmentierung und Paging (12)

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 in der Folge betrachtete Speicherverwaltungssystem verwendet zur (physikalischen und virtuellen) Adressierung 20-bit Adressen. Für das Paging sind alle Seitenrahmen 256 Bytes (hexadezimal 0x00 100) groß. Das verwendete Adressformat ist folgendes:

Seg# (8 bit)	Page# (4 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
0x02	0x04 21F	0x4
0xFC	0x7D 000	0x2
0x3F	0xC1 0E0	0x3
0x09	0x20 221	0xF

Seitentabelle	
Address	Frame#
...	...
0x04 21F	0x451
0x04 220	0x023
0x04 221	0x845
0x04 222	0x734
0x04 223	0x475
...	...
0x20 221	0x311
0x20 222	0xF00
...	...
0x20 22E	0x000
0x20 22F	0xDDD

Fortsetzung Seitentabelle	
Address	Frame#
0x20 230	0x666
0x20 231	0x765
...	...
0x7D 000	0x471
0x7D 001	0x871
0x7D 002	0x111
...	...
0xC1 0E0	0x394
0xC1 0E1	0x588
0xC1 0E2	0x127
0xC1 0E3	0x600
...	...

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):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0x0123F	
0x3F311	
0x09EFF	
0x090D7	
0xFC000	
0x0238A	

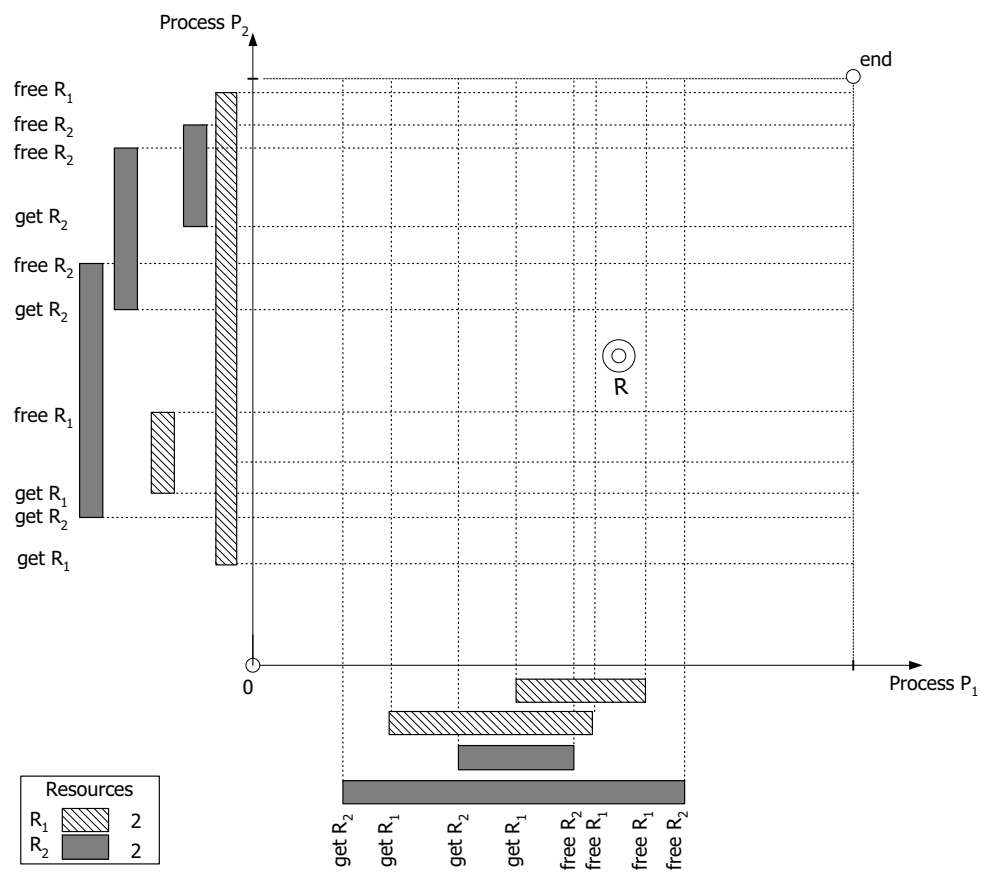
3 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt seiner Abarbeitung er jeweils die einzelnen Ressourcen benötigt. 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 0 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.
4. In der Grafik ist ein Punkt R mit einem Kreis markiert. Kann dieser Punkt bei der Abarbeitung der Prozesse erreicht werden? Begründen Sie Ihre Antwort!

Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 1 und 2 *deutlich unterscheidbar* sind.



4 Konzepte von Betriebssystemen (20)

a) (4)

Geben Sie zwei Abstraktionen an, die ein Betriebssystem zur Verfügung stellt. Beschreiben Sie für jede dieser Abstraktionen, welche Aufgaben sie dem Benutzer abnimmt.

b) (3)

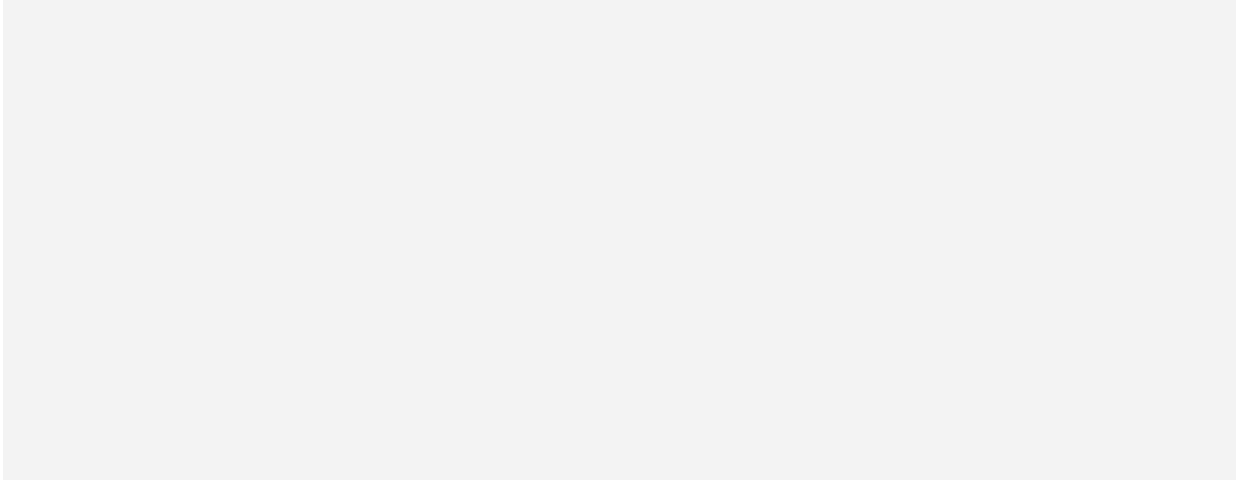
Wodurch gewährleistet ein Betriebssystem die Portabilität von Programmen?

c) (5)

Geben Sie die drei Fälle an, in denen die Kontrolle über einen Rechner von einem Benutzerprogramm an das Betriebssystem wechselt.

d) (8)

Was ist ein Mode Switch? Was ist ein Process Switch? Was passiert bei jedem dieser Switches? Wie stehen Mode Switch und Process Switch in Beziehung?



KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (30)

2.) (25)

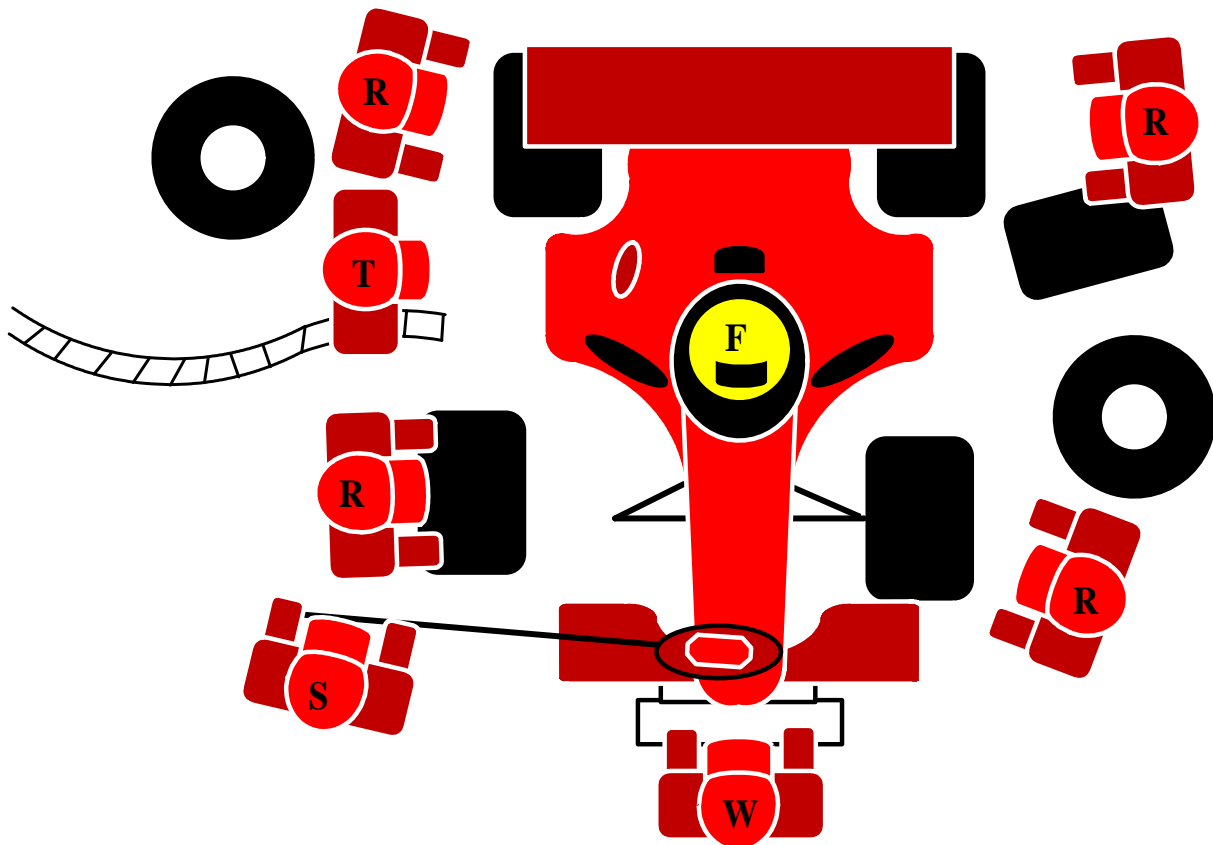
3.) (25)

4.) (20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Ein **Boxenstopp** in der Formel 1 läuft folgendermaßen ab: Sobald der Wagen eingefahren ist, beginnt der Tankwart (T) mit dem auffüllen des Tanks, während der Mann mit dem Wagenheber (W) den Wagen anhebt. Ist der Wagen in der Luft, beginnen die vier Reifenwechsler (R) mit dem Austausch der Reifen. Sind alle vier Reifen gewechselt, lässt der Mann mit dem Wagenheber das Fahrzeug wieder zu Boden und geht mit dem Wagenheber

zur Seite. Wenn der Wagenheber beiseite geräumt *und* der Tankvorgang beendet ist, gibt der Signalmann dem Fahrer das Signal zum Wegstarten.

Bedenken Sie bitte, dass es beim **Boxenstopp** auf optimale Synchronisation zwischen den Boxenmitarbeitern untereinander und zwischen Box und Fahrer ankommt, um keine wertvolle Zeit zu verlieren. Verwenden Sie außerdem nur unbedingt notwendige Synchronisationskonstrukte um dieses Ziel zu erreichen.

Zu verwendende Funktionen:

`initE(Name)` Legt einen Eventcounter mit dem angegebenen Namen *Name* an und initialisiert ihn mit der Zahl 0. Danach können die Funktionen **Advance(*Name*)** und **Wait(*Name*,*Wert*)** auf den Eventcounter angewendet werden.

`initS(Semaphor,init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`reFuel()` Initiiert einen Tankvorgang. Diese Funktion wird vom Tankwart ausgeführt und blockiert, bis der Tankvorgang beendet ist und der Tankwart aus dem Gefahrenbereich verschwunden ist.

`changeTire()` Wechselt den Reifen. Diese Funktion wird von den Reifenwechslern ausgeführt und blockiert, bis der Vorgang beendet ist und der Reifenwechsler aus dem Gefahrenbereich verschwunden ist.

`liftCar()` Hebt den Wagen an. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

`lowerCar()` Lässt den Wagen wieder sinken. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

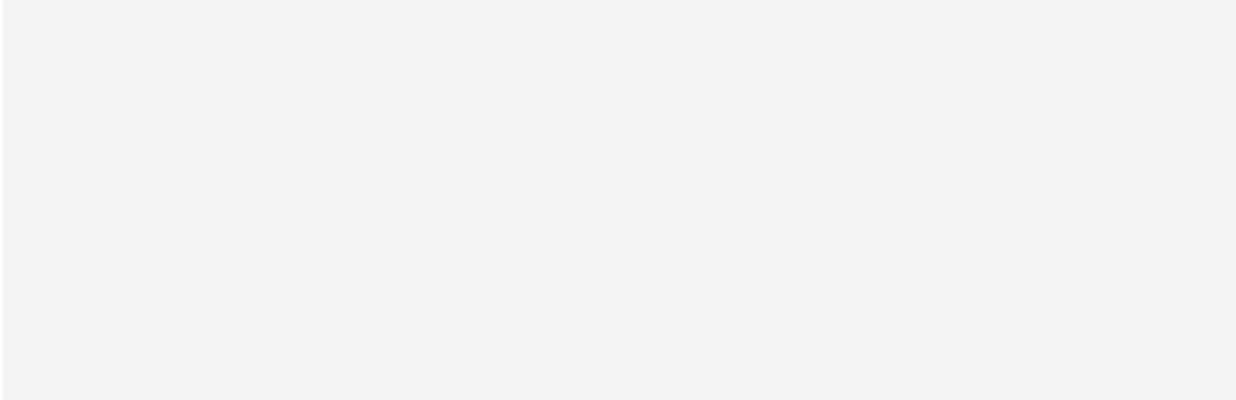
`removeJack()` Entfernt den Wagenheber aus dem Gefahrenbereich. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist und der bedienende Mann aus dem Gefahrenbereich verschwunden ist.

Synchronisieren Sie den Arbeitsablauf der sieben Mitarbeiter des Boxenteams für *einen einzigen* Boxenstopp mittels **Eventcounter**. Sie können davon ausgehen, dass am Anfang der Wagen zum Stillstand gekommen ist, der Wagenheber bereits unter dem Wagen positioniert ist, der Wagen aber noch nicht angehoben ist und weder Reifenwechsler noch Tankwart mit ihrer Arbeit begonnen haben.

Synchronisieren Sie weiters die Kommunikation zwischen Fahrer und Signalgeber mittels **Semphor(e)**.

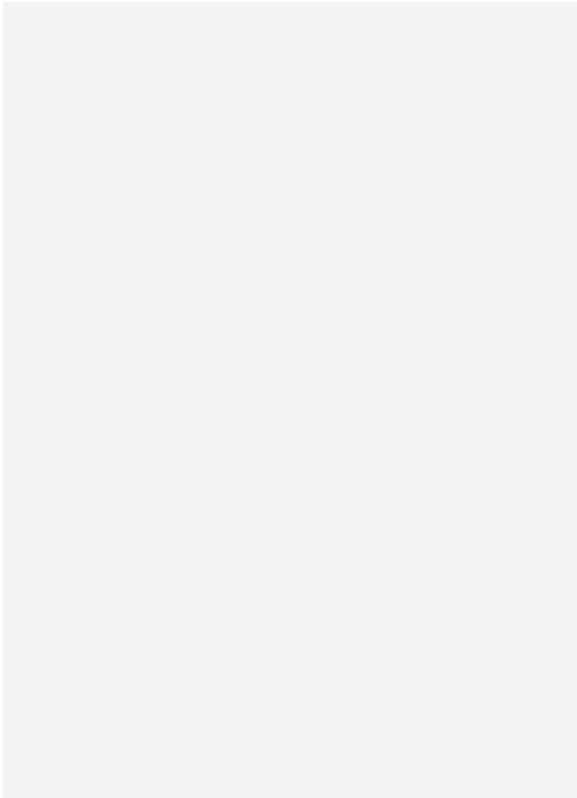
a) Initialisierungen (5)

Nehmen Sie hier die notwendigen Initialisierungen, welche beim Einfahren in die Box ausgeführt werden, vor:



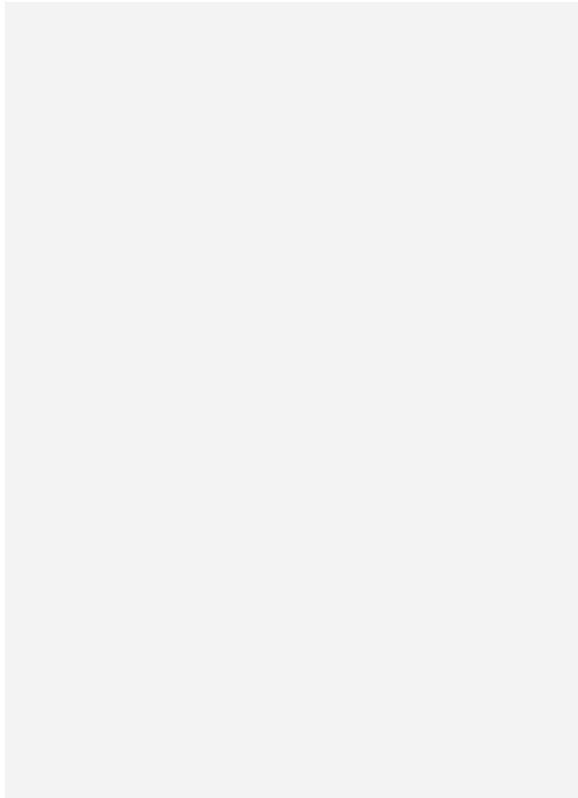
b) (5)

Entwerfen Sie hier das Programm für einen beliebigen Reifenwechsler (R):



c) (5)

Entwerfen Sie hier das Programm für den Mann der den Wagenheber bedient (W):



d) (5)

Entwerfen Sie hier das Programm für den
Signalgeber (S):

e) (5)

Entwerfen Sie hier das Programm für den
Tankwart (T):

f) (5)

Entwerfen Sie hier das Programm für den
Fahrer (F):

```
gib_gas();
```

2 Speicherverwaltung (25)

a) Segmentierung (13)

In einem Computersystem mit virtuellem Speicher gibt es drei Tasks, A, B, und C, welche die folgenden Ressourcen benötigen:

Task	Code (bytes)	Data (bytes)
A	1250	798
B	3423	3745
C	1024	0

Geben Sie den Verschnitt an Speicher (= Speicher, der reserviert, aber nicht tatsächlich benötigt wird) in Bezug zum reservierten Speicher für jedes der Programme für die zwei Fälle in Punkt 1) und 2) in Prozent an.

Nehmen Sie an, dass Code und Daten getrennt gespeichert werden. Geben Sie unbedingt das Ergebnis in Form einer *Dezimalzahl* mit einer Stelle hinter dem Komma (z.B. 13.1 %) an.

1) Verwendung von Paging, Seitengröße 2 KB (=2048 Bytes) (6)

Task A: Anzahl der Seiten = \Rightarrow Verschnitt = . %

Task B: Anzahl der Seiten = \Rightarrow Verschnitt = . %

Task C: Anzahl der Seiten = \Rightarrow Verschnitt = . %

2) Segmentierung des Speichers (3)

Verschnitt von Task A = . %

Verschnitt von Task B = . %

Verschnitt von Task C = . %

3) Erklärung (4)

Erklären Sie einerseits für Paging, andererseits für Speichersegmentierung, durch welchen Effekt es dazu kommen kann, dass der vorhandene physikalische Speicher nicht optimal genutzt wird. Bitte Namen der Effekte und Erklärung angeben.

Paging:

Segmentierung:

b) Kombination aus Segmentierung und Paging (12)

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 in der Folge betrachtete Speicherverwaltungssystem verwendet zur (physikalischen und virtuellen) Adressierung 20-bit Adressen. Für das Paging sind alle Seitenrahmen 256 Bytes (hexadezimal 0x00 100) groß. Das verwendete Adressformat ist folgendes:

Seg# (8 bit)	Page# (4 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
0x02	0x04 21F	0x4
0xFC	0x7D 000	0x2
0x3F	0xC1 0E0	0x3
0x09	0x20 221	0xF

Seitentabelle	
Address	Frame#
...	...
0x04 21F	0x451
0x04 220	0x023
0x04 221	0x845
0x04 222	0x734
0x04 223	0x475
...	...
0x20 221	0x311
0x20 222	0xF00
...	...
0x20 22E	0x000
0x20 22F	0xDDD

Fortsetzung Seitentabelle	
Address	Frame#
0x20 230	0x666
0x20 231	0x765
...	...
0x7D 000	0x471
0x7D 001	0x871
0x7D 002	0x111
...	...
0xC1 0E0	0x394
0xC1 0E1	0x588
0xC1 0E2	0x127
0xC1 0E3	0x600
...	...

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):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0x0123F	
0x3F311	
0x09EFF	
0x090D7	
0xFC000	
0x0238A	

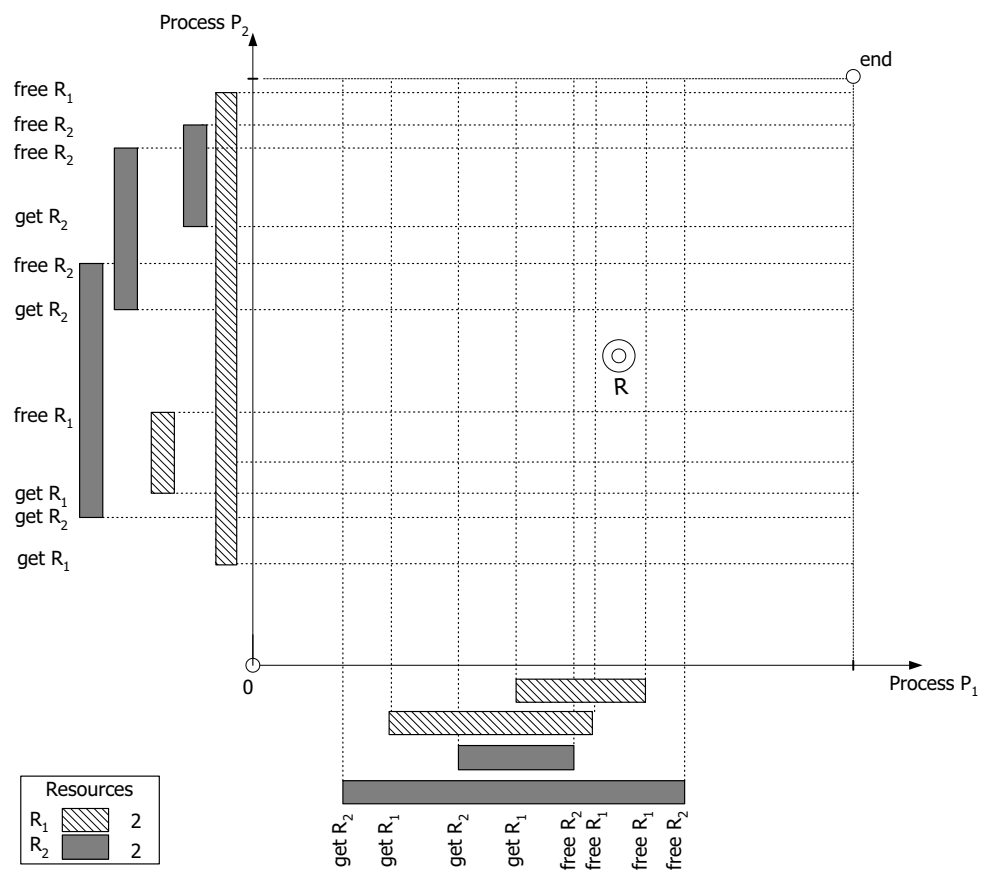
3 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt seiner Abarbeitung er jeweils die einzelnen Ressourcen benötigt. 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 θ 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.
4. In der Grafik ist ein Punkt R mit einem Kreis markiert. Kann dieser Punkt bei der Abarbeitung der Prozesse erreicht werden? Begründen Sie Ihre Antwort!

Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 1 und 2 *deutlich unterscheidbar* sind.



4 Konzepte von Betriebssystemen (20)

a) (4)

Geben Sie zwei Abstraktionen an, die ein Betriebssystem zur Verfügung stellt. Beschreiben Sie für jede dieser Abstraktionen, welche Aufgaben sie dem Benutzer abnimmt.

b) (3)

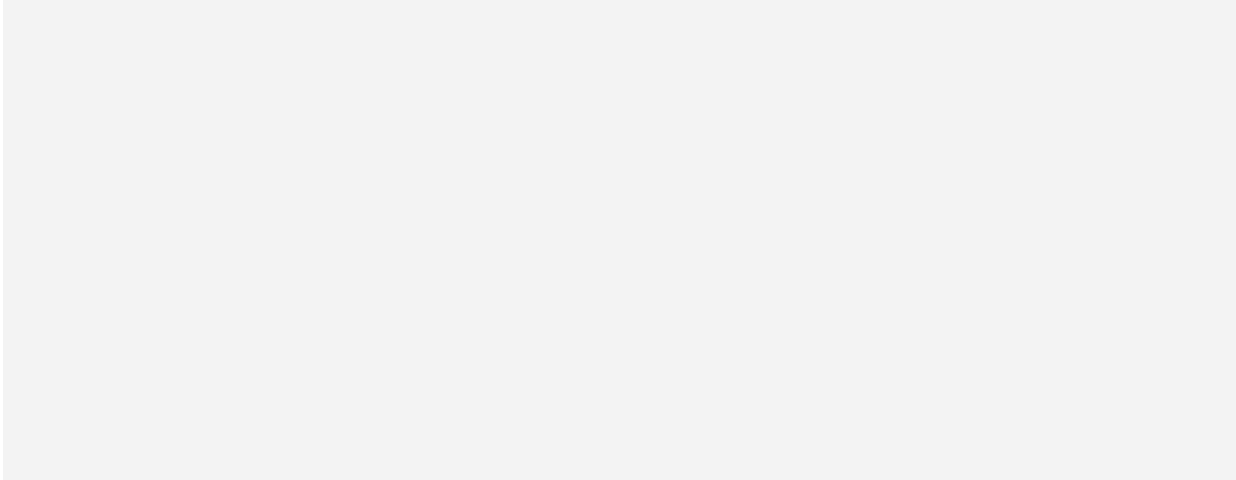
Wodurch gewährleistet ein Betriebssystem die Portabilität von Programmen?

c) (5)

Geben Sie die drei Fälle an, in denen die Kontrolle über einen Rechner von einem Benutzerprogramm an das Betriebssystem wechselt.

d) (8)

Was ist ein Mode Switch? Was ist ein Process Switch? Was passiert bei jedem dieser Switches? Wie stehen Mode Switch und Process Switch in Beziehung?



KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (35)

2.) (20)

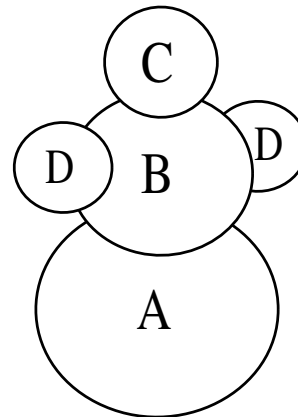
3.) (20)

4.) (25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (35)



Die Informatik-Studenten feiern das kommende Semesterende, indem sie gemeinsam Schneemänner bauen. Die einzelnen Teile eines Schneemannes werden gemäß obigen Bauplanes mit *A* bis *D* benannt.

Sie wollen Algorithmen zum Bauen des Schneemannes entwickeln:

- Es sind die Funktionen `setze_A()`, `setze_B()`, `setze_C()` und `setze_D()` zu entwickeln, die jeweils das bezeichnete Teil am nächst möglichen Schneemann anbringen. Jeder Student kann *mehrmals* beliebig eine dieser Funktionen verwenden, auch *gleichzeitig* mit anderen Studenten. Die jeweilige Funktion soll solange blockieren, bis ein Schneemann frei ist, an dem das Teil montiert werden kann.

Die erfordernten vorhandenen Teile für das Anbringen eines neuen Teiles sind aus folgender Tabelle auszulesen:

Anzubringendes Teil	Benötigte(s) Teil(e)
A	-
B	A
C	A B
D	A B

- Es steht ihnen dazu Funktion **setze(x)** zum Plazieren eines Teiles zur Verfügung, wobei **x** ein Teil aus **{A,B,C,D}** bezeichnet. Die Funktion fügt den Teil automatisch am ersten Schneemann an, der diesen noch nicht vollständig enthält.

a) Die “parallele Klasse” (18)

Die Studenten der “parallelen Klasse” haben folgende Regel festgelegt:

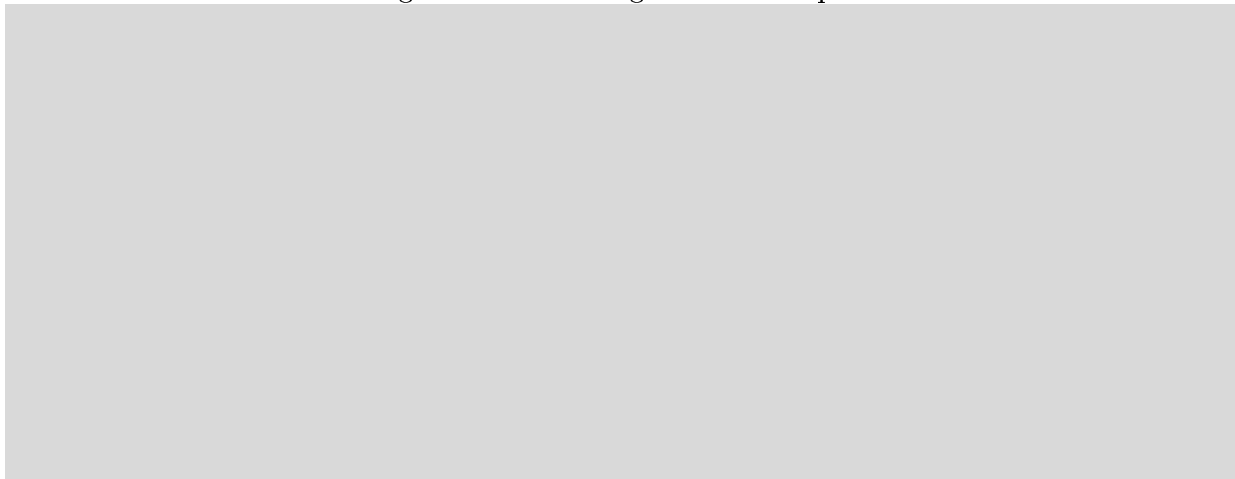
- Es darf an maximal **MAX** Schneemännern gleichzeitig gebaut werden. Ein weiterer Schneemann darf erst angefangen werden, wenn ein anderer fertig geworden ist.

Die Synchronisation ist mit (*möglichst wenig!*) Semaphoren durchzuführen. 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.

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

a1) Initialisierungen (3)

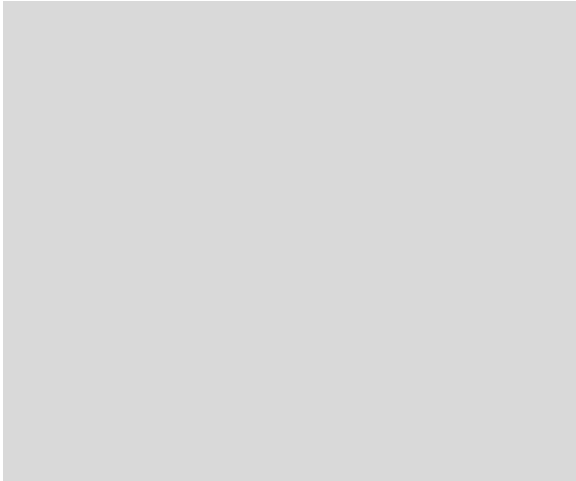
Geben Sie hier die notwendigen Initialisierungen von Semaphoren an.



a2) (5)

Programm zum Platzieren einer Kugel *A*:
setze_A()

BEGIN

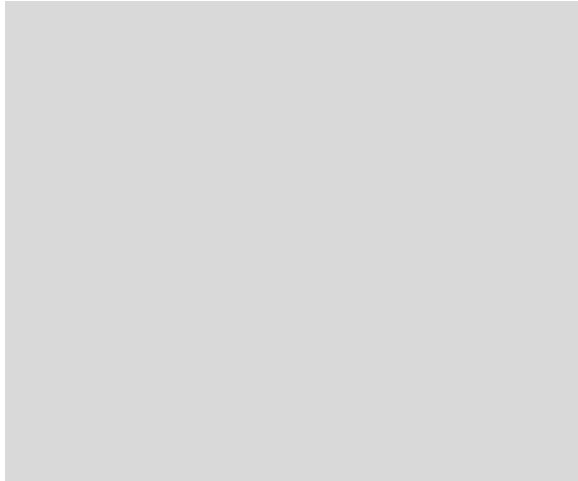


END

a3) (4)

Programm zum Platzieren der Kugel *B*:
setze_B()

BEGIN

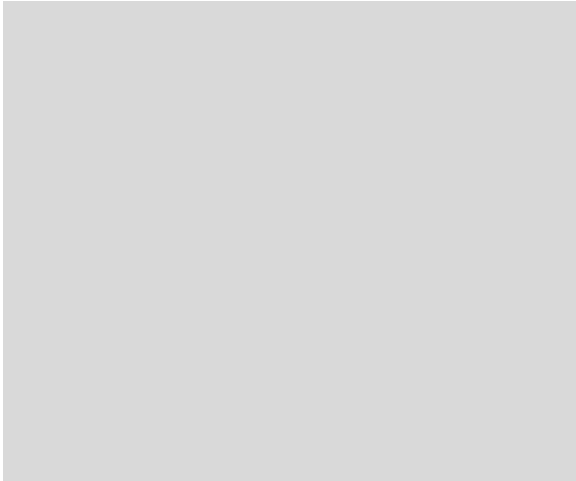


END

a4) (3)

Program zum Platzieren der Kugel *C*:
setze_C()

BEGIN

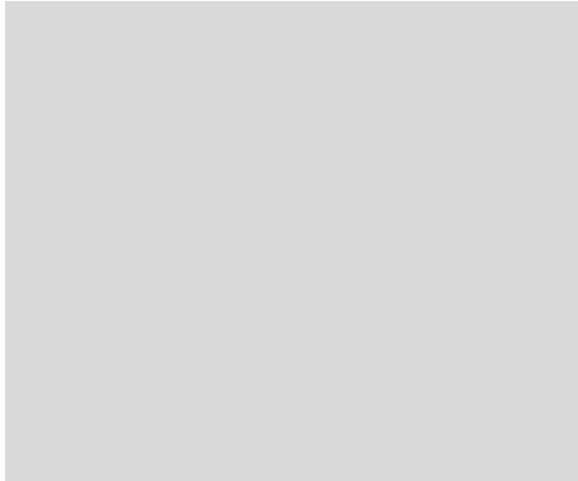


END

a5) (3)

Programm zum Platzieren eines Armes *D*:
setze_D()

BEGIN



END

b) Die “sequentielle Klasse” (17)

Die Studenten der “sequentuellen Klasse” haben folgende Regel festgelegt:

- Es darf an maximal **einem** Schneemann gleichzeitig gebaut werden. Ein weiterer Schneemann darf erst angefangen werden, wenn der vorherige fertig geworden ist.

Die Synchronisation ist mit (*möglichst wenig!*) Sequencern und Eventcountern durchzuführen. Auf Sequencer kann die Funktion **sticket(s)** angewendet werden. Auf Eventcounter können die Funktionen **eawait(e,v)** und **eadvance(e)** angewendet werden.

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

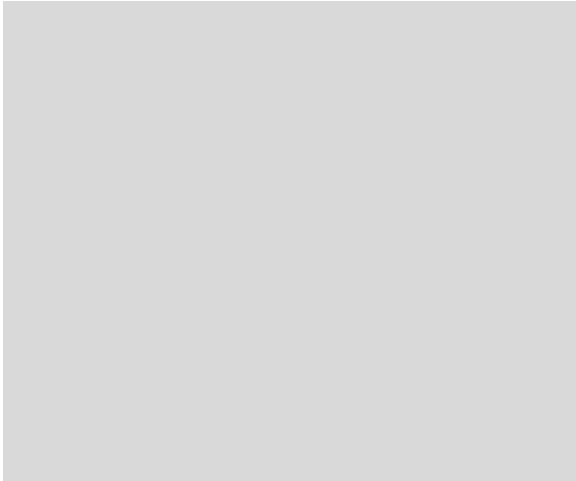
b1) Deklarationen (3)

Geben Sie hier die Deklarationen von Sequenzern/Eventcountern an. Verwenden Sie hierzu die Notation **Sequencer X;** bzw. **Eventcounter Y;**.

b2) (4)

Programm zum Platzieren einer Kugel *A*:
`setze_A()`

BEGIN

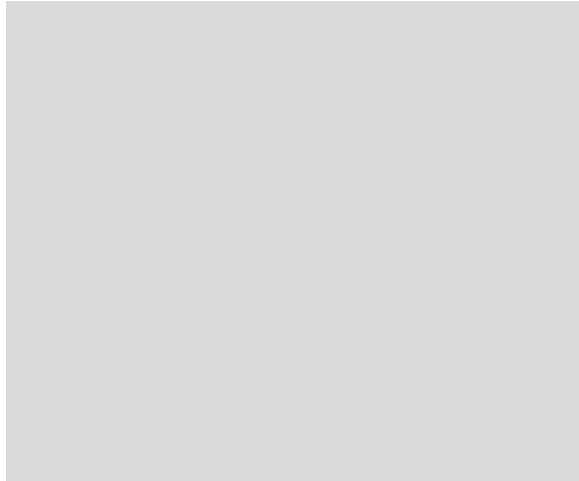


END

b3) (3)

Programm zum Platzieren der Kugel *B*:
`setze_B()`

BEGIN

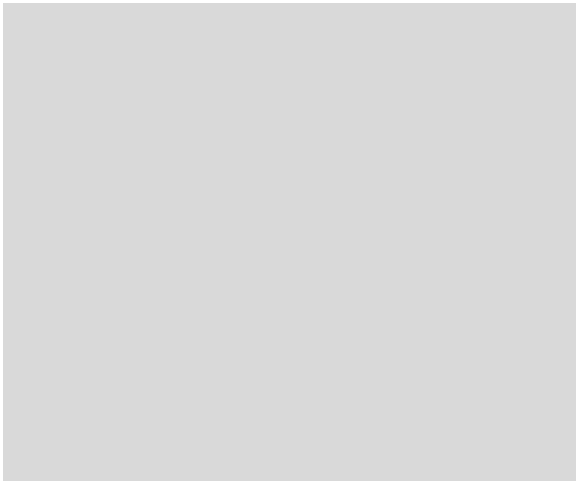


END

b4) (4)

Program zum Platzieren der Kugel *C*:
`setze_C()`

BEGIN

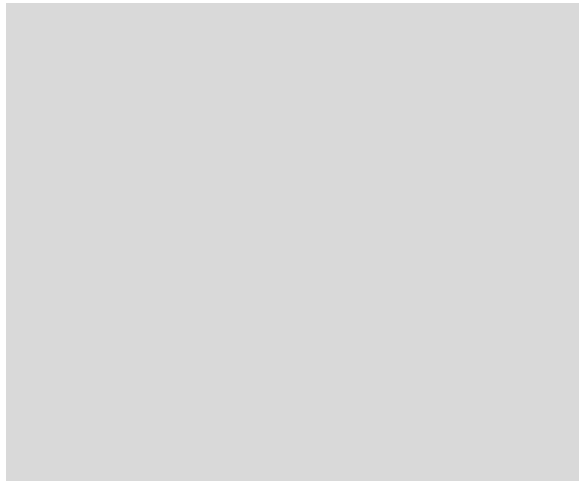


END

b5) (3)

Programm zum Platzieren eines Armes *D*:
`setze_D()`

BEGIN



END

2 Speicherverwaltung (20)

a) Kombination aus Segmentierung und Paging (13)

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	0xFFFFFFFF	0x001
0xABC	0xBC050010	0x002
0xB00	0xBC050012	0x0B5
0xEB0	0xA0010010	0x004
0xFFF	0x00000000	0x005

Seitentabelle	
Address	Frame#
0x00000000	0x123456
0x00000001	0x152451
0x00000002	0x152452
...	...
0xA0010010	0x761010
0xA0010011	0x761011
0xA0010012	0x761012
0xA0010013	0x761013
0xA0010014	0x761014
...	...
0xBC050010	0x666012
0xBC050011	0x666011
0xBC050012	0x666010
0xBC050013	0x66600F
...	...
0xBC0500C6	0x666000
0xBC0500C7	0x666001
0xBC0500C8	0x666002
...	...
0xFFFFFFFF	0x000001

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):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)		
0x000001FF			
0xABC00100			
0xFFFF002AB			
0x000000EA			
0xEB000000			
0xB000B4FF			
0xABC001FF			
0xB000B500			
0xEB0002AA			

b) Verständnisfragen (7)

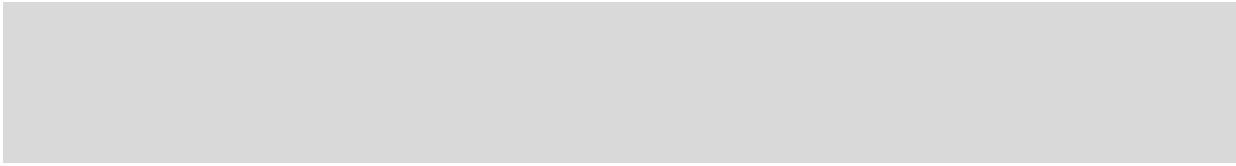
Kreuzen Sie bitte die richtigen Antworten an! Achtung! Falsche Antworten werden negativ gewertet!

- Eine Austauschstrategie, die auf alle Seiten des Hauptspeichers angewandt wird, nennt man
 - ☐ lokale Ersetzungsstrategie
 - ☐ globale Ersetzungsstrategie
- 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
- Beim *Fixed Partitioning* können sich keine Partitionen im Hauptspeicher überlappen.
 - ☐ richtig
 - ☐ falsch
- Zu welchen Effekten(mehrere möglich) kann es bei Segmentierung kommen?
 - ☐ Unterschiedliche Segmentlängen
 - ☐ Internal Fragmentation
 - ☐ External Fragmentation
- Bei einem Buddy System sind die anforderbaren Speicherblockgrößen immer größer als die größte bisher angeforderte Speicherblockgröße.
 - ☐ richtig
 - ☐ falsch

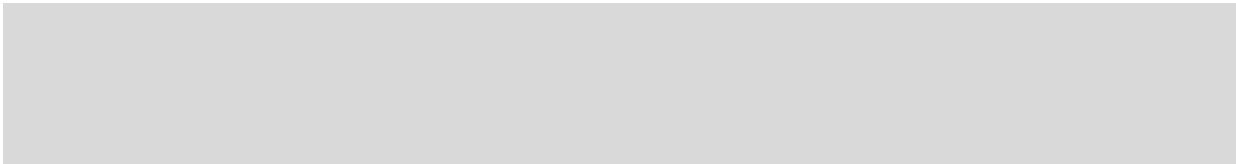
3 Security (20)

a) (4)

Was versteht man unter dem Schlagwort *Least Priviledge*?




Geben Sie dazu ein Beispiel an:



b) (6)

Nennen Sie bitte drei Arten von Security Threats und geben Sie jeweils ein Beispiel dazu an:



c) (10)

Ist es möglich mit einem symmetrischen Verschlüsselungsalgorithmus eine fälschungssichere elektronische Unterschrift zu realisieren?

☐ Ja

☐ Nein

Wenn ja, geben Sie bitte an wie so ein Algorithmus funktioniert. Im Fall, dass Sie die vorherige Frage mit nein beantwortet haben, begründen Sie bitte, warum das nicht möglich ist.

A large, empty gray rectangular box intended for the user to provide a detailed answer to the question about symmetric encryption and electronic signatures.

Welchen Vorteil hat die symmetrische gegenüber der asymmetrischen Verschlüsselung?

A smaller, empty gray rectangular box intended for the user to provide a detailed answer to the question about the advantages of symmetric encryption over asymmetric encryption.

4 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3 . Die Anzahl der vorhandenen Ressourcen im System können Sie aus dem Ressourcen-Vektor R ablesen. Aus der Claim-Matrix C ist die maximal notwendige Anzahl von Ressource pro Prozess ersichtlich. 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.

$$\mathbf{R} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

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 0 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 die Punkte erreicht werden können, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	<input type="radio"/>	<input type="radio"/>
S_2	<input type="radio"/>	<input type="radio"/>
S_3	<input type="radio"/>	<input type="radio"/>
S_4	<input type="radio"/>	<input type="radio"/>
S_5	<input type="radio"/>	<input type="radio"/>
S_6	<input type="radio"/>	<input type="radio"/>

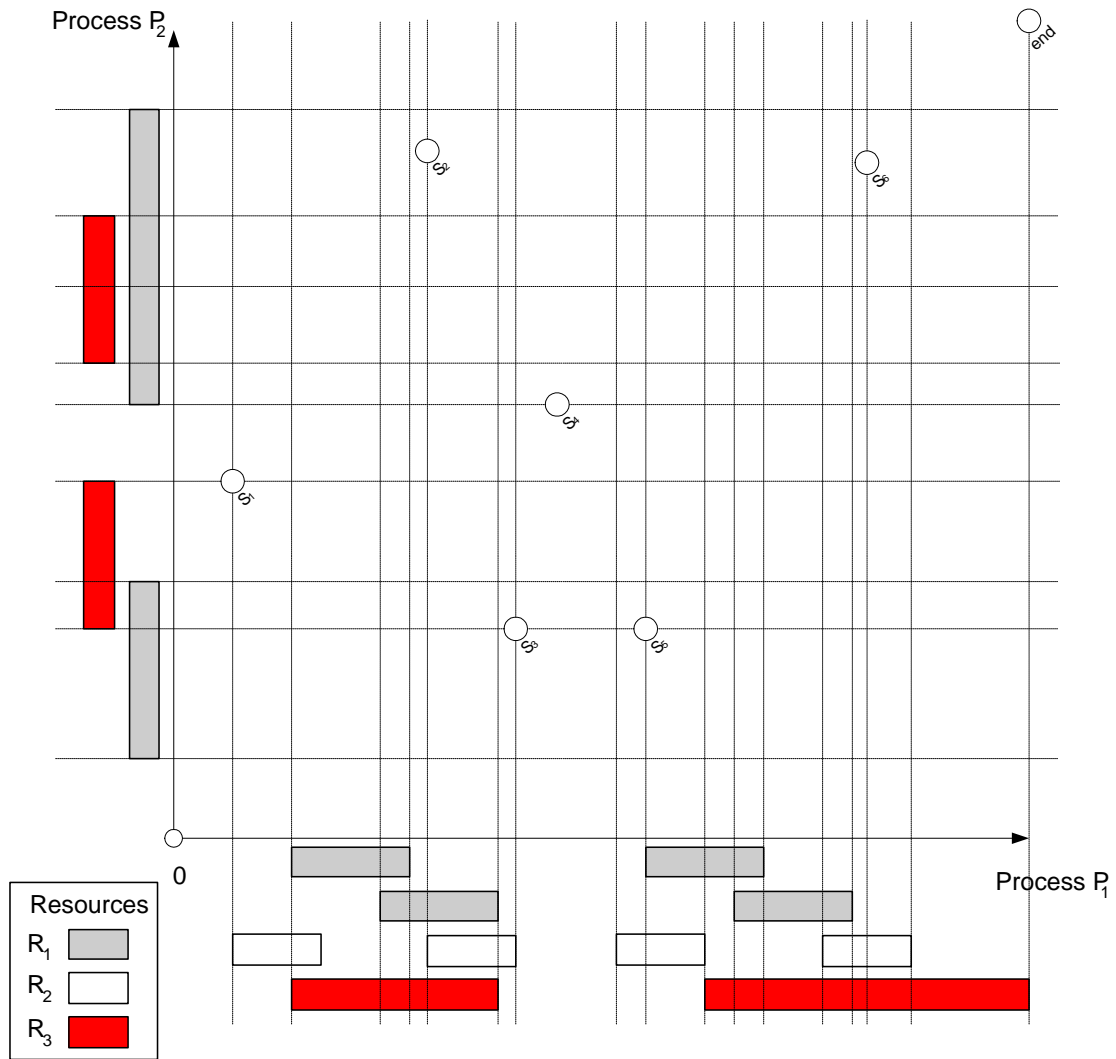


Abbildung 1: Abarbeitungs Diagramm

KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (30)

2.) (25)

3.) (25)

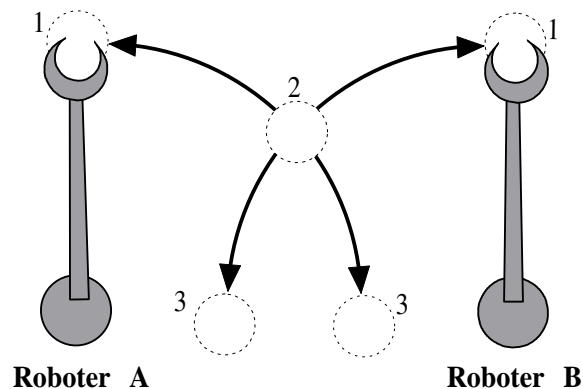
4.) (20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

In einer Fabrik sind zwei Roboter so angeordnet, dass sich ihre Arbeitsbereiche teilweise überlappen. Jeder Roboter kann eine von drei Arbeitspositionen anfahren, um Objekte aufzunehmen oder abzulegen.



Implementieren Sie die Teile eines Programms `Transport(von, nach)`, welches einen Roboter ein Objekt von Position *von* nach Position *nach* legen lässt.

Ihnen stehen folgende Funktionen zur Verfügung:

`Grab()` lässt den Roboter ein Objekt von jener Position aufnehmen, auf welcher der Roboterarm gerade steht.

`Release()` lässt den Roboter ein Objekt an jener Position ablegen, auf welcher der Roboterarm gerade steht.

`GoTo(Position)` Führt den Roboterarm an die angegebene Position.

Beachten Sie folgende Randbedingungen:

- Die Arme der Roboter A und B dürfen sich niemals überkreuzen. Weiters dürfen sich beide Arme nie gleichzeitig auf Position 2 oder 3 befinden. Position 2 kann aber z. B. angefahren werden, wenn der andere Greifer auf Position 3 steht.
- Gehen Sie davon aus, dass bei Aufruf der Funktion `Transport()` ein Objekt auf *von* vorhanden ist, die Robotergreifer auf Position 1 stehen und kein Objekt gegriffen wurde.
- Es können beliebig viele Objekte an einem Ort abgelegt werden, d.h es muss *nicht* überprüft werden, ob eine Position frei ist.
- Stellen Sie den Greifer nach erfolgter Aktion in die Position 1 zurück.
- Gehen Sie davon aus, dass je eine Instanz von `Transport()` auf jedem der Roboter gleichzeitig laufen kann.
- Stellen Sie sicher, dass während des Be- oder Entladens der andere Roboter die nicht blockierten Positionen anfahren kann, sofern dadurch kein Deadlock entstehen kann.
- Verwenden Sie ausschließlich Semaphore zur Synchronisation. Es sind möglichst wenig Synchronisationskonstrukte zu verwenden, die Verwendung von globalen Variablen ist verboten.

Verständnisfrage

Zu welchen Problemen kann es kommen, wenn Roboter A nicht jedesmal auf Position 1 zurückgestellt wird, sondern auf der letzten Entladeposition stehenbleiben würde? Kreuzen Sie die richtige(n) Antwort(en) an und begründen Sie Ihre Antwort!

☐ Deadlock

☐ Starvation

☐ Lifelock

Implementierung

Legen Sie die notwendigen Synchronisationskonstrukte mit der Funktion `InitSem(Semaphorname, Initialwert)` an und initialisieren Sie sie entsprechend der Situation auf dem Bild.

Transport()

Das Programm `Transport(von, nach)` verwendet die Unterprogramme `Transport1To(nach)`, `Transport2To(nach)` und `Transport3To(nach)`. Implementieren Sie diese, so dass `Transport()` wie angegeben funktioniert:

```
Transport(von,nach)
```

```
{  
    switch(von) {  
        case 1:  
            Transport1To(nach);  
            break;  
        case 2:  
            Transport2To(nach);  
            break;  
        case 3:  
            Transport3To(nach);  
            break;  
    }  
}
```

```
Transport1To(nach)
```

```
{
```

```
}
```

Transport2To(nach)

{

Transport3To(nach)

{

}

4

}

2 Memory Management - Replacement Policies (25)

Gegeben ist eine virtuelle Speicheradressierung (Paging mit Frame Allocation Size von 3) und eine Folge von Page-Zugriffen mit den folgenden Page-Nummern: 3,2,5,4,3,5,1,5,3,4,5. Es sind die Inhalte der Page-Frames nach jedem Page-Zugriff zu bestimmen. Zusätzlich sind eventuelle Page-Faults zu markieren.

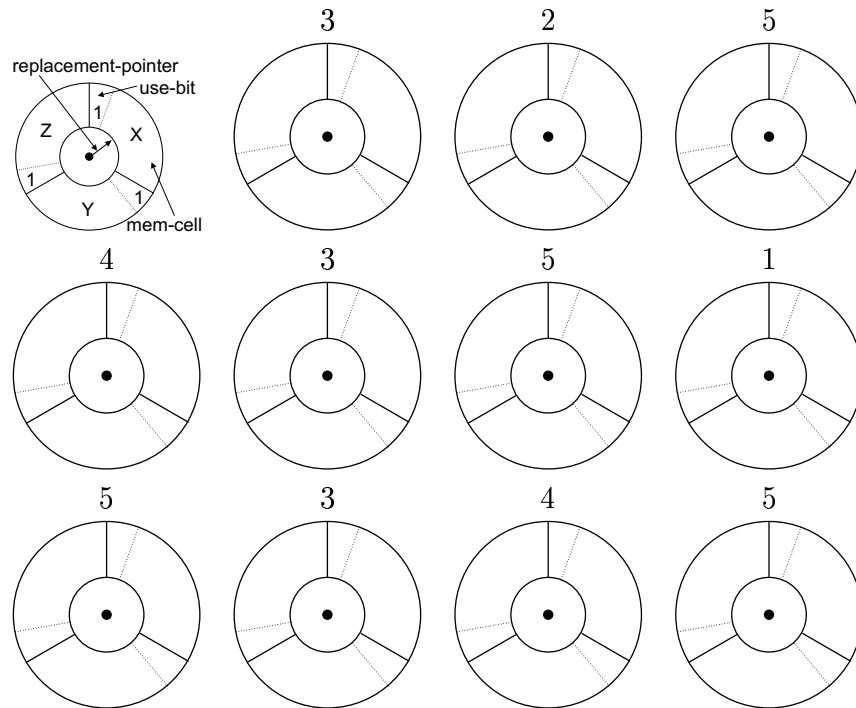
Tragen Sie in folgender Tabelle die Inhalte der Page-Frames ein, entsprechend der folgenden Replacement Policies:

- *Optimal* (OPT)
- *Least Recently Used* (LRU)
- *First-in,First-out* (FIFO)
- *Simple Clock Policy* (CLOCK)

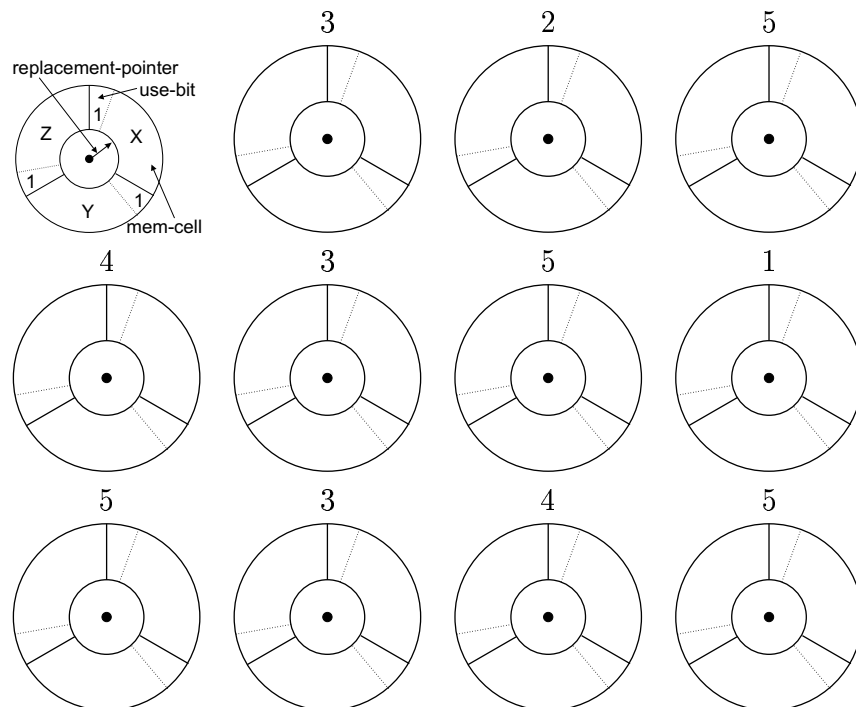
Weiters sind nachfolgende Page Faults mittels \otimes zu markieren (für die CLOCK-Policy sind sämtliche Schritte in den Kreisdiagrammen auf der nächsten Seite einzuzeichnen - inklusive Replacement-Pointer und Usage-Bit). Sollte es gültige Alternativlösungen geben, so ist eine Variante davon auszuwählen.

	3	2	5	4	3	5	1	5	3	4	5
OPT											
	-										
	-	-									
	-	-	-	○	○	○	○	○	○	○	○
LRU											
	-										
	-	-									
	-	-	-	○	○	○	○	○	○	○	○
FIFO											
	-										
	-	-									
	-	-	-	○	○	○	○	○	○	○	○
CLOCK											
	-										
	-	-									
	-	-	-	○	○	○	○	○	○	○	○

Templates für die CLOCK-Policy (die Felder sind in der Reihenfolge *rechts - unten - links* aufzufüllen). Die Pages sind mit den Werten $\{X, Y, Z\}$ vorbelegt, welche zu den Werten $\{1, 2, 3, 4, 5\}$ paarweise disjunkt sind.



Reserve-Vorlage (die ungültige Version ist **durchzustreichen**):









Vergleich der Page-Faults

Übertragen Sie aus dem vorhergehendem Beispiel die Anzahl der Page-Faults.

Policy	Anzahl der Page-Faults
OPT	
LRU	
FIFO	
CLOCK	

Diese vier Replacement-Policies lassen sich für den allgemeinen Fall bezüglich der Effizienz mit den Bewertungen *besser/gleich/schlechter* vergleichen.

- Tragen Sie nun entsprechend den Page-Faults aus obiger Tabelle die Kriterien “<” (*besser*), “=” (*gleich*) oder “>” (*schlechter*) in die linken Teilspalten in der folgenden Tabelle ein. Die Bewertungen sind dabei jeweils vom Zeilennamen ausgehend zum Spaltennamen zu lesen (Beispiel: sollte OPT schlechter als LRU sein, tragen Sie dafür “>” im entsprechenden Feld ein).
- Markieren Sie mittels \otimes in folgender Tabelle jene Paare von Policies, für welche die aus den **konkreten Zugriffen des obigen Beispiels** gewonnenen Aussage *besser/gleich/schlechter* **auch** für den **allgemeinen Fall** mit beliebigen Page-Zugriffen gilt.

	LRU		FIFO		CLOCK	
OPT		○		○		○
LRU	-			○		○
FIFO	-		-			○

3 Filemanagement (25)

Files bestehen aus mehreren Gruppen von sequentiell aufeinanderfolgenden Blöcken. In der Tabelle 1 finden Sie die Namen von fünf Files und die Längen von bis zu vier Gruppen von sequentiellen Blöcken pro File.

In Abbildung 1 ist ein Speichermedium schematisch als Matrix von Blöcken dargestellt. Die einzelnen Blöcke sind nummeriert von 0, links oben, bis 99, rechts unten.

Tragen Sie in Abbildung 1 die Files aus der Tabelle 1 ein. Die Files sollen nach der Strategie *Indexed allocation mit variabler Länge* im Speicher abgelegt werden. Markieren Sie dazu alle benötigten Blöcke (für FileX tragen Sie bitte X in das Rechteck ein ($X=\{A...E\}$), und zeichnen Sie für jeden Beginn einer Blocksequenz einen Pfeil vom dazugehörigen Index-Block zum Anfangsblock der Blocksequenz. Tragen Sie die Adressen jedes Index-Blocks in die Tabelle 1 ein.

Die Files werden nacheinander abgearbeitet, wobei bei jedem Schritt EINE Gruppe von Blöcken der in Tabelle 1 angegebenen Länge an das entsprechende File angehängt wird. Bei Länge 0 wird nichts angehängt.

Es soll eine möglichst große Sequenz von freien Blöcken am Ende des Speichers übrigbleiben.

Beschreiben Sie den Algorithmus den Sie zum Lösen dieses Beispiels angewandt haben in Pseudo-Code.

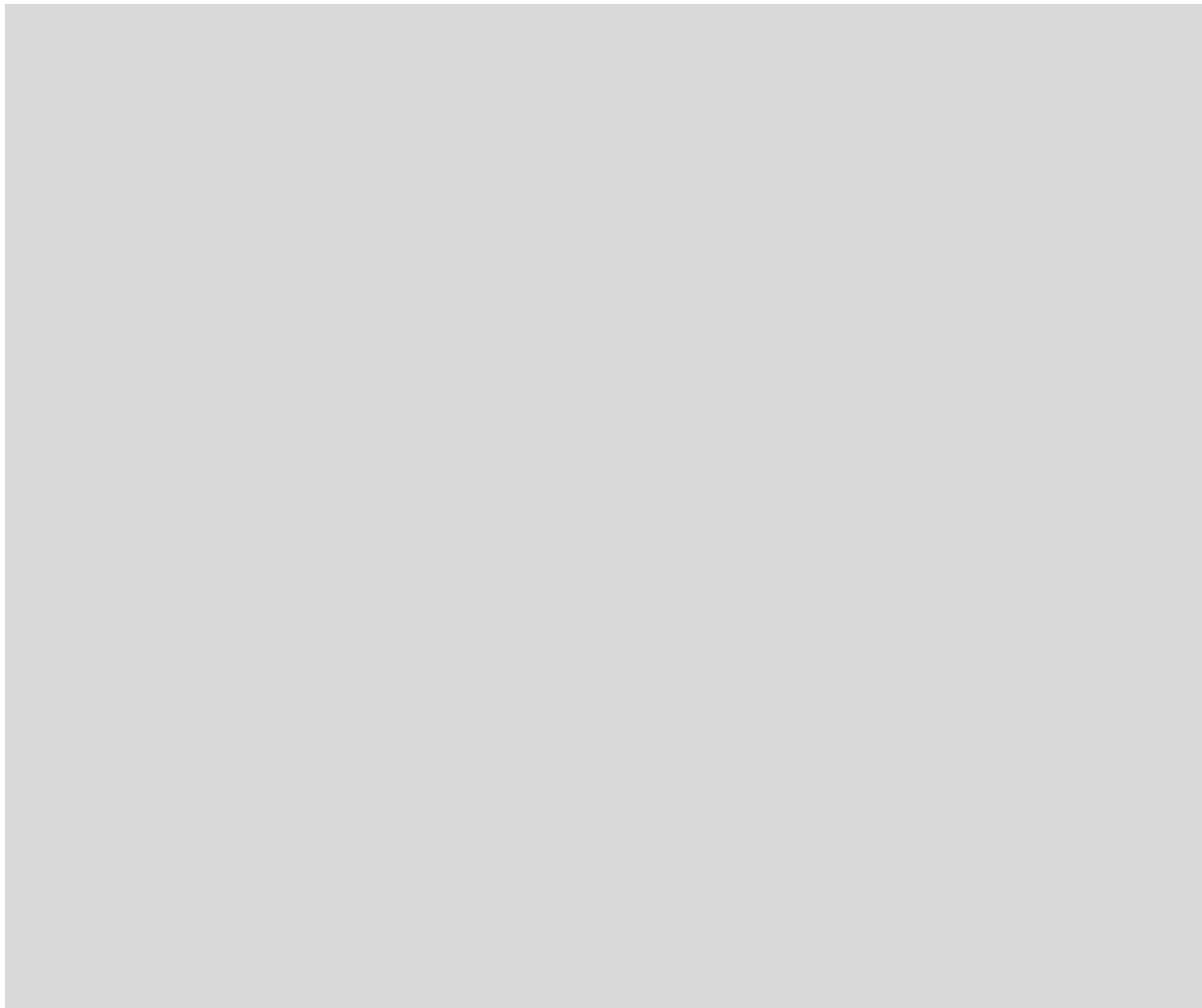
	0	1	2	3	4	5	6	7	8	9
0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
40	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
60	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
70	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
80	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
90	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 1: Speicher

File Name	Index Block	Durchlauf 1 Length 1	Durchlauf 2 Length 2	Durchlauf 3 Length 3	Durchlauf 4 Length 4
FileA		10	2	10	0
FileB		2	5	2	5
FileC		4	0	0	0
FileD		8	4	0	0
FileE		7	3	0	0

Tabelle 1: File Allocation Table

Pseudo-Code des Algorithmus:



4 Scheduling (20)

Real-Time Scheduling

Klassifizieren Sie die Scheduling-Strategien Rate Monotonic Scheduling (RMS) und Earliest Deadline First (EDF) anhand der folgenden Eigenschaften:

- "preemptive" versus "non-preemptive" Scheduling-Verfahren
- zentrale (centralized) versus verteilte (distributed) Scheduling-Entscheidung (Entscheidung der Abarbeitung von Prozessen)
- dynamisches (= Task Set zur Laufzeit veränderbar) versus statisches (= fixiertes Task Set; Abarbeitung der Tasks wird off-line entschieden) Scheduling
- dynamische versus statische Zuweisung von Prioritäten

Bitte klassifizieren Sie EDF und RMS durch Eintragen von Kreuzen (Kreuze haben die Bedeutung "diese Eigenschaft trifft zu"):

Eigenschaft	RMS	EDF
preemptive		
nonpremtive		
zentrale Entscheidung		
verteilte Entscheidung		
dynamisches Scheduling		
statisches Scheduling		
dynamische Prioritätenzuweisung		
statische Prioritätenzuweisung		

Kreuzen Sie an, ob die folgenden Aussagen korrekt oder inkorrekt sind:

Scheduling nach dem Round-Roubin Prinzip wird bei Verwendung sehr großer Zeitquanten zu Scheduling nach dem First-In-First-Out Prinzip.

- ☐ korrekt ☐ inkorrekt

Eine Scheduling-Strategie für eine CPU ist "preemptive", wenn das Betriebssystem einen Prozess *nicht* von der CPU suspendieren kann.

- ☐ korrekt ☐ inkorrekt

Echtzeitsysteme benutzen generell preemptive CPU Scheduling Verfahren.

- ☐ korrekt ☐ inkorrekt

Die Antwortenzeiten von Systemen, die "preemptive" Scheduling-Strategien benutzen, sind besser voraussagbar (=Angaben sind zuverlässiger), als die Antwortzeiten von Systemen, die kooperatives (= "non-preemptive") Scheduling benutzen.

☐ korrekt ☐ inkorrekt

Generell kann man sagen, dass rechenintensive Prozesse bei Scheduling nach dem Round-Robin-Prinzip durchschnittlich schneller abgearbeitet werden als I/O-intensive Prozesse.

☐ korrekt ☐ inkorrekt

Scheduling-Strategien, die Prioritäten statisch zuweisen, sind leichter zu implementieren als Scheduling-Strategien, die Prioritäten dynamisch zuweisen.

☐ korrekt ☐ inkorrekt

Scheduling-Strategien, die Prioritäten statisch zuweisen, verlangen mehr Laufzeit-Overhead als Scheduling-Strategien, die Prioritäten dynamisch zuweisen.

☐ korrekt ☐ inkorrekt

Scheduling nach dem Multi-level Feedback Queue Prinzip bevorzugt Prozesse mit langen Ausführungszeiten.

☐ korrekt ☐ inkorrekt

Scheduling nach dem Multi-level Feedback Queue Prinzip bevorzugt I/O-intensive Prozesse, um gute I/O-Geräte-Auslastung zu erreichen.

☐ korrekt ☐ inkorrekt

Deadline Scheduling verlangt einen intensiven Ressourcen-Einsatz; deshalb ist der Overhead im Vergleich zu anderen Scheduling Strategien groß.

☐ korrekt ☐ inkorrekt

Scheduling nach dem Round-Robin-Prinzip unter Verwendung von langen Zeitquanten bevorzugt I/O-intensive Prozesse gegenüber rechenzeitintensiven Prozessen.

☐ korrekt ☐ inkorrekt

Eine Deadline-Verletzung eines "Soft Real-Time" Prozesses hat fatale Folgen; deshalb muss eine solche Deadline von einem Scheduling-Algorithmus auf jeden Fall eingehalten werden.

☐ korrekt ☐ inkorrekt

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 Industrierobotern 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.

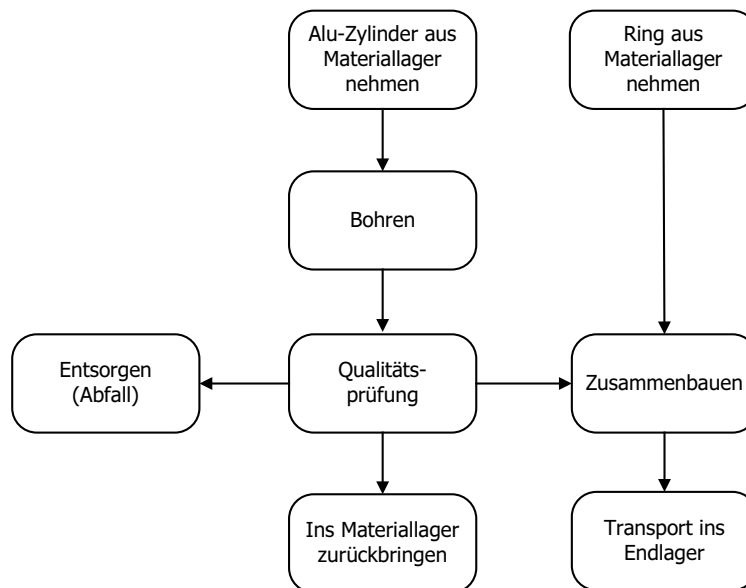


Abbildung 1: Abfolge der Arbeitsschritte

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 Semaphorearrays (`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!

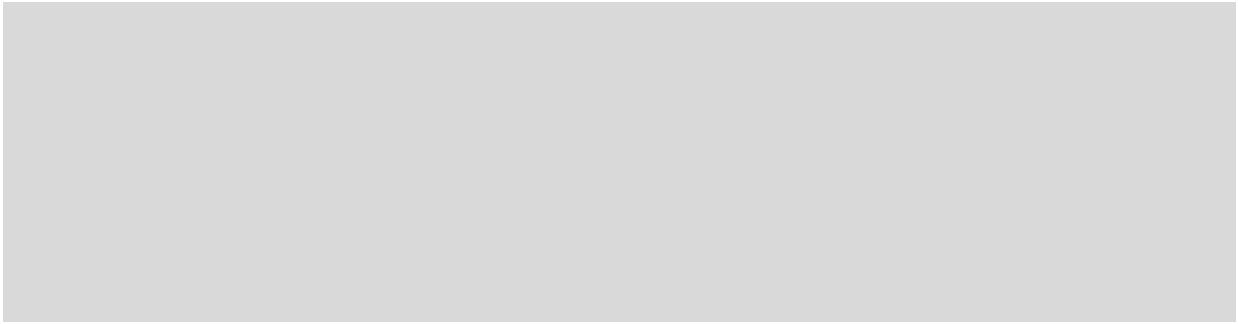
b) Vervollständigen Sie die Programme der Industrieroboter. (30)

Verwenden Sie hierzu die folgenden Routinen:

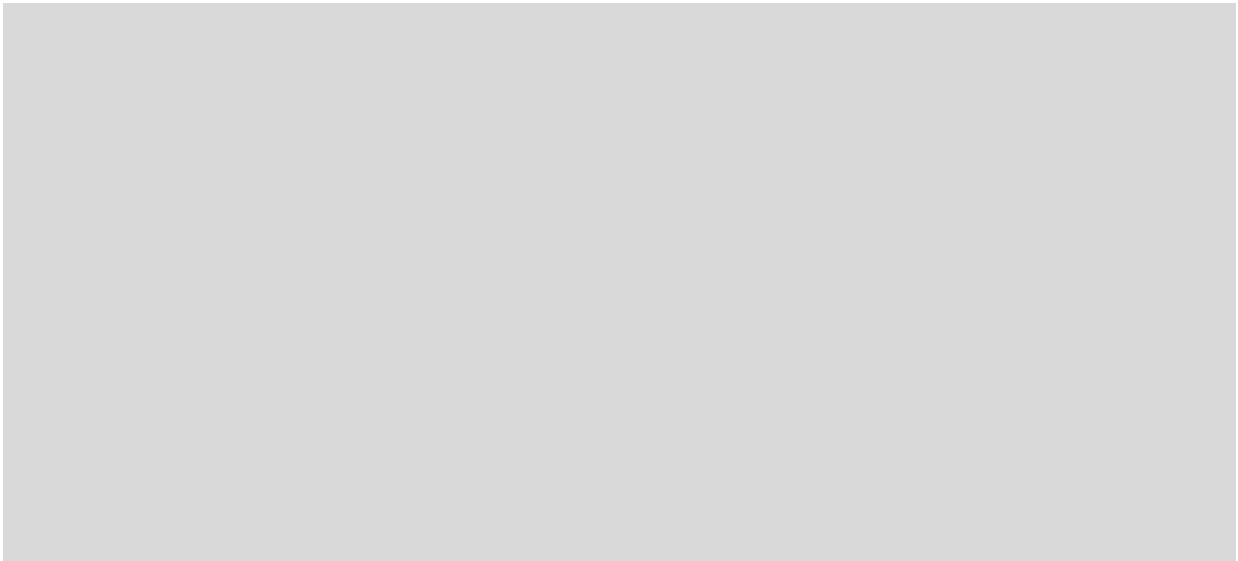
- *Nimm(roboter)* nimmt das Teil vom Industrieroboter $roboter \in \{roboter1, \dots, roboter5\}$ entgegen.
- *HoleAusLager(obj)* besorgt das Teil *obj* aus dem Materiallager. Diese Aktivität ist Roboter 1 und Roboter 2 vorbehalten.
($obj \in \{Aluzyylinder, Kunststoffring\}$).
- *Gib(ort)* reicht das 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

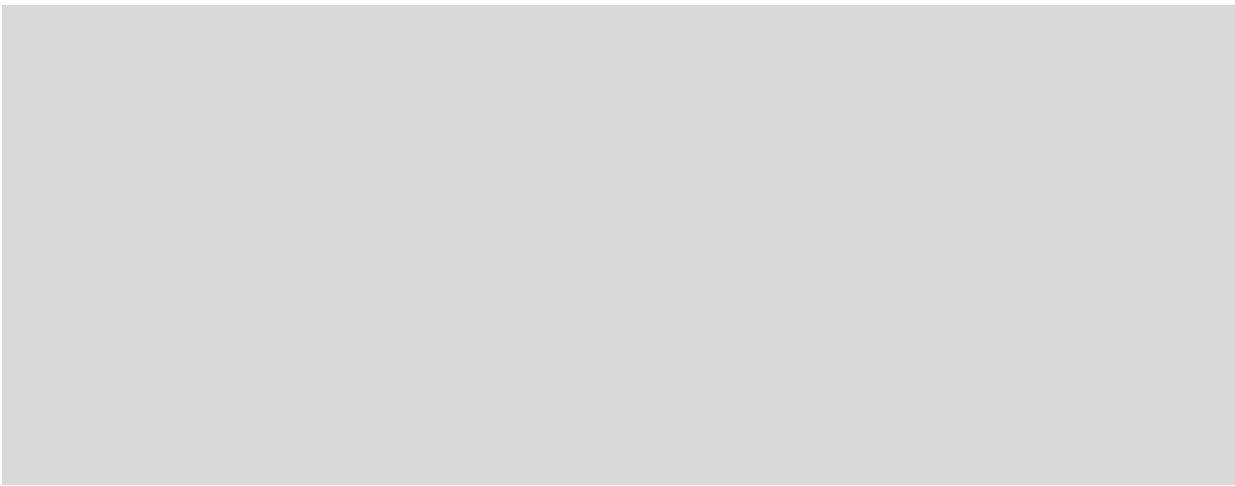


Roboter 3



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-Monotonic Scheduling Algorithmus für den Worst Case¹ 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.

	0	5	10	15	20	25	30
T1							
T2							
T3							
T4							
T5							
DV							

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

¹Im 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!

	0	5	10	15	20	25	30
T1							
T2							
T3							
T4							
T5							
DV							

	0	5	10	15	20	25	30
T1							
T2							
T3							
T4							
T5							
DV							

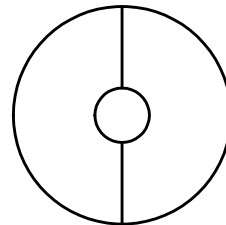
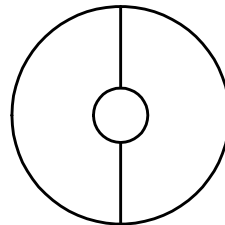
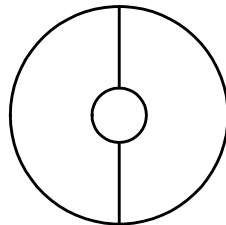
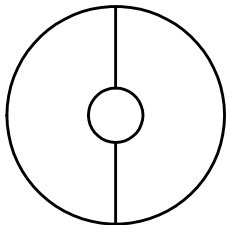
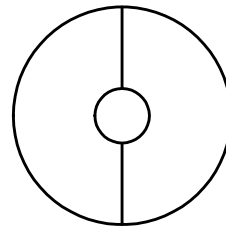
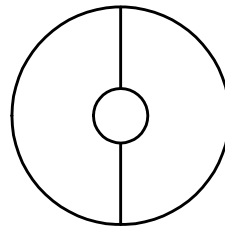
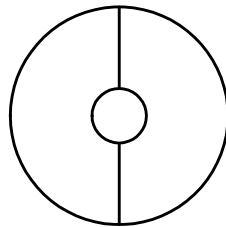
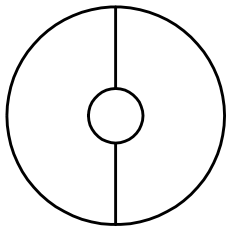
3 Speicherverwaltung - Replacement Policies(25)

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 letzte Mal zugegriffen wurde

Die angegebenen Zeiten sind die Ticks vom Zeitpunkt des Prozessesstarts (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

4 Prozessverwaltung und I/O (20)

a) (6)

Erklären Sie die beiden Begriffe *Mode Switch* und *Process Switch*. Was sind die Aufgaben jedes dieser Switches und wie stehen sie in Beziehung? Geben Sie für jeden der beiden Switches an, in welchem Execution Mode er abgearbeitet wird.

b) (4)

Nennen Sie die drei grundlegenden Mechanismen, mit denen die Kontrolle von einem laufenden Prozess an das Betriebssystem übergeht.

c) (10)

Geben Sie die Schritte an, die die Prozessverwaltung eines Multiprocessing-Betriebssystems im Rahmen einer blockierenden I/O-Operation (z.B. Lesen eines Zeichens von der Tastatur) ausführt. Nehmen Sie an, dass während dieser I/O Operation genau ein anderer Prozess zum Laufen kommt. Geben Sie für jeden Schritt an, in welchem Execution Mode des Betriebssystems er abläuft.





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.

Punkt	erreichbar	nicht erreichbar
S_1	<input type="radio"/>	<input type="radio"/>
S_2	<input type="radio"/>	<input type="radio"/>
S_3	<input type="radio"/>	<input type="radio"/>
S_4	<input type="radio"/>	<input type="radio"/>
S_5	<input type="radio"/>	<input type="radio"/>
S_6	<input type="radio"/>	<input type="radio"/>

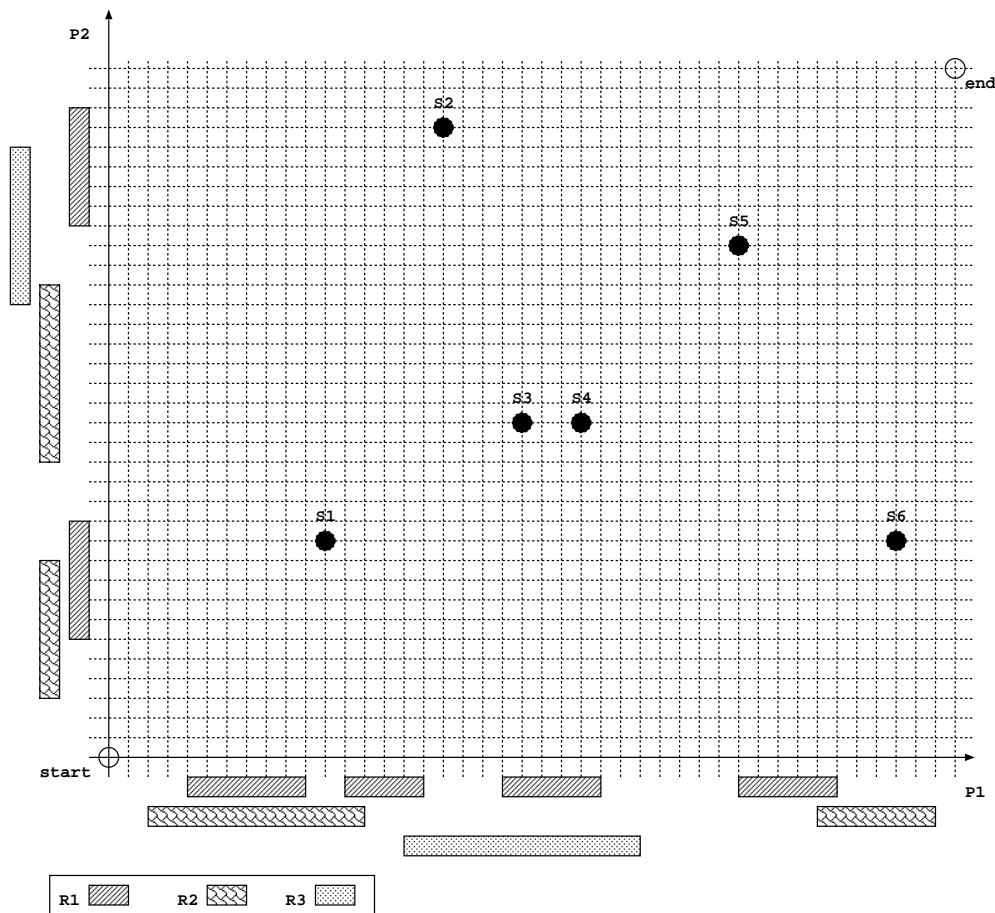


Abbildung 1: Abarbeitungs Diagramm

- Beschreiben Sie den Unterschied zwischen Process Initiation Denial und Resource Allocation Denial:

- Was ist Starvation:

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.

- `BlechInDenOfen()` Stellt das Backblech in den Ofen, nimmt es in Betrieb und wartet bis die Maroni fertig sind. Beim Terminieren ist das Backrohr bereits abgedreht und frei.
- `TopfHolen()` Nimmt einen Topf und stellt diesen zum Glühweinxmischen bereit.
- `GluehweinMischen()` Gibt Gewürze sowie Wein & Wasser für **70 Portionen** Glühwein in einen bereitgestellten Topf.
- `TopfAufDenOfen()` Setzt einen gefüllten Topf auf eine freie Herdplatte, nimmt diese in Betrieb und terminiert wenn der Glühwein fertig ist. Beim Terminieren ist die Herdplatte bereits abgedreht und frei.

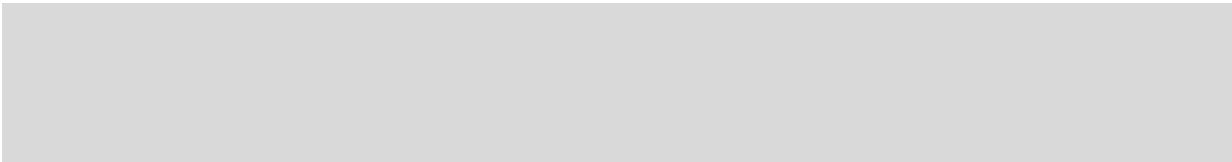
2.1 Initialisierung (5)



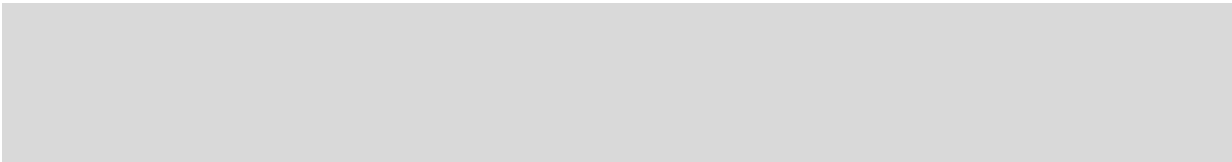
2.2 Hungriger Student (3)

Implementieren Sie das Programm `hungriger_Student`, das auf allen Gästen parallel abläuft.

```
while(hungrig || durstig){
    if(durstig){
```



```
        durstig = Trinken() /* trinkt den Gluehwein & aktualisiert durstig */
    }
    if(hungrig){
```

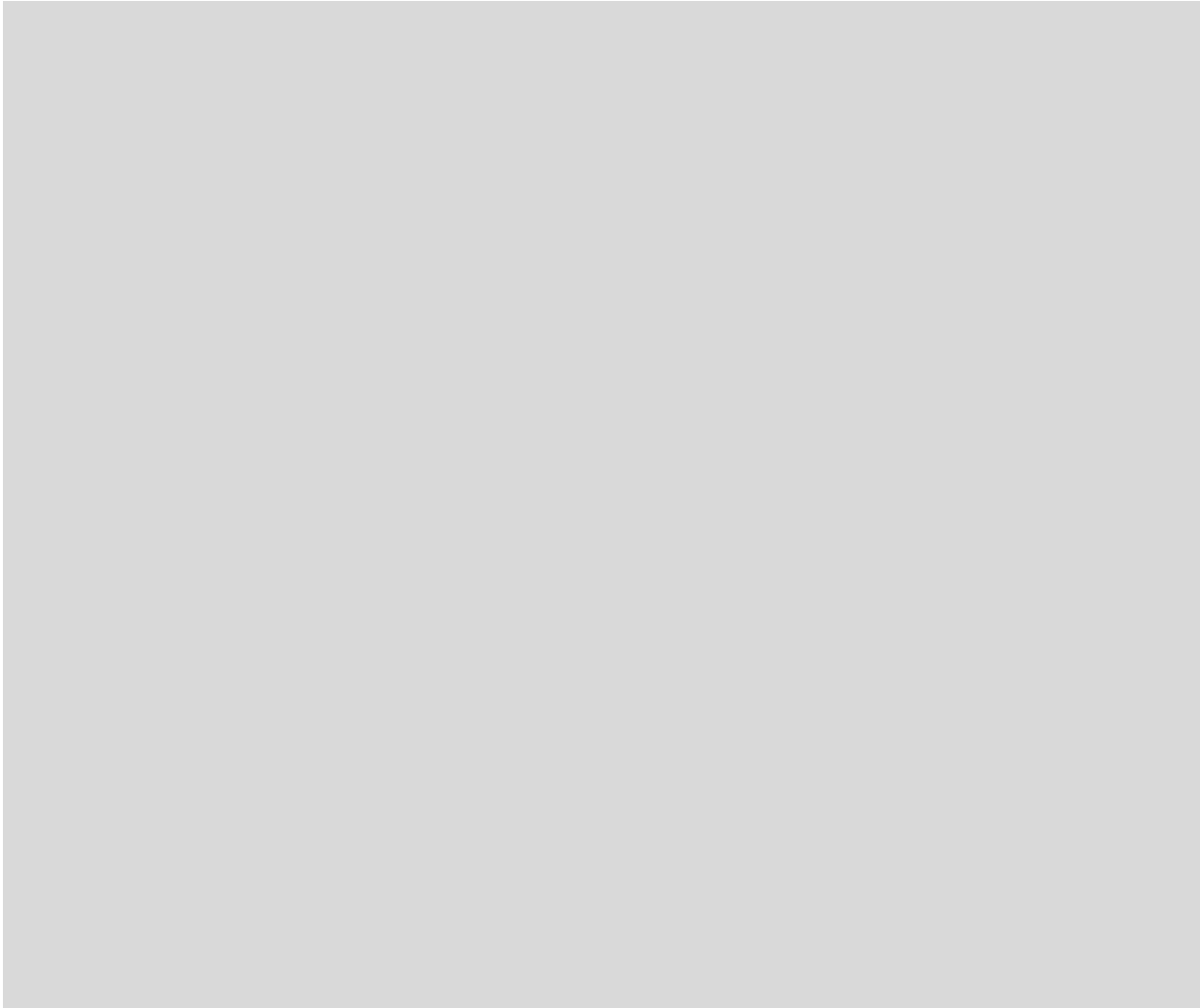


```
        hungrig = Essen() /* Iszt die Maroni & aktualisiert hungrig */
    }
}
Heimtorkeln()
```

2.3 Maronibrater (11)

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

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

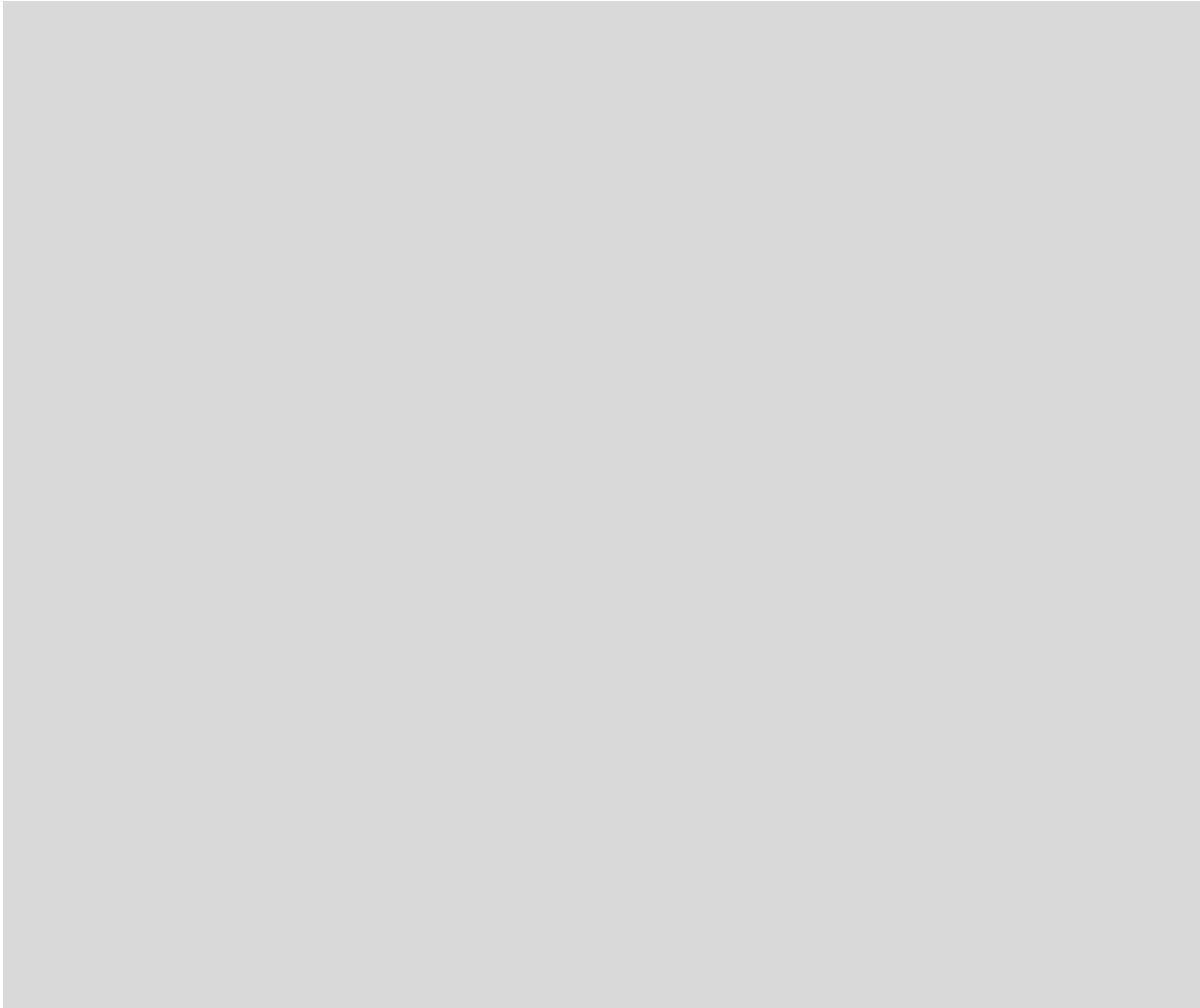


```
    GebtDenHungerden()  
}  
Aufräumen()
```


2.4 Glühweinkoch (11)

Implementieren Sie das Programm `Gluehwein_Koch` das auf allen Glühweinköchen parallel abgearbeitet wird.

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



```
    GebtDenDurstigen()  
}  
Aufräumen()
```

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-Counter.

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.

Ausgangssituation

Translation Lookaside Buffer (TLB):

Page#	Frame#	FIFO
22	0xFAC	2
3	0x723	1
12	0x333	3

Page Table:

Page#	Frame#
0	0x2FF
1	0x341
2	0x121
3	0x723
...	...
12	0x333
...	...
22	0xFAC
23	
24	
25	
...	...
37	
38	
39	
...	...
6A	0xA1C
6B	0xAFF

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

Virtuelle Adresse

0x6A001B

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x245677

Physikalische Adresse

TLB Hit ☐ Ja ☐ Nein

Page Hit ☐ Ja ☐ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

b) Page-Replacement Algorithms(9)

Ein Prozess referenziert die Pages A,B,C,D und E laut vorgegebenen Zugriffsschema. Simulieren Sie das Verhalten der entsprechenden Page-Replacement Algorithmen in der dafür vorgesehenen Tabelle. Am Anfang ist der TLB leer. Bei mehrdeutigen Lösungsmöglichkeiten geben Sie bitte eine gültige an. Markieren Sie einen Page Fault mit \checkmark (letzte Zeile).

OPTIMAL

page fault?

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

Anzahl d. page

faults:

LRU

page fault?

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

Anzahl d. page

faults:

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?

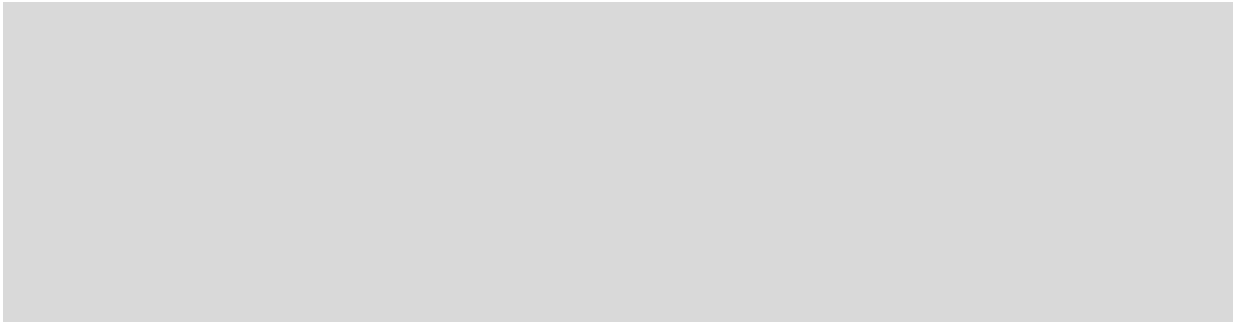
Zugriff(e)

- 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

4 Security (20)

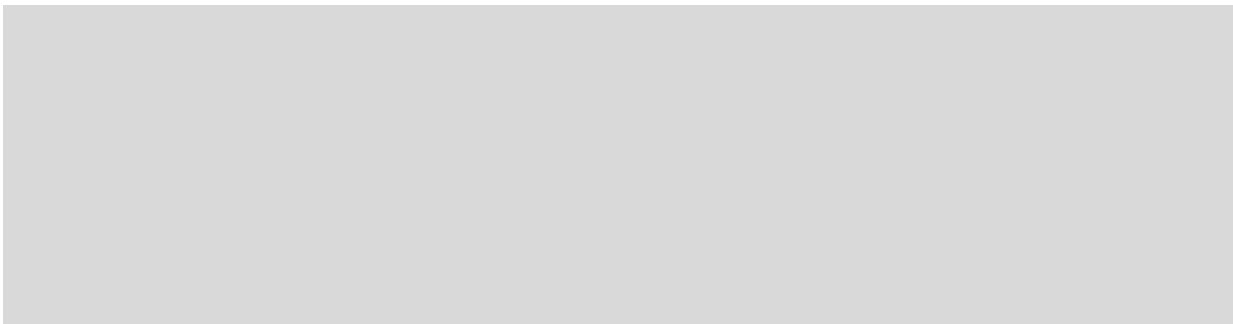
a) (3)

Was beschreibt das Modell von Bell und LaPadula?



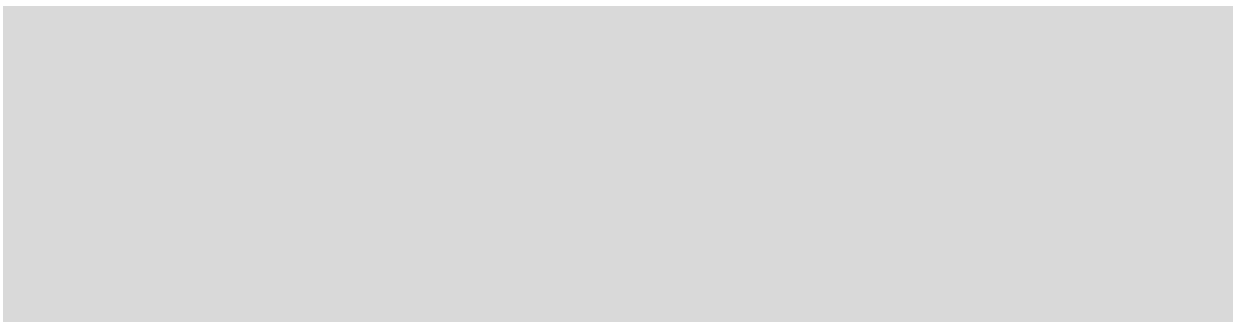
b) (4)

Im Modell von Bell und LaPadula gibt es die *Simple Security Property* und die *★-Property*. Beschreiben Sie für jeden der beiden Begriffe was er besagt und was das damit verbundene Konzept bewirken soll.



c) (5)

Beschreiben Sie die Funktionsweise von *Public Key Verschlüsselungsverfahren*. Erklären Sie insbesondere, welche Schlüssel man bei diesem Verfahren benötigt, wer welche Schlüssel kennen darf und wie die Schlüssel von den Sendern und Empfängern von Daten verwendet werden.



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?



KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(25)

2.)(30)

3.)(23)

4.)(22)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und zwei Ressourcen R_1, R_2 . Die Ressourceallokationen der Prozesse in Abhängigkeit des jeweiligen Prozessfortschritts sind in Tabelle 1 eingetragen.

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 Fläche, 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. Begründung)



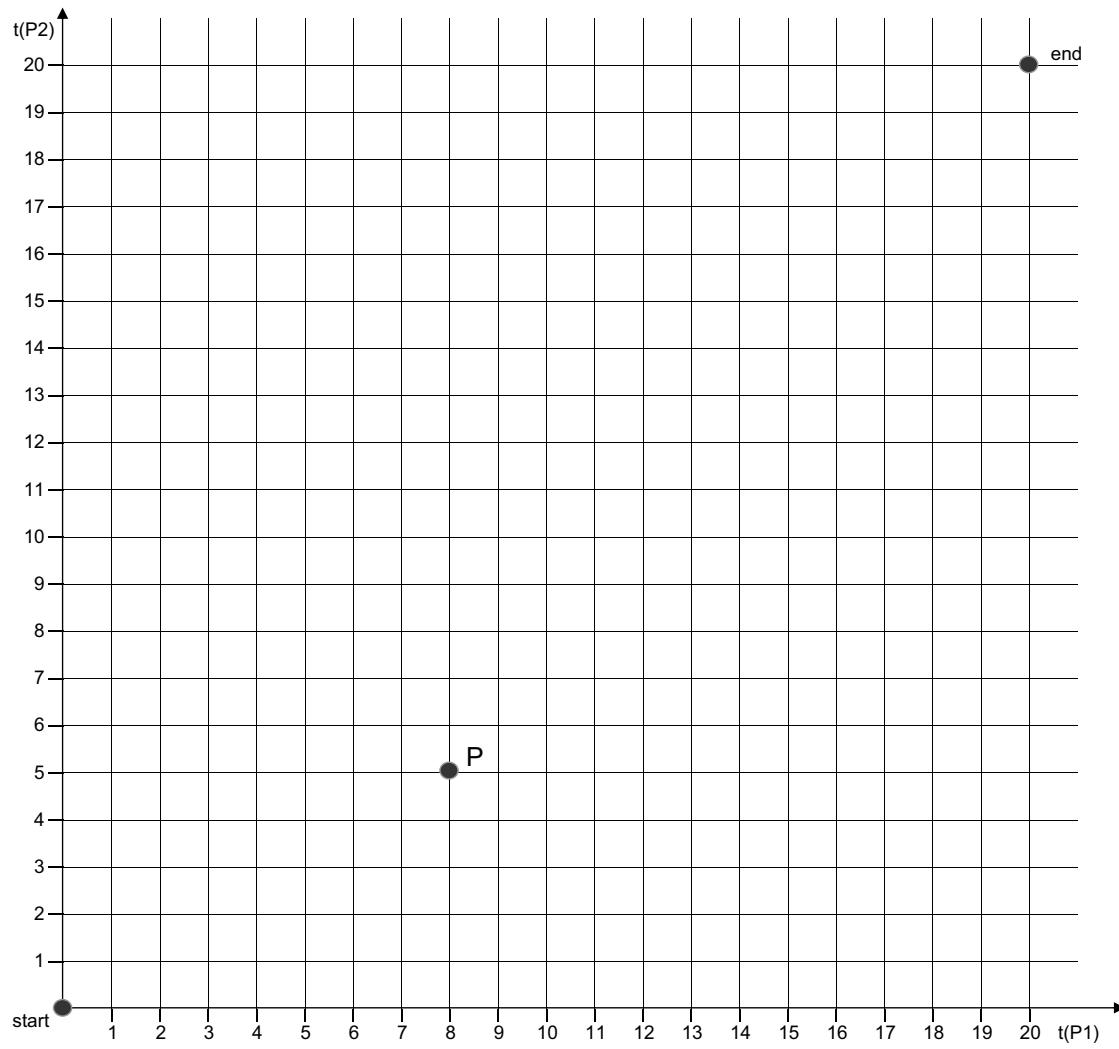


Abbildung 1: Abarbeitungs Diagramm

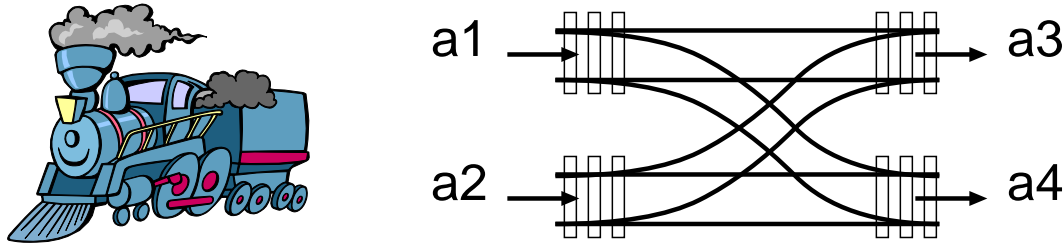
1.2 Deadlock Avoidance

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 Begründung)

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 $\text{von} \notin \{\mathbf{a1}, \mathbf{a2}\}$ oder $\text{nach} \notin \{\mathbf{a3}, \mathbf{a4}\}$ gelten (d.h. eine nicht im System vorgesehene 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 $\text{von} \in \{\mathbf{a1}, \mathbf{a2}, \mathbf{a3}, \mathbf{a4}\}$). 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

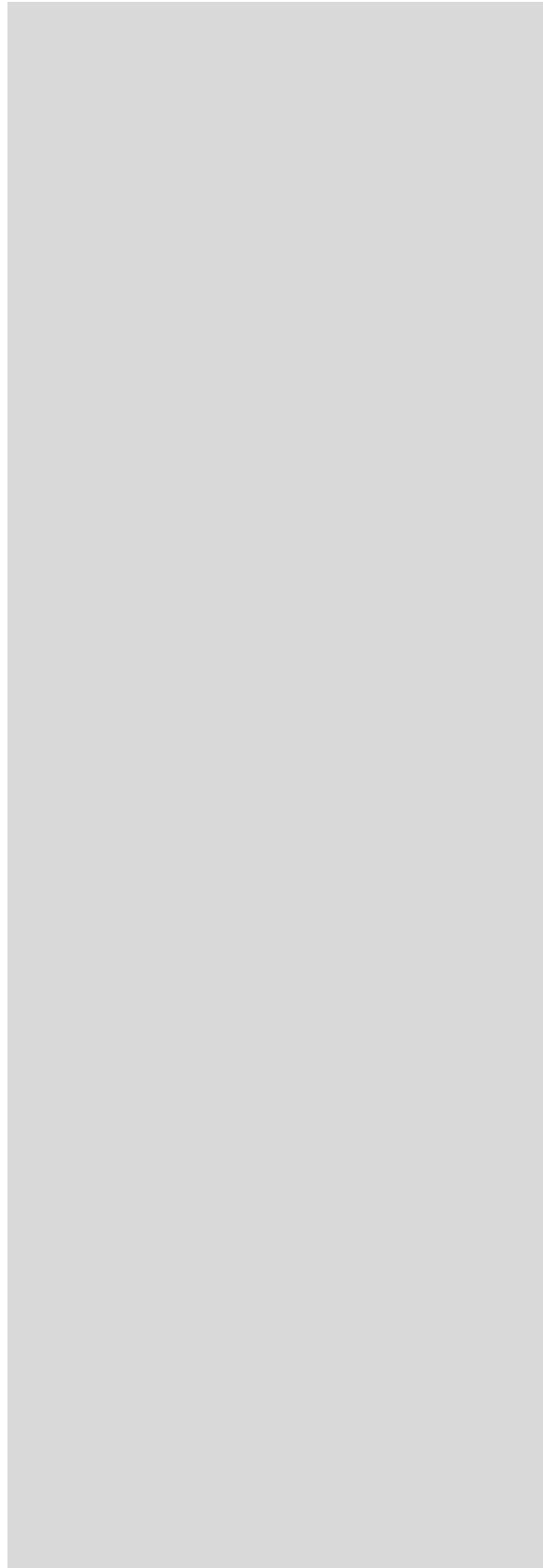
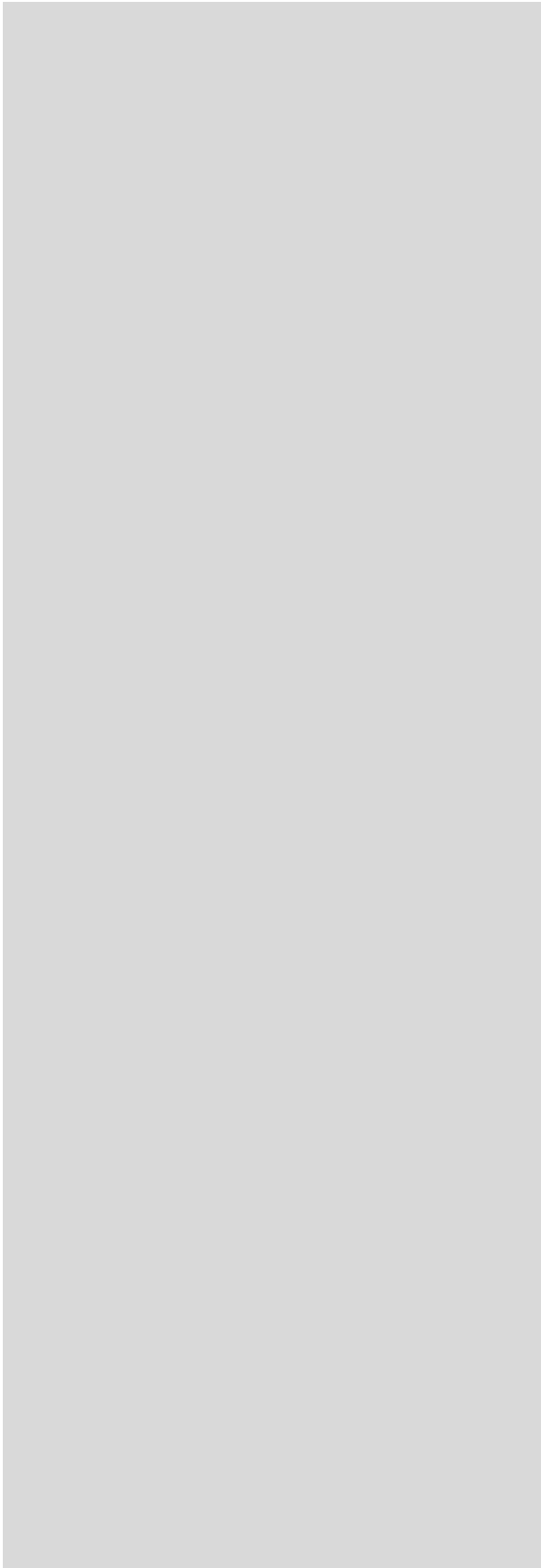
current = GERADE;

END

c) Zugkontrolle (14)

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

BEGIN



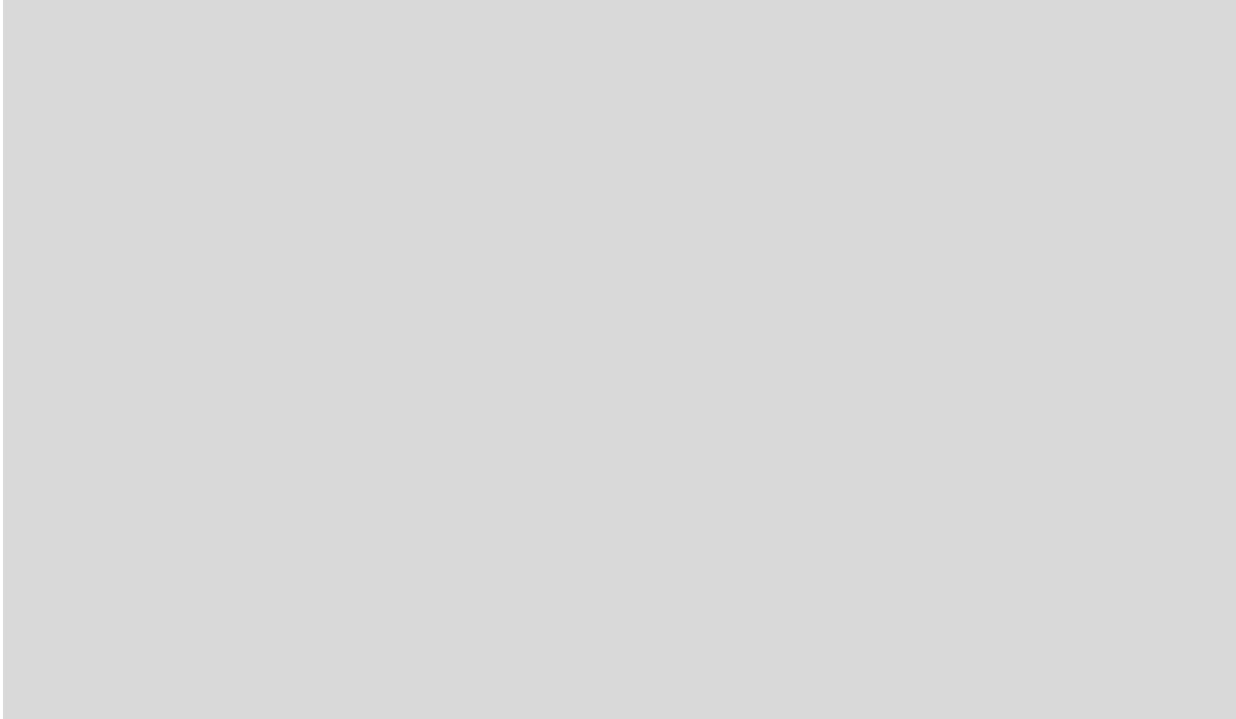
END

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
...	...

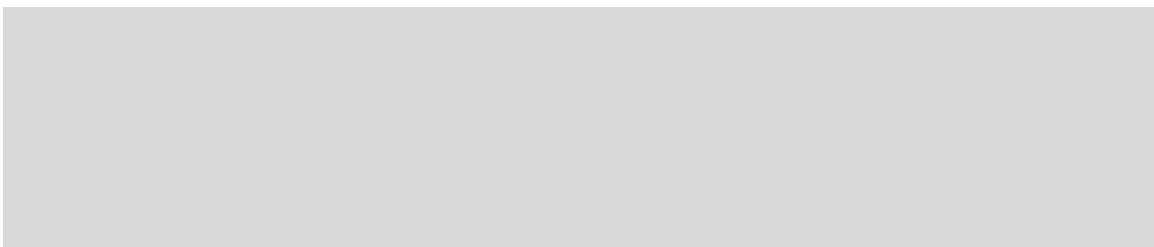
Ermitteln Sie unter Benützung obiger Tabellen die physikalischen Adressen zu folgenden virtuellen Adressen (ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld) (13P, pro Fehler -2):

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

A																		
B																		
C																		
D																		

0 5 10 15

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

A																		
B																		
C																		
D																		

0 5 10 15

Ersatzvorlage: Scheduling nach dem -Verfahren:

A																		
B																		
C																		
D																		

0 5 10 15

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.

Prüfung Betriebssysteme

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(35)

2.)(25)

3.)(20)

4.)(20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (35)

Die Fähre *M/s Mariella* transportiert Fahrzeuge und Personen zwischen Helsinki und Stockholm. Um eine maximale Auslastung der Fähre zu erreichen, beschließt der Kapitän, die Fahrt erst dann zu beginnen, wenn die Fähre voll beladen ist.



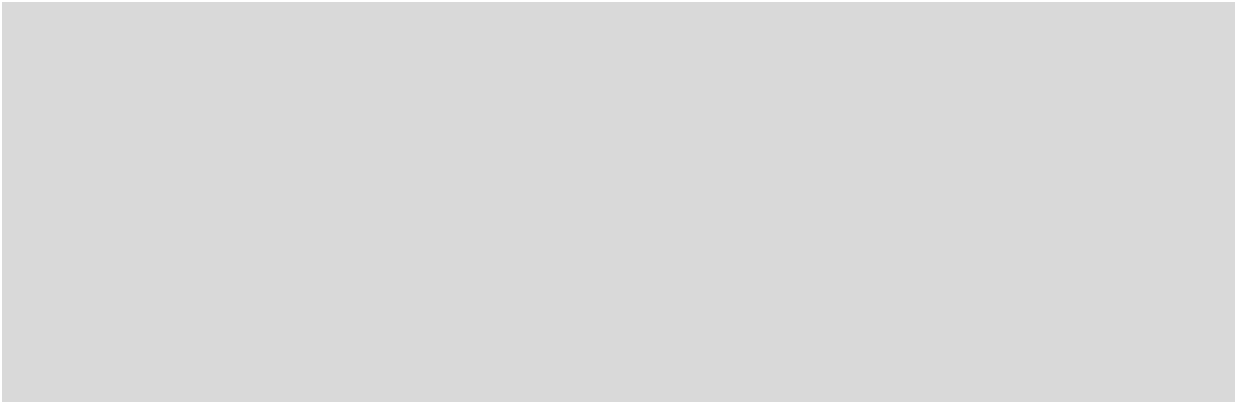
Die Fähre besitzt Platz für 2500 Passagiere, 200 PKW und 30 LKW.

Implementieren Sie Synchronisationsroutinen für den Betrieb der Fähre. Berücksichtigen Sie dabei folgende Randbedingungen:

- Für die Verwendung von *Semaphoren* stehen Ihnen die Funktionen `initsem(name, initialwert)` und `P(name)` sowie `V(name)` zur Verfügung.
- Für die Verwendung von *Eventcountern* stehen Ihnen die Funktionen `initcvc(name, initialwert)`, `read(name)`, `await(name)` und `advance(name, value)` zur Verfügung.
- Für die Verwendung von *Sequencern* stehen Ihnen die Funktionen `initseq(name, initialwert)`, `read(name)` und `sticket(name)`. Die Synchronisation mit globalen Variablen ist verboten.
- Verwenden Sie so wenig Synchronisationskonstrukte wie möglich.

1.1 Ressourcen anlegen und initialisieren

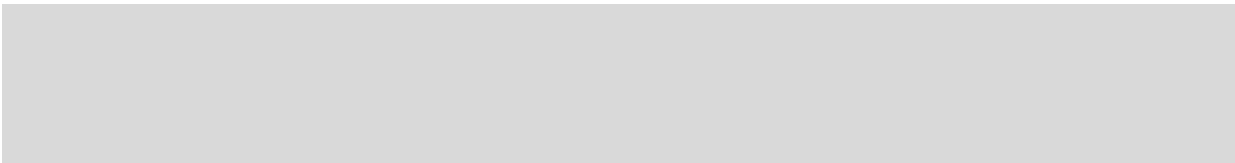
Die Initialisierung wird nur ein einziges Mal am Anfang aufgerufen. Bitte definieren Sie hier alle benötigten Ressourcen.



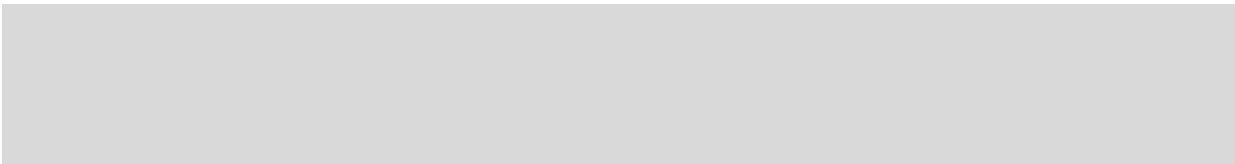
1.2 Kapitän

Der Kapitän legt mit dem Schiff an, koordiniert die Entladung der Fahrzeuge und schickt die Passagiere vom Schiff. Danach wartet der Kapitän bis das Schiff wieder voll beladen ist und die Fahrzeugrampe und der Passagierzugang geschlossen sind und legt dann wieder in Richtung Zielhafen ab. Das Beladen der Fähre mit PKW, LKW und das Aufnehmen von Personen soll gleichzeitig erfolgen.

```
do forever{  
  
    Fahre_in_Hafen()  
  
    Anlegen()  
  
    Ankern()  
  
    Installiere_Passagierrampe()  
  
    Schicke_Passagiere_vom_Schiff()
```



```
    Oeffne_Fahrzeugrampe()  
  
    Entlade_Fahrzeuge()
```




```
Schliesse_Fahrzeugrampe()  
  
Entferne_Passagierrampe()  
  
Anker_lichten()  
  
Fahre_zum_Zielhafen()  
  
}
```

1.3 PKW beladen

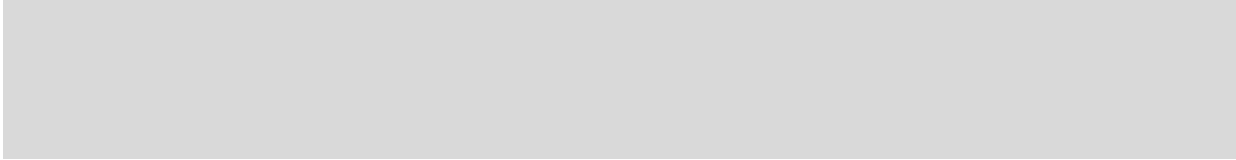
Dieses Programm dient zum Beladen der Fähre mit einem PKW. Es kann parallel mit anderen Programmen und mehrfach gestartet werden. Achten Sie darauf, dass die Fähre am Anlegeplatz steht, die Laderampe heruntergefahren ist und noch genug Platz auf der Fähre ist, andernfalls soll die Funktion blockieren, bis die nächste Fähre zum Beladen bereit ist. Diese Funktion kann gleichzeitig in mehreren Instanzen aufgerufen werden, wobei immer nur **ein** Parkvorgang (sowohl PKW als auch LKW) gleichzeitig durchgeführt werden können. Pro Fahrzeug muss jeweils ein Passagierplatz für den Fahrer reserviert werden.

```
Parke_PKW_auf_Fähre()
```


```
Fahrer_bezieht_Kabine()
```

1.4 LKW beladen

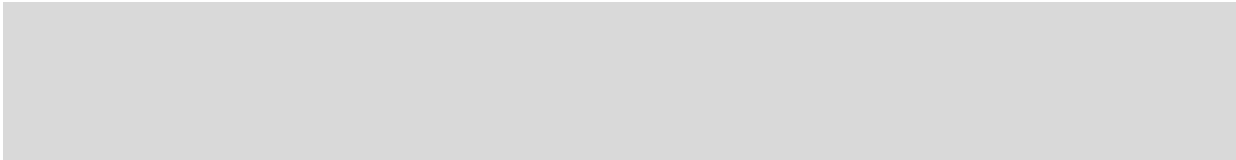
Dieses Programm dient zum Beladen der Fähre mit einem LKW. Es kann parallel mit anderen Programmen und mehrfach gestartet werden. Achten Sie darauf, dass die Fähre am Anlegeplatz steht, die Laderampe heruntergefahren ist und noch genug Platz auf der Fähre ist, andernfalls soll die Funktion blockieren, bis die nächste Fähre zum Beladen bereit ist. Diese Funktion kann gleichzeitig in mehreren Instanzen aufgerufen werden, wobei immer nur **ein** Parkvorgang (sowohl PKW als auch LKW) gleichzeitig durchgeführt werden darf. Pro Fahrzeug muss jeweils ein Passagierplatz für den Fahrer reserviert werden.



```
Parke_LKW_auf_Fähre()
```

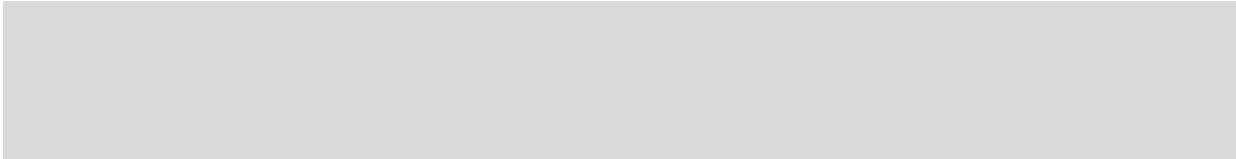


```
Fahrer_bezieht_Kabine()
```

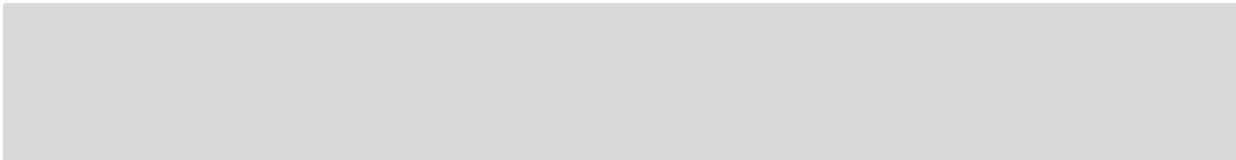


1.5 Neuer Passagier

Dieses Programm dient zum Aufnehmen eines neuen Passagiers. Ein neuer Passagier soll sich folgendermaßen verhalten: Ist er unter den glücklichen 2500 Passagieren, so versucht er an Bord zu gehen, ansonsten wartet er auf die nächste Fähre.



```
Passagier_bezieht_Kabine()
```



2 Scheduling (25)

Round-Robin und Feedback Scheduling

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 = -1; Zuteilung = 0). Ein eintreffender Task wird immer ans Ende der Ready Queue gestellt.

Task	Arrival Time	Service Time
A	-1	5
B	2	3
C	3	6
D	5	4
E	6	2

Tragen Sie in der Zeile **P** 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 Ready Queue 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).

	-1	0				5					10					15					20
P		A																			
Ready Queue	A																				

Abbildung 1: Round Robin Scheduling

	-1	0				5					10					15					20
P		A																			
Ready Queue	A																				

Abbildung 2: Feedback Scheduling

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.

Task	Ausführungszeit	Periodendauer
A	2	6
B	2	12
C	1	4
D	1	5

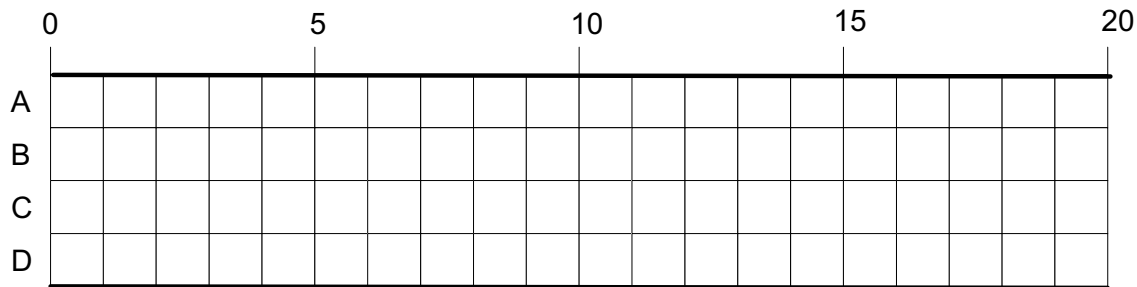


Abbildung 3: Rate-Monotonic Scheduling

Kann es bei diesem Taskset zu einem Deadline-Miss kommen (ja/nein/undefiniert), begründen Sie Ihre Antwort ?

Verständnisfragen

Was bedeutet Starvation?

Geben Sie 3 Scheduling Methoden an bei denen es zu Starvation kommen kann:

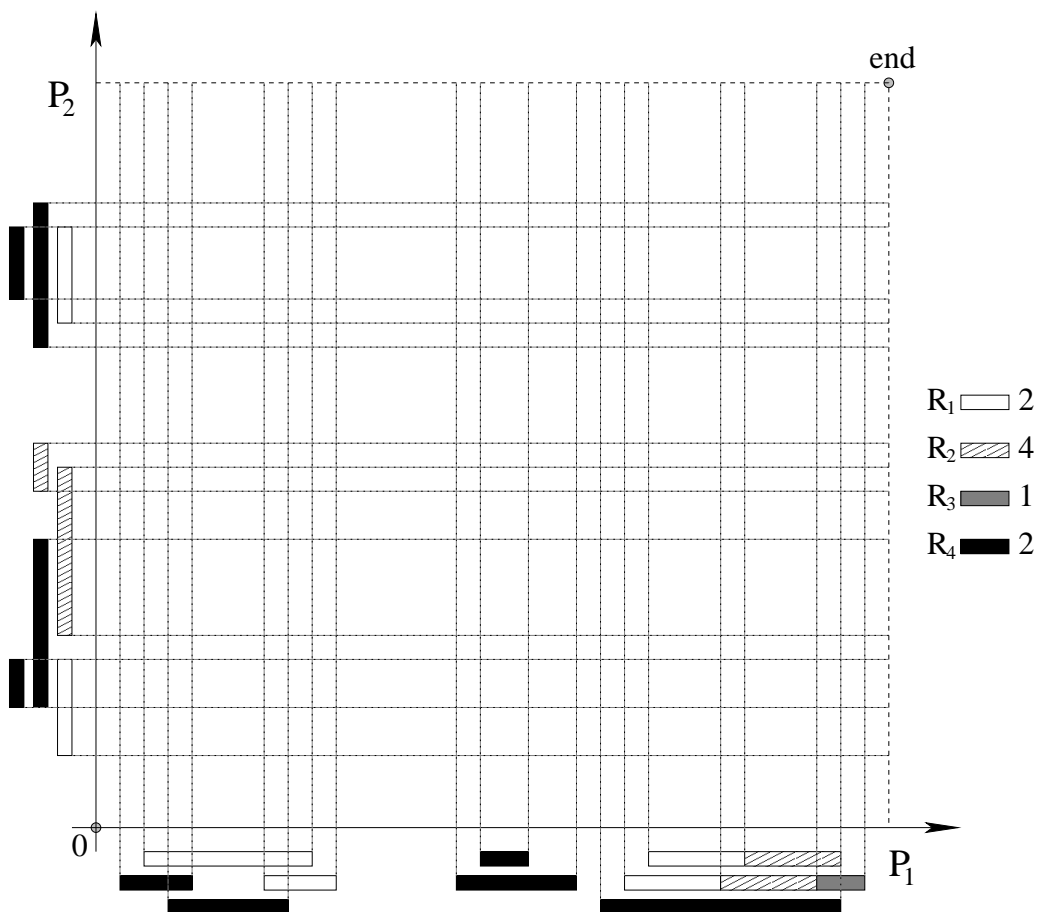
Erklären Sie das **gemeinsame** Problem dieser 3 Methoden, wieso es zu Starvation kommen kann:

3 Deadlock (20)

Zwei Prozesse P_1 und P_2 teilen sich gemeinsam Ressourcen des Systems. Der zum System gehörige Ressource Vector ist: $(R_1, R_2, R_3, R_4) := (2, 4, 1, 2)$. Werden von einem Prozess Ressourcen benötigt, welche der andere Prozess gerade belegt, so wird er mindestens bis zum Freiwerden der notwendigen Ressourcen verzögert. Welche Ressourcen von einem Prozess im Laufes seines „Lebens“ benötigt werden, entnehmen Sie bitte der Abbildung.

Durch einen Kantenzug von 0 nach end lässt sich nun der Fortschritt der beiden Prozesse bei der (quasi)parallelen Abarbeitung darstellen.

1. Umranden und schraffieren Sie in der Abbildung all jene Bereiche, durch welche ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen darf.
2. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht passieren kann ohne in einem Deadlock zu enden. Kennzeichnen Sie diese Bereiche durch ein „D“.
3. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht erreichen kann obwohl kein Ressourcenkonflikt in diesem Bereich besteht. Kennzeichnen Sie diese Bereiche durch ein „U“.
4. Zeichnen Sie einen deadlockfreien Kantenzug von 0 nach end ein.



4 Memory Management (20)

Ein Betriebssystem verwendet Speicherseiten der Größe 4KB (2^{12} Bytes). Die Seitentabelle ist als *Inverted Page Table (IPT)* realisiert, d.h., die Seitentabelle hat soviele Einträge, wie es Speicherframes im System gibt. In jedem der Einträge ist für die im entsprechenden Frame geladene Seite die Prozessnummer des Prozesses, zu dem diese Seite gehört, sowie die Seitennummer innerhalb des Prozesses gespeichert. Dabei werden in jedem Eintrag 8 Bits für die Prozessnummer und 8 weitere Bits für die Seitennummer verwendet.

Die Abbildung zeigt eine Folge von sechs Adressreferenzen bestehend aus jeweils Prozessnummer (PID) und logischer Adresse im Prozess. Diese Speicherzugriffe sind von oben nach unten nacheinander durchzuführen, wobei bei Page Faults die Clock Policy zum Replacement von Seiten angewandt wird. Die aktuellen Werte der Use Bits sind links neben der IPT angegeben, der Pfeil kennzeichnet die aktuelle Position des Clock-Zeigers (Laufrichtung des Zeigers ist zyklisch von oben nach unten).

Geben Sie in den Tabellen den Inhalt der IPT und der Use Bits, sowie den Clock-Zeiger nach jeder Speicherreferenz an. Tragen Sie außerdem in die Tabelle rechts oben für jeden Speicherzugriff die physikalische Adresse ein.

Use Bit	IPT	1		PID	log. Adresse	Physikalische Adresse
0	0ea5					
0	c503					
→ 0	477e			47	7effe	
1	0e12			0e	2e000	
0	0e2f			18	23012	
0	3b01			0e	0ea50	
0	3b01			0e	2fe02	
1	0e00			0e	0e240	
0	1823					

2		3		4		5		6	

Prüfung Betriebssysteme

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(25)

3.)(20)

4.)(25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!**1 Speicherverwaltung (30)****a) Lokalität von Prozessen – Working Sets (8)**

Das Working Set Modell beschreibt das Speicherzugriffsverhalten von Prozessen. In diesem Modell ist das Working Set $W(t, \Delta)$ definiert als die Menge der Speicherseiten, die in den letzten Δ Zeiteinheiten vor dem Zeitpunkt t referenziert wurden.

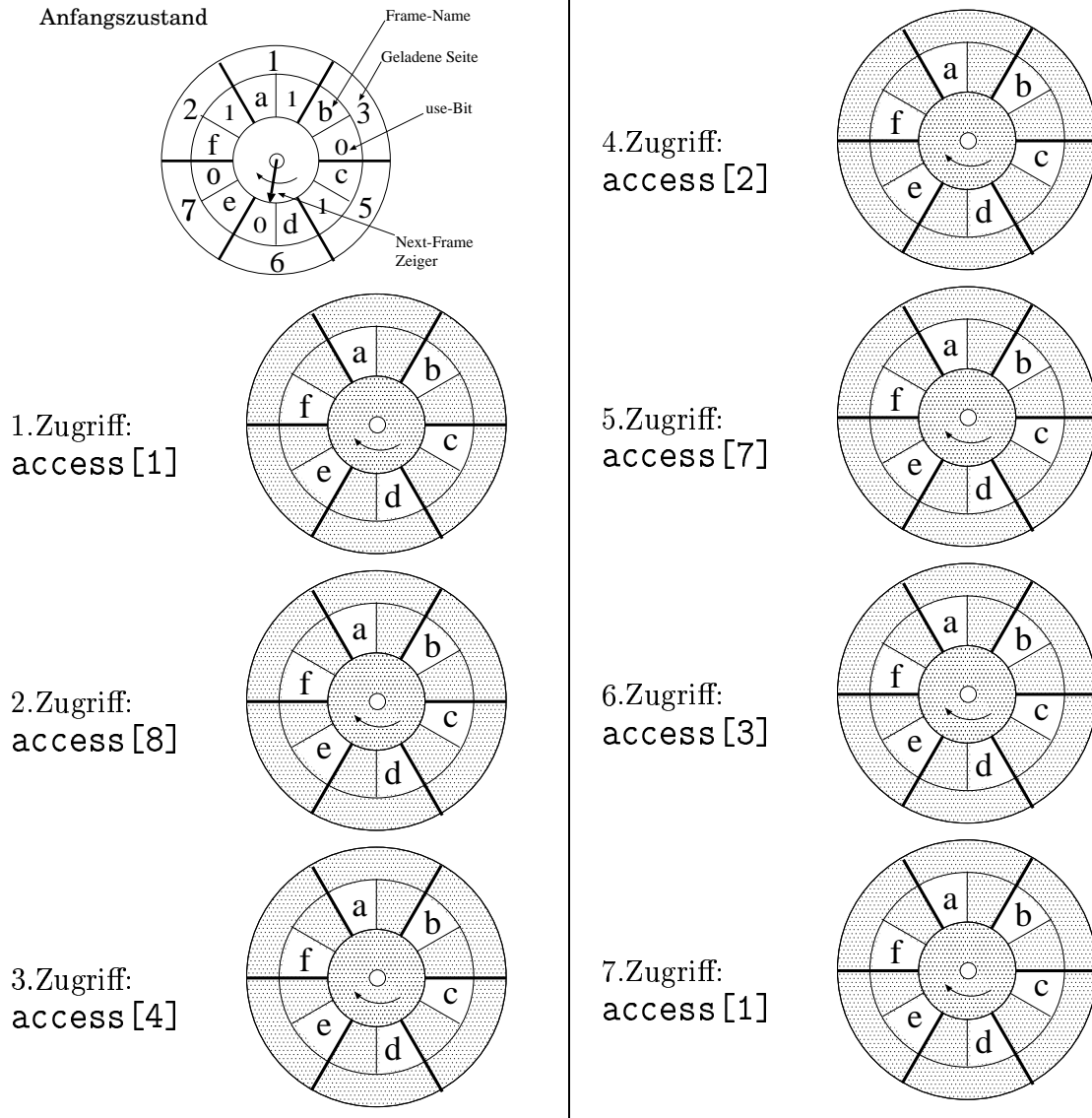
Die folgende Tabelle beschreibt eine Folge von Speicherseitenzugriffen eines Prozesses. In Spalte 2 jeder Zeile steht die Nummer der Seite, auf die in der Zeitscheibe $(t - 1, t]$ zugegriffen wird. Geben Sie in den leeren Feldern die Working Sets $W(t, \Delta)$ für Δ gleich 3, bzw. 4 an.

t	Seitennr.	$W(t, 3)$	$W(t, 4)$
1	0x23	0x23	0x23
2	0x18	0x23, 0x18	0x23, 0x18
3	0x19	0x23, 0x18, 0x19	0x23, 0x18, 0x19
4	0x18	0x18, 0x19	0x18, 0x19
5	0x23	0x19, 0x18, 0x23	0x19, 0x18, 0x23
6	0x19	0x18, 0x23, 0x19	0x18, 0x23, 0x19
7	0x18	0x23, 0x19, 0x18	0x23, 0x19, 0x18
8	0x3F	0x19, 0x18, 0x3f	0x23, 0x19, 0x18, 0x3f
9	0x23	0x18, 0x3f, 0x23	0x19, 0x18, 0x3f, 0x23
10	0x18	0x3f, 0x23, 0x18	0x18, 0x3f, 0x23

b) Replacement Policy (11)

Simulieren Sie das Verhalten einer *clock policy* zum Auslagern von Speicher-Seiten, wenn neue Seiten geladen werden müssen.

Für jede Seite im Speicher existiert ein *use*-Bit. Die einzelnen Plätze im Hauptspeicher sind in diesem Beispiel mit Buchstaben von a ... f benannt. Der *next frame*-Zeiger bewegt sich in den verwendeten Grafiken im Uhrzeigersinn weiter.



Zeichnen Sie nun in den obigen Grafiken

- die Position des *next frame*-Zeigers
- die Inhalte aller Frames (geladene Seiten)

für die Ausführung von Befehlen mit Speicherzugriffen ein. Ein Befehl `access[x]` steht für einen Speicherzugriff auf die Seite **x**. Geben Sie den gefragten Speicherzustand **nach** der Ausführung des jeweiligen Befehls an, wobei Sie für die Befehle von einer sequentiellen Reihenfolge ausgehen können.

c) Verständnisfragen (11)

Kreuzen Sie bei den folgenden Aussagen an, ob diese richtig oder falsch sind.

- ☐ *richtig* Durch Vergrößerung des *Working Sets* von Prozessen kann man das Risiko
- ☐ *falsch* von “thrashing” vermindern.

- ☐ *richtig* Die Clock Replacement Strategie liefert im Durchschnitt weniger Page
- ☐ *falsch* Faults als die Least Recently Used Replacement Strategy.

- ☐ *richtig* Große Seitengrößen bei reinem Paging führen zu kleinen Seitentabellen
- ☐ *falsch* (page tables).

- ☐ *richtig* Um die externe Fragmentierung zu reduzieren, muss man die *Page Size*
- ☐ *falsch* vergrößern.

- ☐ *richtig* Unter “prepaging” versteht man einen Mechanismus, bei dem *nur* Pages,
- ☐ *falsch* die in Zukunft referenziert werden, (und sonst keine) und damit benötigte
“Pages” (Seiten) vor ihrer Verwendung in den Hauptspeicher gebracht
werden.

- ☐ *richtig* Beim FIFO-Verfahren kann eine Vergrößerung der Anzahl der Arbeits-
- ☐ *falsch* speicherseiten zu einer Zunahme der Seitenfehler führen.

- ☐ *richtig* Eine virtuelle Adresse verweist immer auf eine Seite auf dem
- ☐ *falsch* Sekundärspeicher.

- ☐ *richtig* Die Free Frame List verwaltet bei der Segmentierung die freien Frames
- ☐ *falsch* im Speicher.

- ☐ *richtig* Bei der Clock Policy handelt es sich um eine Demand Paging Strategie.
- ☐ *falsch*

- ☐ *richtig* Die LRU-Strategie wird in der Praxis wenig verwendet, da sie gegenüber
- ☐ *falsch* den anderen Ansätzen eine hohe Seitenfehlerrate produziert.

- ☐ *richtig* Für einen gleich großen Hauptspeicher nutzt eine kleine Seitengröße das
- ☐ *falsch* Lokalitätsprinzip besser aus als eine große Seitengröße.

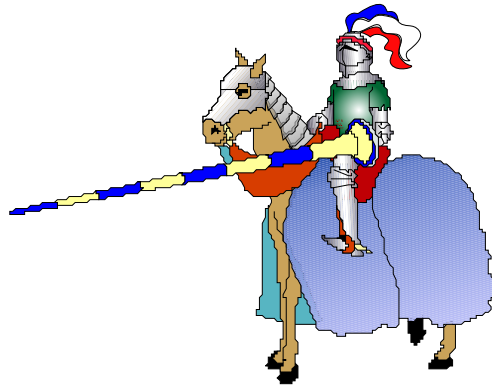
- ☐ *richtig* Ein virtueller Speicher kann sowohl mit Paging als auch mit Segmentie-
- ☐ *falsch* rung implementiert werden.

- ☐ *richtig* Beim Buddy System kann es zu externer Fragmentierung kommen.
- ☐ *falsch*

- ☐ *richtig* Beim Buddy System kann es zu interner Fragmentierung kommen.
- ☐ *falsch*

2 Synchronisation (25)

2.1 Semaphore (20)



Wir schreiben das Jahr 867. Ritter Kunibert ist ein angesehener und gefürchteter Turnierritter. Zum Leidwesen seiner zwei Knechte, ist Ritter Kunibert dem Vergnügen vor dem Turnier nicht abgeneigt. Und so kommt es wie es kommen muss, Ritter Kunibert ist am Morgen des Turniers noch völlig erschöpft von der Turnierparty und benötigt die Hilfe seiner Diener um sich passend einzukleiden. Da nicht mehr viel Zeit bis zu Kuniberts Auftritt bleibt ist es wichtig, dass

- Ritter Kunibert in der richtigen Reihenfolge eingekleidet und für das Turnier vorbereitet wird
- und dies unter Ausnutzung maximaler Parallelität geschieht.

Die folgende Tabelle listet die dafür notwendigen Schritte auf:

ID	Aktion	Funktion	benötigt	Akteur
A	Aufstehen	stand_up()	-	Kunibert
B	Stillhalten	hold_still()	A	Kunibert
C	Hose anziehen	pants()	B	Ruprecht
D	Kettenhemd anziehen	shirt()	B	Vladimir
E	Schuh links anziehen und zubinden	shoe(left)	C	Ruprecht
F	Schuh rechts anziehen und zubinden	shoe(right)	C	Vladimir
G	Beinschoner links befestigen	guard(shin, left)	E	Ruprecht
H	Beinschoner rechts befestigen	guard(shin, right)	F	Vladimir
I	Armprotektor links befestigen	guard(arm, left)	D	Ruprecht
J	Armprotektor rechts befestigen	guard(arm, right)	D	Vladimir
K	Brustpanzer fixieren	protector(front)	G,H,I,J	Ruprecht
L	Rückenpanzer fixieren	protector(back)	K	Ruprecht
M	Helm aufsetzen	helmet()	G,H,I,J	Vladimir
N	aufs Pferd setzen	get_on_horse()	L, M	Kunibert
O	Schild in die Hand geben	shield()	N	Ruprecht
P	Lanze ausrichten	lance()	N	Vladimir
Q	Kämpfen	fight_for_honor()	O,P	Kunibert

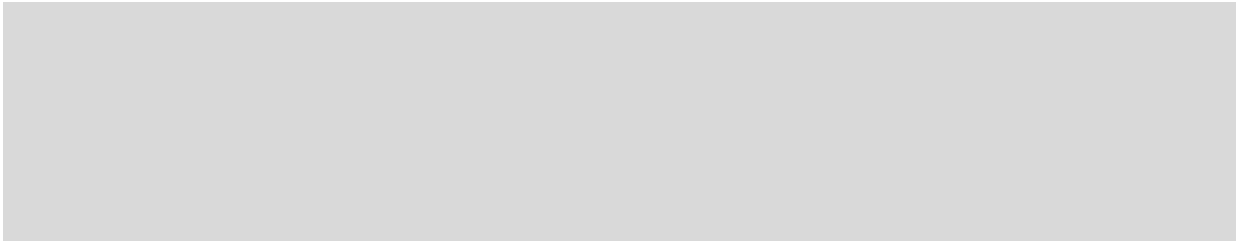
Das Feld `ID` enthält eine Kurzbezeichnung für eine **Funktion**, welche im Feld **Aktion** beschrieben ist. Um eine solche **Funktion** ausführen zu können, müssen davor eine oder mehrere andere Funktionen fertiggestellt worden sein: die IDs dieser Funktionen sind im Feld **benötigt** angegeben. Das Feld **Akteur** gibt die ausführende Person an.

Die Synchronisation ist mittels einer minimalen Anzahl von Semaphoren zu realisieren. Die Verwendung von globalen Variablen und busy waiting ist verboten.

Initialisierung

Benutzen Sie zur Initialisierung der Semaphore die Funktion `init_sem(sem,value)`, wobei `sem` der Name des Semaphores ist und `value` der entsprechende Initialisierungswert.

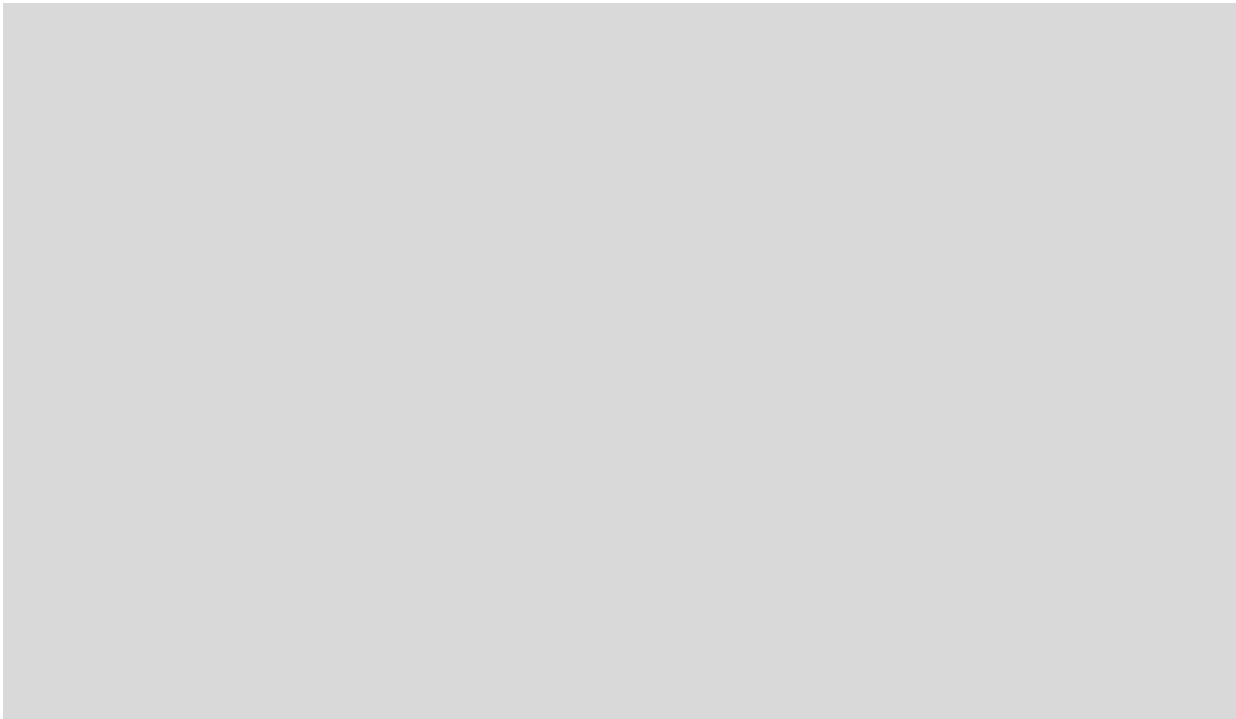
```
init() {
```



```
}
```

Ritter Kunibert

```
Kunibert() {
```

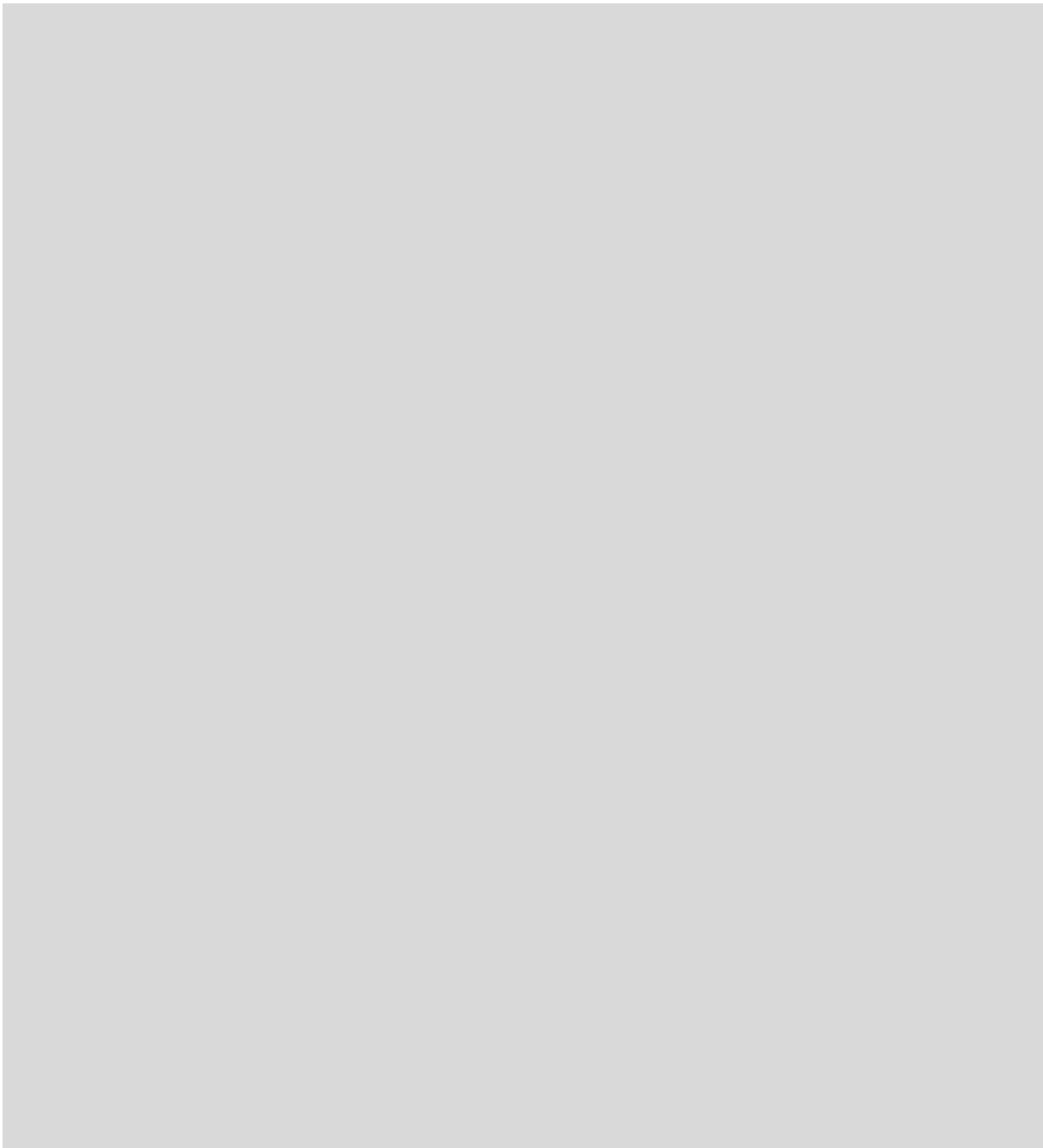




```
}
```

Knecht Ruprecht

```
Ruprecht() {
```

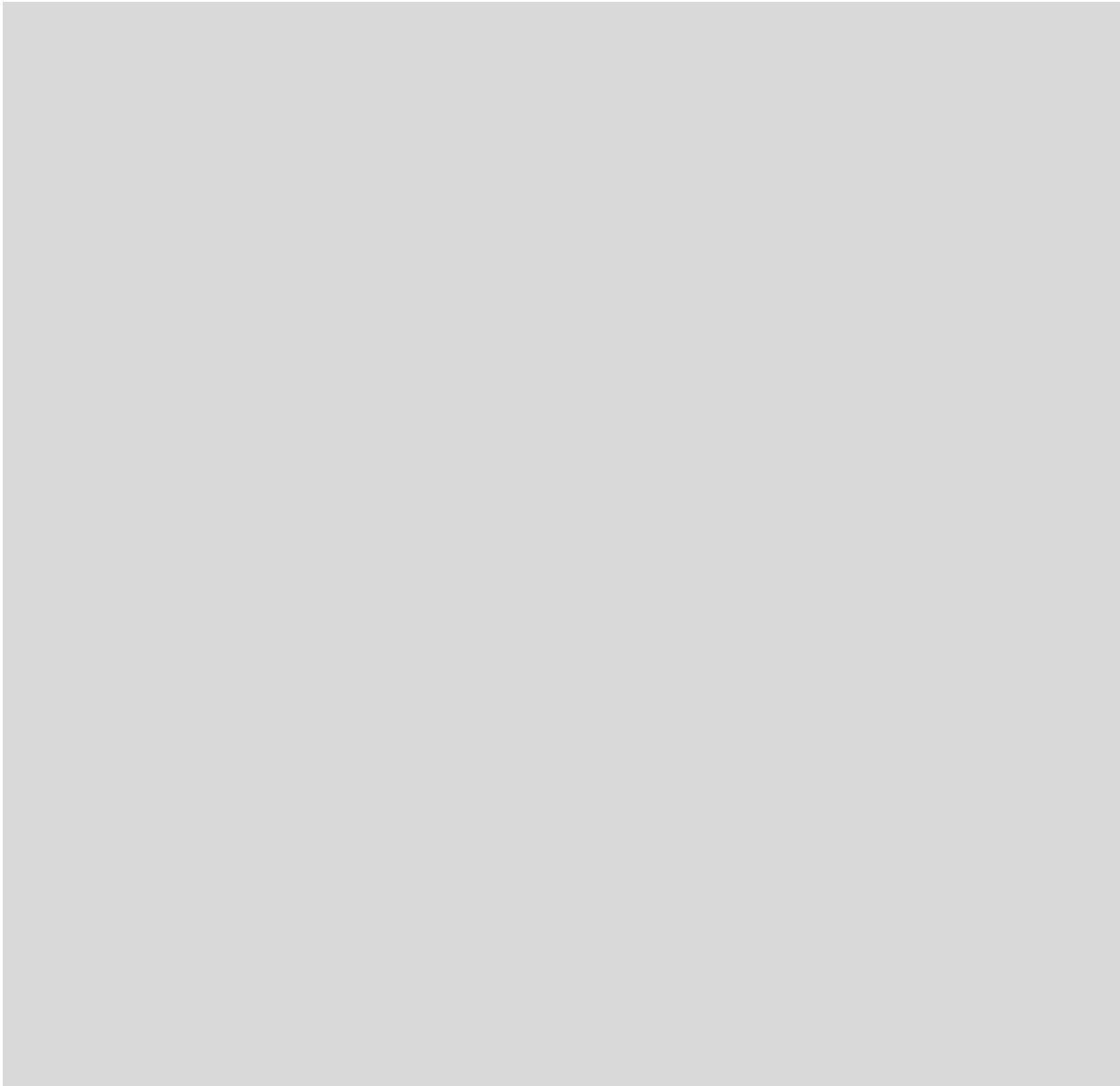




```
}
```

Knecht Vladimir

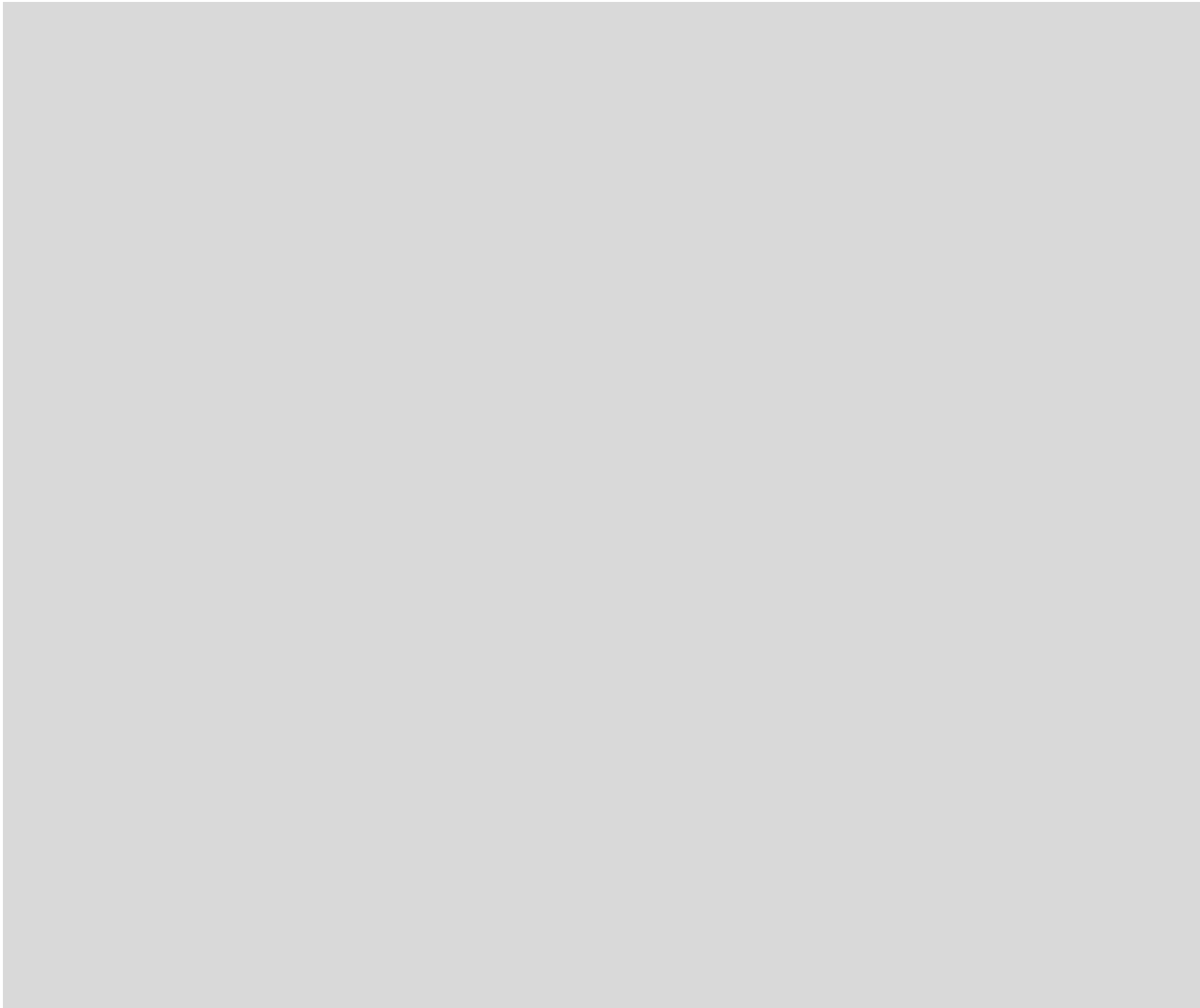
```
Vladimir() {
```



```
}
```

2.2 The Producer/Consumer Problem (5)

Welche Grundthematik behandelt das *Producer/Consumer* Problem:



3 Security (20)

a) (2)

Geben Sie mit Begründung an, ob aktive oder passive *Security Threats* schwieriger zu erkennen sind.

b) (3)

Was bedeutet das Prinzip *Least Priviledge*?

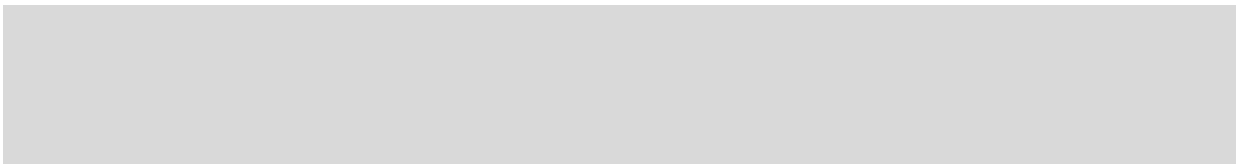
Nennen Sie ein Beispiel, wo dies verletzt wäre:

c) (6)

Beschreiben Sie das Prinzip des Public-Key Verschlüsselungsverfahrens, sowie die Vor- und Nachteile gegenüber symmetrischen Verschlüsselungsverfahren.

Prinzip:

Vorteile:



Nachteile:

d) (6)

Nennen Sie drei Methoden eines Hackers zum Herausfinden von Passwörtern:

Nennen Sie drei potentielle Schwachstellen des Passwortes “*password*”:

e) (3)

Was beschreibt das Modell von Bell und LaPadula?

4 Deadlock (25)

In einer Firma die sich mit Software und Web Design beschäftigt gibt es folgende Geräte: 4 Computer (C), 1 Scanner (S), 2 Drucker (D) und 3 Telefone (T).

In der Firma gibt es 4 Angestellte, einer ist zuständig für Web Design (WD), der zweite für Datenbanksoftware (DS) der dritte betreut die Kunden (KB), und der vierte ist System Administrator (*root*).

WD benötigt für seine Tätigkeit 3 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. DS benötigt für seine Tätigkeit 1 Computer, 1 Drucker, und 1 Telefon. KB benötigt für seine Tätigkeit 1 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. Root benötigt für seine Tätigkeit 1 Computer und 1 Telefon.

Zur Zeit belegt WD 2 Computer und 1 Telefon. DS belegt 1 Computer und 1 Telefon. KB belegt 1 Computer und 1 Drucker. Root belegt 1 Telefon.

a) (5)

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

		WD	DS	KB	root	
<i>Resource</i> =	$\begin{pmatrix} \text{C} \\ \text{S} \\ \text{D} \\ \text{T} \end{pmatrix}$	<i>Claim</i> =				$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$
<i>Available</i> =	$\begin{pmatrix} \text{C} \\ \text{S} \\ \text{D} \\ \text{T} \end{pmatrix}$	<i>Allocation</i> =				$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$

b) (12)

Suchen Sie jetzt eine Abarbeitungsreihenfolge für die Tätigkeiten der Angestellten, bei der alle Tätigkeiten ausgeführt werden können. Verwenden Sie dazu den *banker's algorithm* und geben Sie für *jeden* Schritt die Claim- und Allocationmatrix, sowie den Availability Vector und die als nächstes durchzuführende Tätigkeit (WD, DS, KB oder root) an. Wenn

keine Tätigkeit mehr auszuführen ist, dann schreiben sie 'fertig', falls ein Deadlock auftritt, schreiben Sie 'Deadlock' in den nächsten Schritt.

$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$



$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$



$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$



$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} & \text{bar} & \text{bar} & \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$



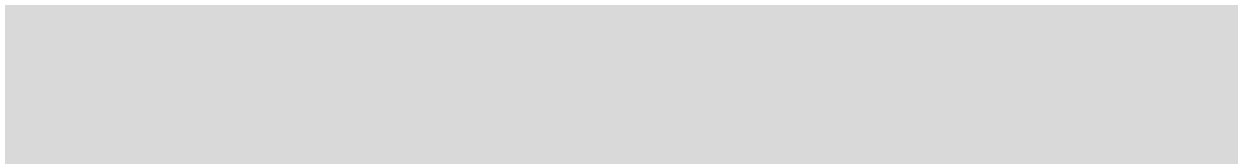
$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} \\ \text{bar} \\ \text{bar} \\ \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} \\ \text{bar} \\ \text{bar} \\ \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$



$$Claim = \begin{pmatrix} C \\ S \\ D \\ T \end{pmatrix} \begin{pmatrix} \text{bar} \\ \text{bar} \\ \text{bar} \\ \text{bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{bar} \\ \text{bar} \\ \text{bar} \\ \text{bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{bar} \end{pmatrix}$$

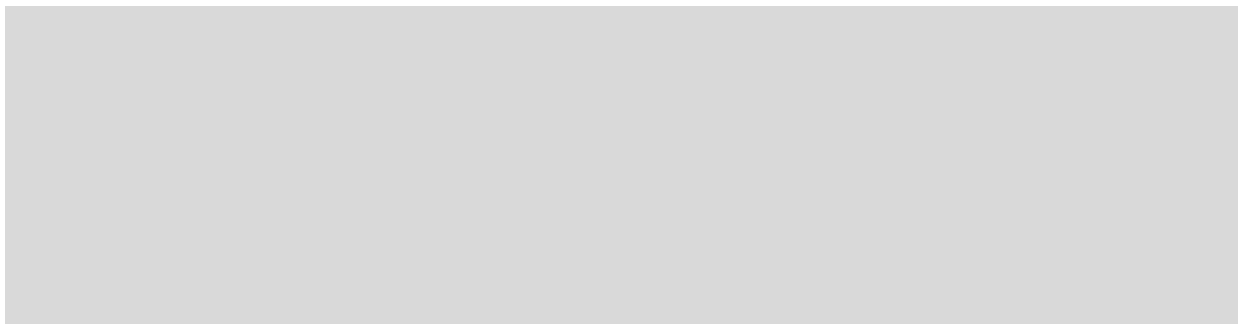
c) (3)

Was ist Deadlock?



d) (5)

Welche 4 Bedingungen müssen erfüllt werden dass es zu einen Deadlock kommt?



Prüfung Betriebssysteme

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(25)

2.)(25)

3.)(25)

4.)(25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronization (25)

In der Stadt *Peja* gibt es eine Bier-Fabrik. In der Fabrik will man den Bierabfüllungsprozess automatisieren, in dem man 4 Maschinen gekauft hat, die folgende Aufgaben (Funktionen) erfüllen:

Maschine 1: Bierflasche in die Produktionslinie platzieren - `flasche_platzieren()`
Bierflasche von der Produktionslinie entfernen - `flasche_entfernen()`

Maschine 2: Bierflasche mit Bier abfüllen - `flasche_abfüllen()`

Maschine 3: Bierflasche zumachen (mit Kronenkorken) - `flasche_zumachen()`

Maschine 4: Klebt Etikette an die Bierflasche - `etikette_kleben()`

Aufgrund der Ressourcenbeschränkungen kann jeweils nur 1 Flasche in die Produktionslinie platziert werden. Synchronisieren Sie die Arbeit der Maschinen effizient mittels Semaphore! Zusätzlich zu den Operatoren `P()` und `V()` stehen folgende Funktionen zur Verfügung:

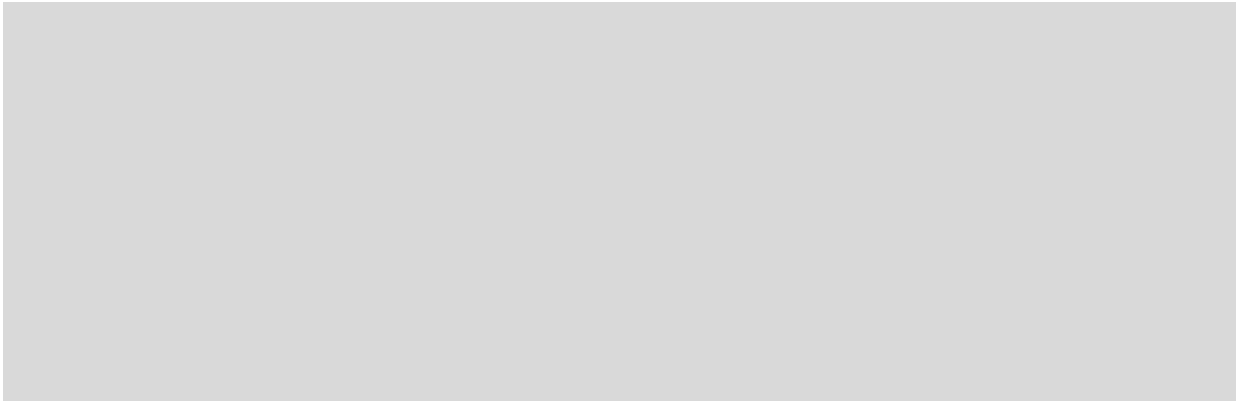
- `INIT(sem, value)`: erzeugt und initialisiert den Semaphor `sem` mit dem Wert `value`
- `finshed()`: evaluiert zu `TRUE` wenn die Tagesschicht beendet wird, oder aus anderen Gründen der Abfüllungsprozess gestoppt werden muss. Wenn das der Fall ist (`finshed()`=`TRUE`), dann soll die platzierte Flasche fertig abgefüllt, etikettiert, zugemacht und entfernt werden. Erst dann beenden die Maschinen ihre Arbeit.

Passen Sie auf, dass folgende Reihenfolge eingehalten wird:

Flasche platzieren -> Flasche abfüllen -> Flasche zumachen -> Flasche entfernen

Nachdem Flaschen platziert und bevor die Flaschen von der Produktionslinie entfernt werden, können die Etiketten geklebt werden. Achten Sie darauf, dass `flasche_abfüllen()` und `flasche_zumachen()` parallel zu der Funktion `etikette_kleben()` ausgeführt werden können. Um den Abfüllungsprozess zu optimieren, soll keine fixe Reihenfolge der Funktionen angegeben werden.

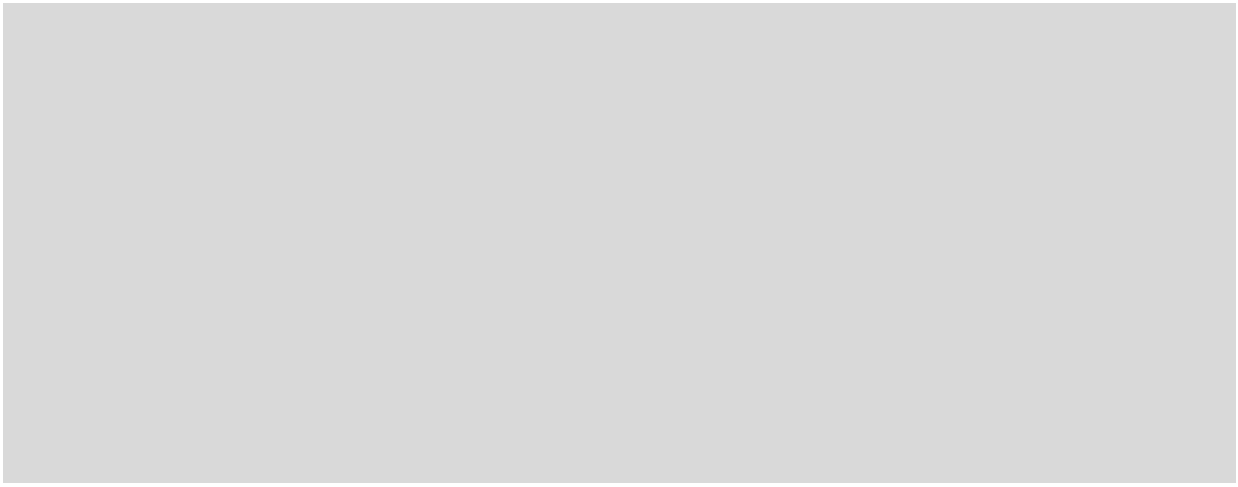
1.1 Initialisierung



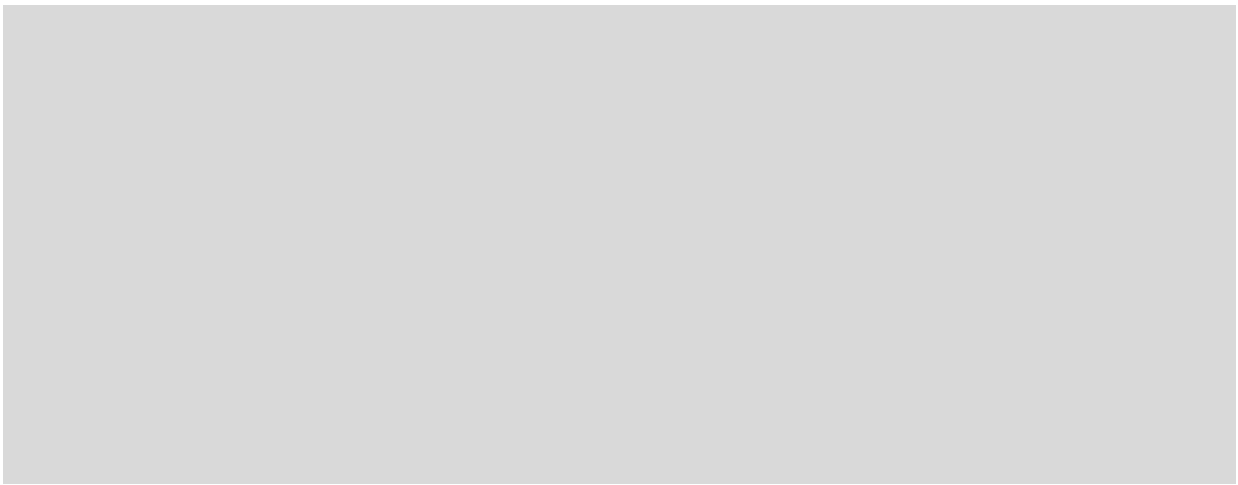
1.2 Maschine1_Flaschen



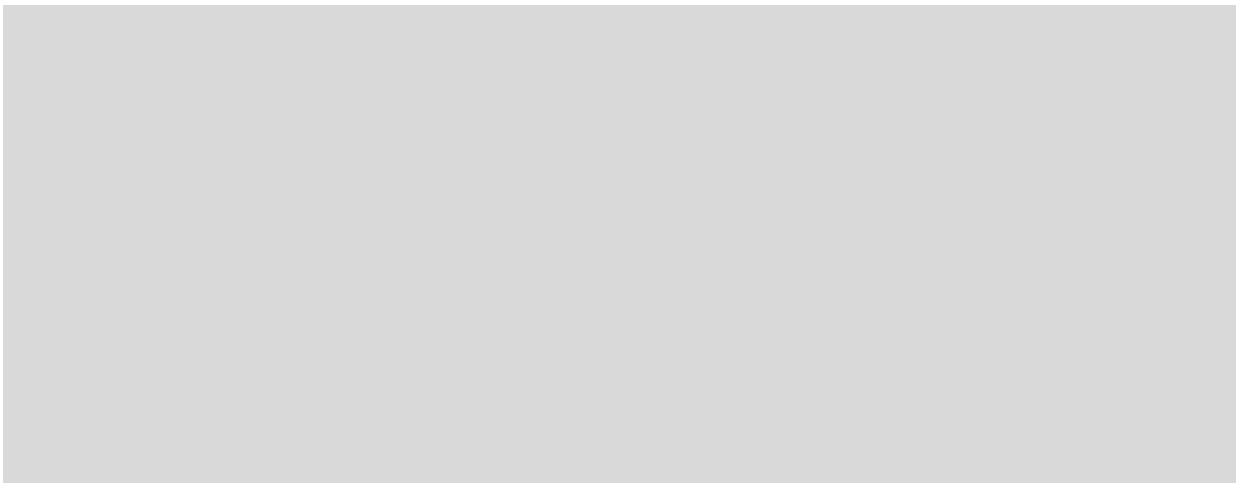
1.3 Maschine2_Bier



1.4 Maschine3_Kronenkorken



1.5 Maschine4_Etiketten



2 Scheduling (25)

2.1 Uniprocessor 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	1	5
B	2	9
C	2	7
D	1	4

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

A																	
B																	
C																	
D																	
	0				5					10					15		

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																	
B																	
C																	
D																	
	0				5					10				15			

Ersatzvorlage: Scheduling nach dem

-Verfahren: Erfolgreich: ☐ Ja ☐ Nein

A																	
B																	
C																	
D																	
	0				5				4	10				15			

2.2 Verständnisfragen (12)

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

- ☐ Ja ☐ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.
- ☐ 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, wenn eine solche existiert.
- ☐ Ja ☐ Nein Bei Round Robin Scheduling kann das Problem der Starvation 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 Eine für das EDF Scheduling hinreichende Bedingung stellt auch eine hinreichende Bedingung für das RMS Scheduling dar.
- ☐ Ja ☐ Nein Die First-Come-First-Serve-Strategie begünstigt Prozesse mit kurzer Ausführungszeit.
- ☐ Ja ☐ Nein Das Round-Robin-Verfahren ist preemptive.
- ☐ Ja ☐ Nein Beim Highest-Response-Ratio-Next-Verfahren kann das Problem der Starvation auftreten.
- ☐ Ja ☐ Nein Das Shortest-Remaining-Time-Verfahren ist immer preemptive.
- ☐ Ja ☐ Nein Das Shortest-Process-Next-Verfahren ist immer preemptive.
- ☐ Ja ☐ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die reziproke Periodendauer der Tasks.
- ☐ Ja ☐ Nein Ein Scheduler nach dem Feedback Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Shortest-Process-Next-Verfahren.
- ☐ Ja ☐ Nein Earliest Deadline First Scheduling ist optimal in Multi-Prozessor Systemen.
- ☐ Ja ☐ Nein Das Round-Robin-Verfahren ist optimal, wenn die Periodendauer aller Tasks gleich ist.
- ☐ Ja ☐ Nein Ein Scheduler nach dem RMS Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Round-Robin-Verfahren.

3 Deadlock (25)

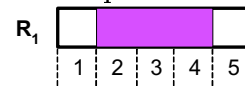
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehles.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Ressource 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 der Kantenzug einer (quasi)parallelen Abarbeitung 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.
5. Beschriften Sie jeweils ein Kästchen im Diagramm mit 'A', von welchem aus der Punkt *end* erreichbar ist
mit 'B', welches unweigerlich zu einen Deadlock führt
mit 'C', welches ein nicht erreichbarer Zustand ist

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

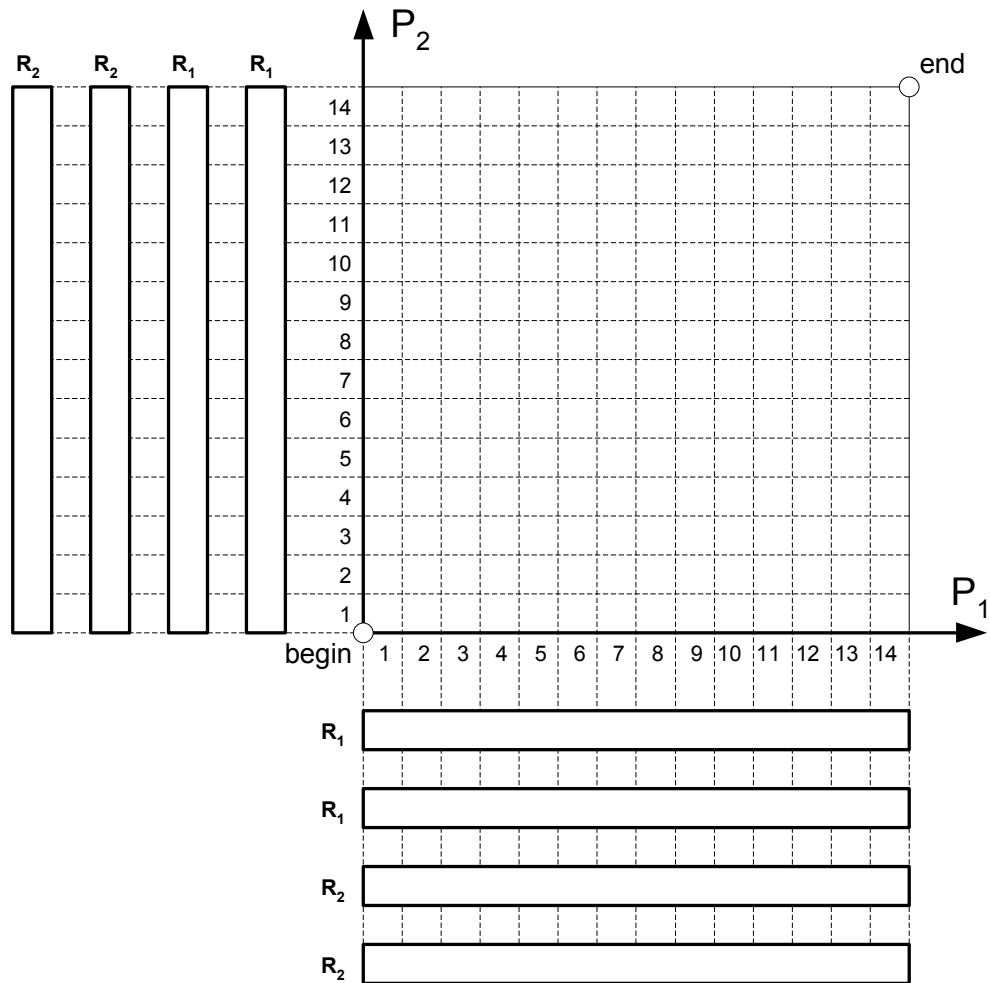
```

Program P_2 :

```

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

```



4 Memory Management (25)

Bei einem Einprogrammsystem (Single Task System) kann man den Hauptspeicher in zwei Bereiche einteilen, den Speicher für das Betriebssystem (Kernel) und für das gerade ausgeführte Programm. Bei einem Mehrprogrammsystem (Multi Task System) ist der Hauptspeicher in mehrere Bereiche unterteilt. Die Verwaltung dieser Bereiche übernimmt eine sogenannte Speicherverwaltung.

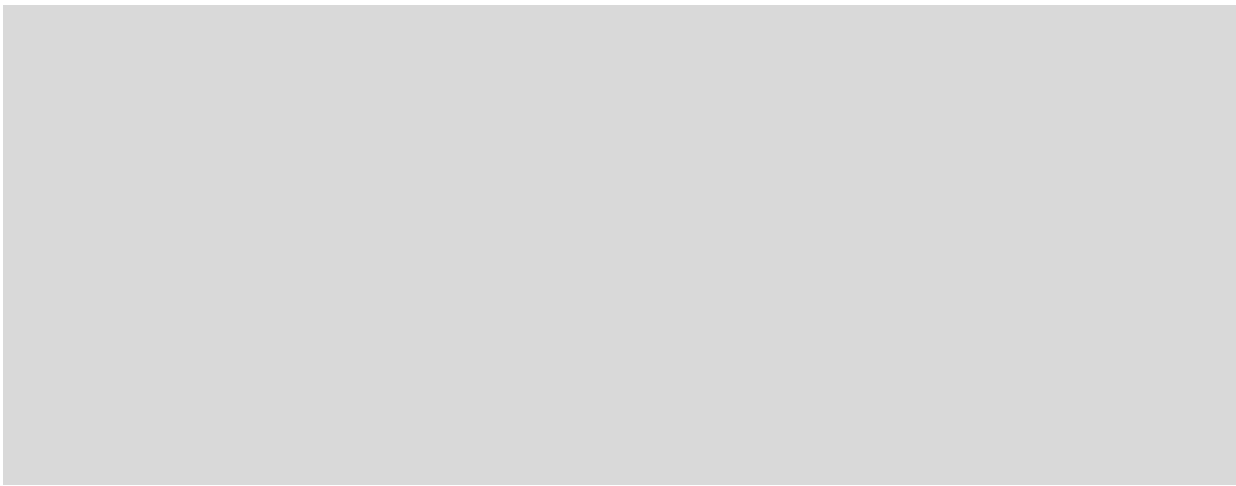
Welche typischen Anforderungen stellt man an eine solche Speicherverwaltung?(3 Punkte)

Erklären Sie den Begriff *virtueller Speicher*.

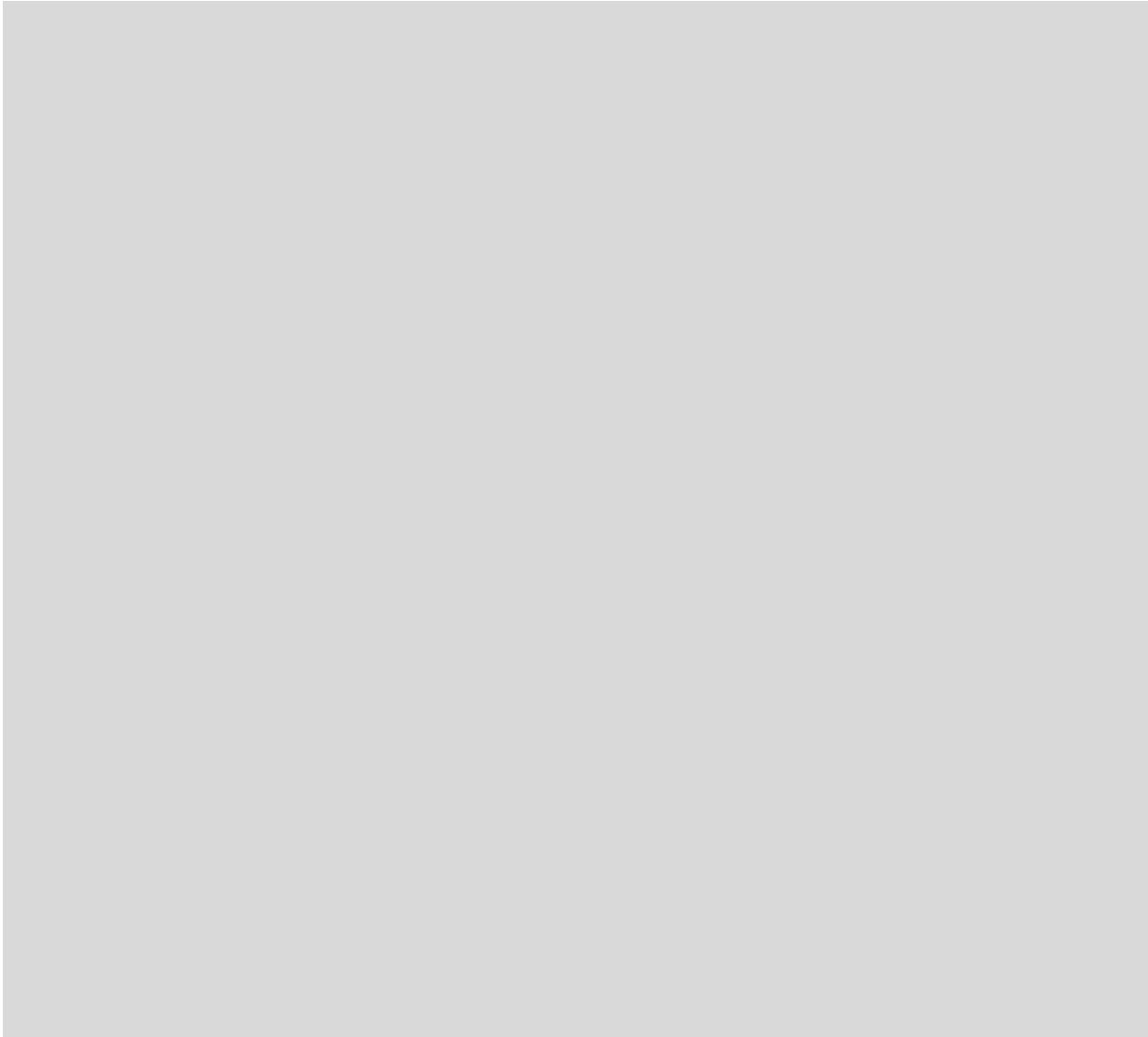
(3 Punkte)

Die zwei Grundtechniken um virtuellen Speicher zu realisieren sind *Segmentierung* und *Paging*. Erklären Sie die Gemeinsamkeiten, Unterschiede, Probleme, ... diese beiden Techniken. (8 Punkte)

Erklären Sie assoziative Zuordnung (associative mapping) und direkte Zuordnung (direct mapping). Geben Sie dazu jeweils ein Beispiel an. (5 Punkte)



Erklären Sie für ein einstufiges Paging-System die genauen Vorgänge bei einem TLB-Hit und TLB-Miss. Auf welche Speicherseiten wird zugegriffen, wie wird entschieden welcher Eintrag ausgewählt wird? (6 Punkte)



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

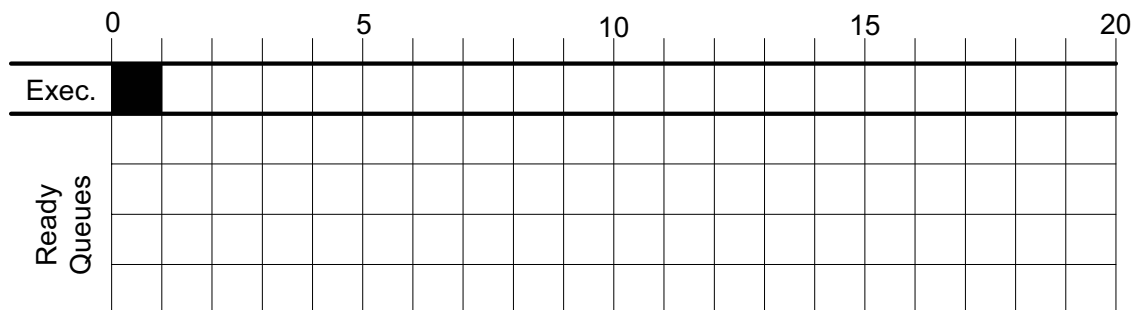
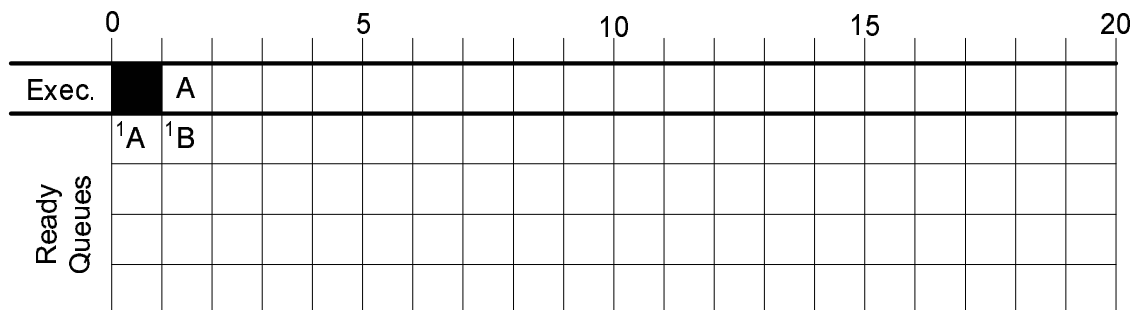
Schedulen Sie das nebenstehende Taskset. Beachten Sie dabei folgendes: Zu den angegebenen arrival times wird ein Task in die Ready Queue gestellt. Der Task kann frühestens beim nächsten diskreten Zeitpunkt dem Prozessor zugeteilt werden (Beispiel siehe Task A: arrival time= 0; Zuteilung = 1). Ein eintreffender Task wird immer ans Ende der Ready Queue gestellt (bei Feedback Scheduling immer ans Ende der höchst-prioren Queue).

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

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.

	0				5					10					15					20
Exec.																				
	A	B																		
Ready Queue																				

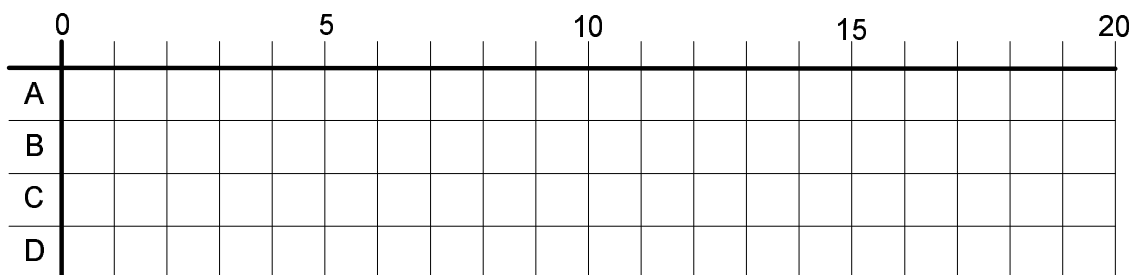
Abbildung 1: Round Robin Scheduling



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



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;
```

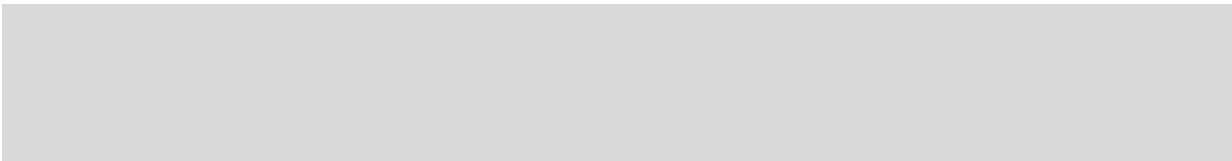
Reader

```
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;
```

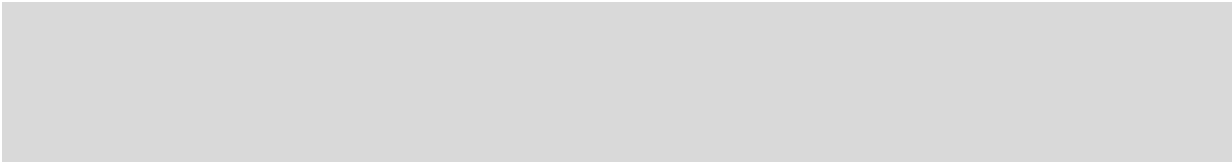
a) (8)

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:

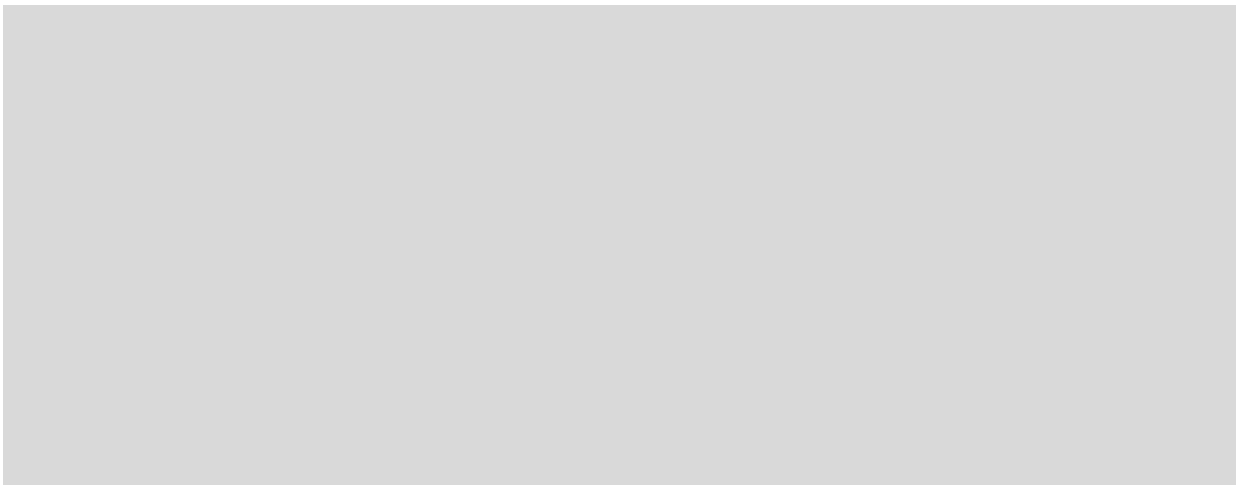
wsem:

b) (17)

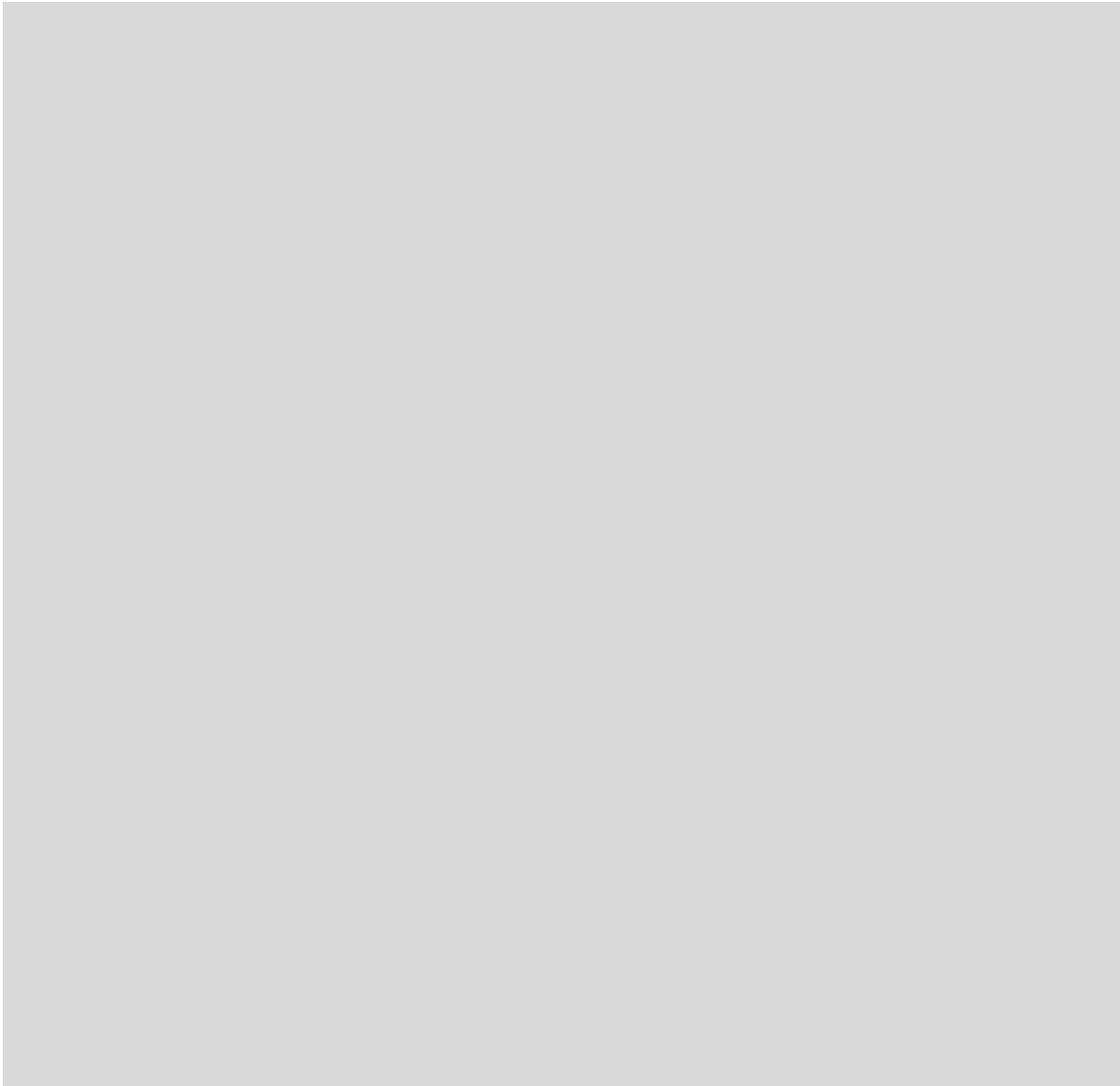
Wenn die maximale Anzahl von gleichzeitig aktiven *Reader*-Prozessen nach oben hin abgeschränkt wird, kann man eine Lösung für das *Reader-Writer Problem* angeben, die nur Semaphore (und keine globalen Variablen) zur Synchronisation verwendet. Schreiben Sie eine solche Lösung unter der Annahme, dass 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)

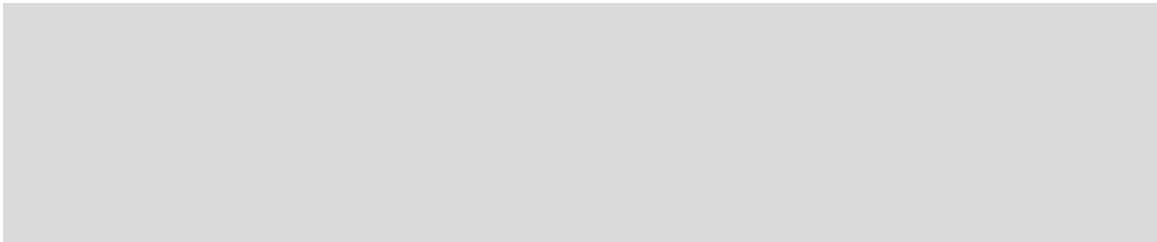
3.1 Begriffe (10)

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.

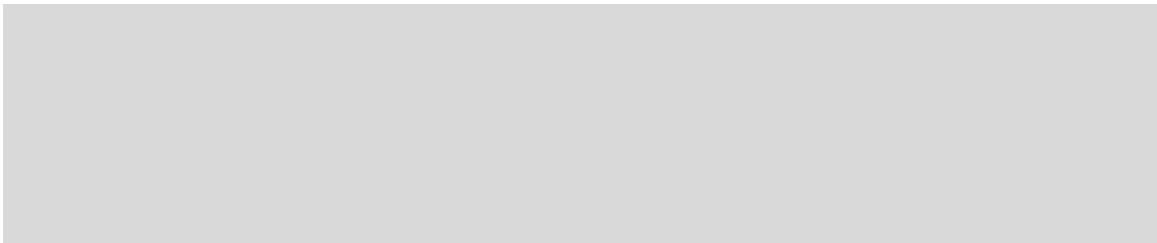
- Confidentiality:



- 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.

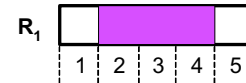
4 Deadlock (25)

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 Ressource 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	<input type="radio"/>	<input type="radio"/>
S_2	<input type="radio"/>	<input type="radio"/>
S_3	<input type="radio"/>	<input type="radio"/>
S_4	<input type="radio"/>	<input type="radio"/>
S_5	<input type="radio"/>	<input type="radio"/>
S_6	<input type="radio"/>	<input type="radio"/>

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;

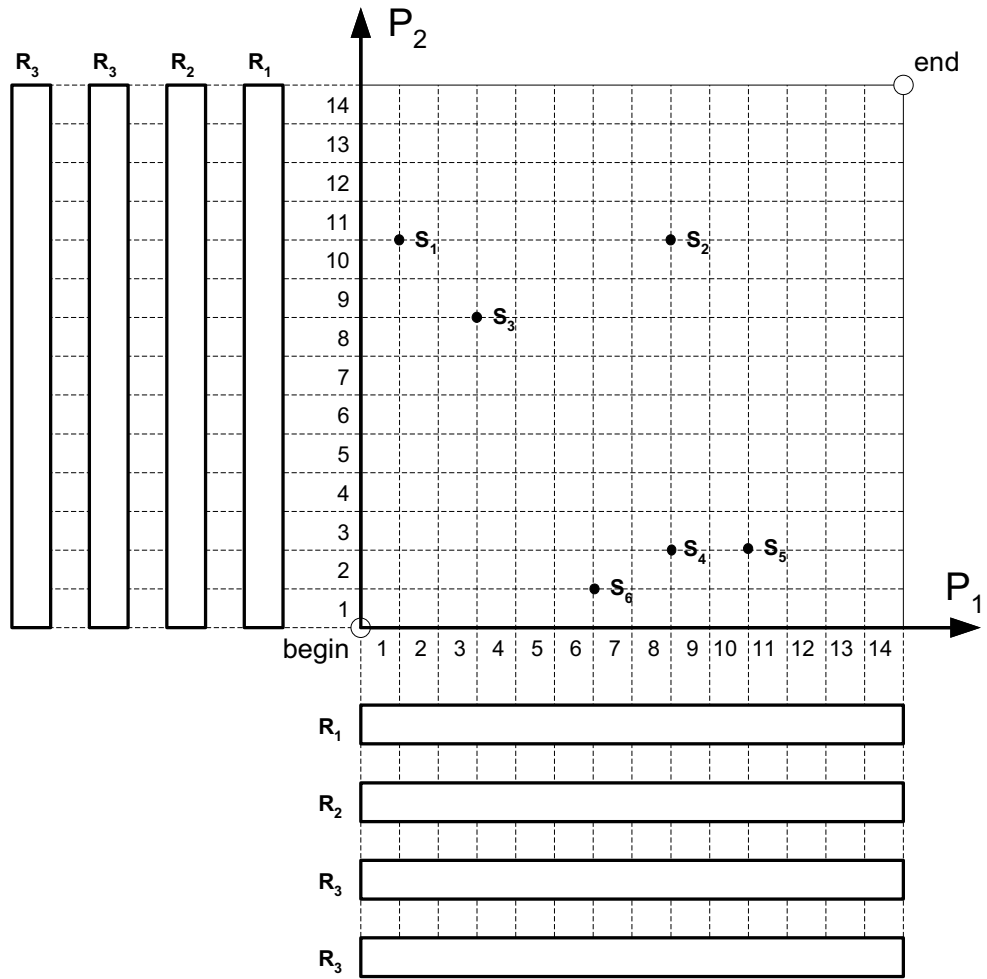
```

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;

```



KNr.

MNr.

Zuname, Vorname

Ges.) (100)

1.) (30)

2.) (25)

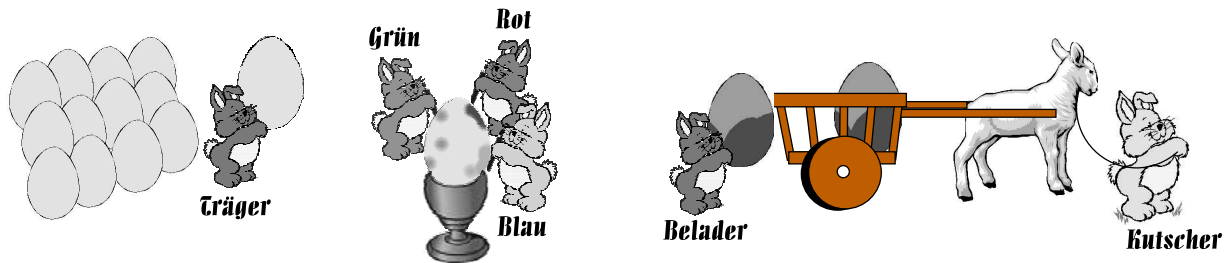
3.) (20)

4.) (25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Immer wenn Ostern vor der Tür steht, herrscht bei den Osterhasen Hochkonjunktur. Eine Abteilung von sechs Osterhasen mit Namen *Träger*, *Rot*, *Grün*, *Blau*, *Belader* und *Kutscher* ist mit der Herstellung und dem Versenden von bemalten Ostereiern beauftragt.

Die Hasen sollen dabei nach folgendem Schema arbeiten:

- *Träger* holt jeweils ein frisches Ei aus einem unbegrenzten Vorrat und platziert dieses im Eierbecher, wenn dieser leer ist.
- *Rot*, *Grün* und *Blau* bemalen die Eier im Eierbecher in den entsprechenden Farben. Das Bemalen eines Eis kann prinzipiell gleichzeitig durchgeführt werden, aus ästhetischen Gründen darf jedes Ei aber nur jeweils **zweifarb** bemalt werden. Es sollen dabei aber alle Farbkombinationen möglich sein. Welche beiden Farben dabei verwendet werden, darf zufällig sein.
- Wenn das Ei fertig bemalt ist, signalisieren *Rot*, *Grün* und *Blau* ihrem Kollegen *Belader*, dass das Ei abzuholen ist.
- *Belader* transportiert das bemalte Ei zum Wagen und belädt diesen.
- Ist der Wagen mit zwei Eiern voll, so signalisiert *Belader* dem *Kutscher* eine Lieferung zu machen.

- *Kutscher* liefert die Eier dann beim Kunden ab. Solange er unterwegs ist, dürfen keine neuen Eier auf den Wagen gelegt werden.

Synchronisieren Sie den Arbeitsablauf der sechs Hasen mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`gehezu(Ziel)` Bewegt den Hasen zum *Ziel*. Als Ziel kann *Vorrat*, *Eierbecher* oder *Wagen* angegeben werden.

`nimmEi()` Lässt den Hasen ein Ei aufnehmen. Funktioniert nur, wenn der Hase neben dem *Vorrat* bzw. neben dem *Eierbecher* steht.

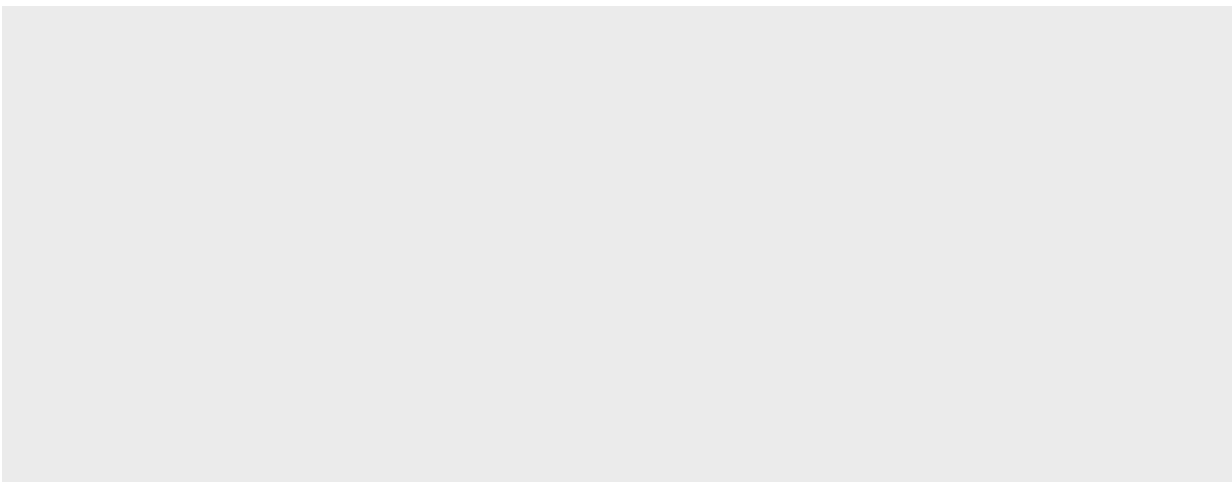
`bemaleEi()` Diese Funktion lässt einen Hasen das im *Eierbecher* befindliche Ei verzieren. Diese Funktion blockiert solange bis der Hase mit dem Bemalen in seiner Farbe fertig ist.

`legeEi()` Der Hase legt das Ei dort ab, wo er gerade steht. Befindet er sich vor dem *Wagen*, so belädt er diesen. Befindet sich der Hase vor dem *Eierbecher* so legt er das Ei in diesem ab. Diese Funktion beachtet nicht, für das Ei ausreichend Platz zur Verfügung steht!

`lieferung()` Mit dieser Funktion liefert *Kutscher* die Eier beim Kunden ab. Diese Funktion blockiert solange, bis der *Wagen* wieder leer an seinem Platz steht.

a) Initialisierungen (8)

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** des Systems entspricht obigem Bild, d.h. *Träger* steht mit einem Ei beim *Vorrat*, *Rot*, *Grün* und *Blau* schicken sich gerade an, ein Ei zu bemalen und *Belader* steht mit einem Ei vor dem *Wagen*, auf dem sich bereits ein weiteres Ei befindet.



b) (10)

Entwerfen Sie die Prozesse, die in den Hasen *Träger* sowie *Rot*, *Grün* und *Blau* ablaufen:

Prozess *Träger*:

```
do forever() {
```

Prozess *Rot|Grün|Blau*:

```
do forever() {
```

```
}
```

```
}
```

c) (12)

Entwerfen Sie die Prozesse, die in den Hasen *Belader* und *Kutscher* ablaufen:

Prozess *Belader*:

Prozess *Kutscher*:

```
do forever() {
```

```
do forever() {
```

```
}
```

```
}
```

2 Scheduling (25)

Priority Scheduling (8)

Schedulen Sie das nebenstehende Task Set. Beachten Sie dabei folgendes: Zu den angegebenen Arrival Times wird ein Task in die der Priorität des Tasks entsprechende Priority Queue gestellt. Jeder 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 entsprechenden Priority Queue gestellt. Abbildung 1 zeigt das Abarbeitungsschema.

Task	Arrival Time	Service Time	Priority
A	0	2	1
B	1	3	2
C	1	1	3
D	4	1	2
E	4	1	1
F	6	2	3
G	6	1	2
H	7	1	3
I	10	2	1
K	11	3	3

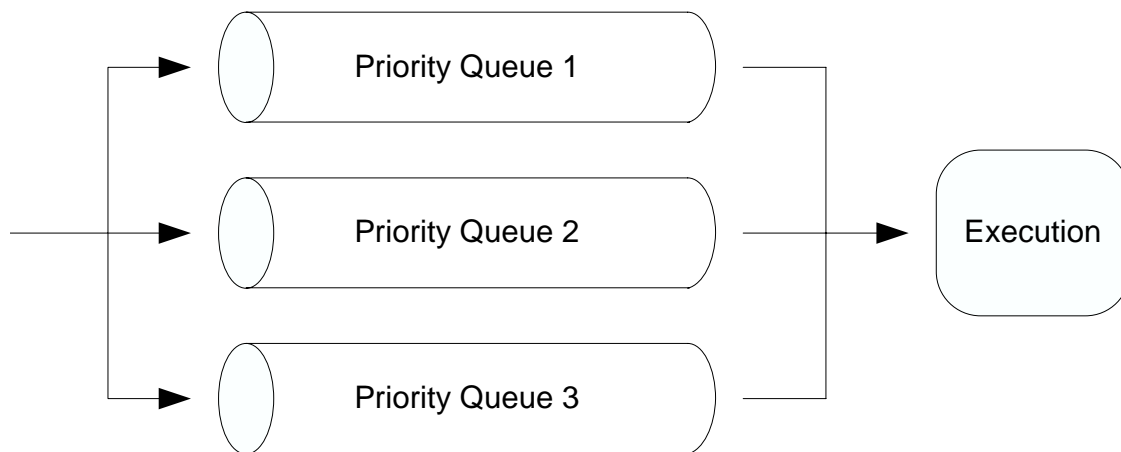
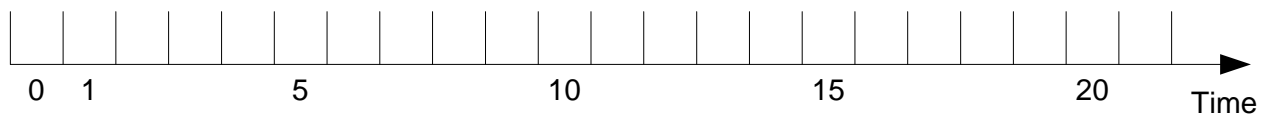


Abbildung 1: Priority Queuing

Beachten Sie, dass die Abarbeitung der Tasks *non preemptive* ist. Schreiben Sie in Abbildung 1 über jeder Priority Queue jene Tasks in zeitlich korrekter Reihenfolge, die dieser Queue zugewiesen werden. Tragen Sie weiters in das folgende Zeitschema die Abarbeitungsreihenfolge der Tasks ein. Der Overhead für den Taskwechsel ist vernachlässigbar.



Verständnisfrage (Eine fehlende Antwort wird negativ, eine falsche Antwort wird doppelt negativ gewertet!)

☐ Ja ☐ Nein Kann es beim Scheduling mit Priority Queues zur Starvation kommen?

Starvation (3)

Erklären Sie den Begriff *Starvation*:

Round-Robin Scheduling (7)

Schedulen Sie das nebenstehende Task Set. 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.

Task	Arrival Time	Service Time
A	0	3
B	5	5
C	7	2
D	10	4
E	15	2

Tragen Sie in der Zeile **P** 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 Ready Queue stehen (beginnend in der obersten Zeile mit dem Task, der als nächstes den Prozessor zugeteilt bekommt). Scheduling Sie das Task Set nach der Round-Robin Methode mit time-slice = 1. Der Overhead für den Taskwechsel ist vernachlässigbar.

	0				5					10					15					20
P		A																		
Ready Queue	A																			

Abbildung 2: Round Robin

Verständnisfragen (7)

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

- ☐ Ja ☐ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.
- ☐ Ja ☐ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die *Processor Utilization* der Tasks.
- ☐ 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 Bei Round Robin Scheduling kann das Problem der Starvation auftreten.
- ☐ Ja ☐ Nein Die First-Come-First-Serve-Strategie begünstigt Prozesse mit kurzer Ausführungszeit.
- ☐ Ja ☐ Nein Das Round-Robin-Verfahren ist preemptive.

3 Deadlock (20)

Ein Stahlhersteller produziert abhängig von der Nachfrage unterschiedliche legierte Stähle mit spezifischen chemischen Zusammensetzungen.

Um die laufenden Aufträge erfüllen zu können, müssen Ressourcen wie Container, Frachtschiffe und Güterzüge koordiniert werden.

Dem Stahlhersteller stehen 80 Container, 3 Frachtschiffe und 5 Züge zur Verfügung.

Zur Zeit sind drei Aufträge in Arbeit:

- Auftrag: # 1 beinhaltet den Transport von 1000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 10 Container, 1 Frachtschiff und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 10 Container und 2 Güterzüge reserviert.
- Auftrag: # 2 beinhaltet den Transport von 4000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 40 Container, 1 Frachtschiff und 4 Güterzüge notwendig. Die Bearbeitung dieses Auftrags hat noch nicht begonnen.
- Auftrag: # 3 beinhaltet den Transport von 1600 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 16 Container, 2 Frachtschiffe und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 16 Container und ein Frachtschiff reserviert.

Process Initiation Denial (10)

Verwenden Sie die Strategie des *Process Initiation Denial*.

Beschreiben Sie *Resource*- und *Available*-Vektor sowie *Claim*- und *Allocation*-Matrix. Die Matrizen sind so auszufüllen, dass die Prozesse (= Aufträge) zeilenweise und die Ressourcen spaltenweise aufgezählt werden.

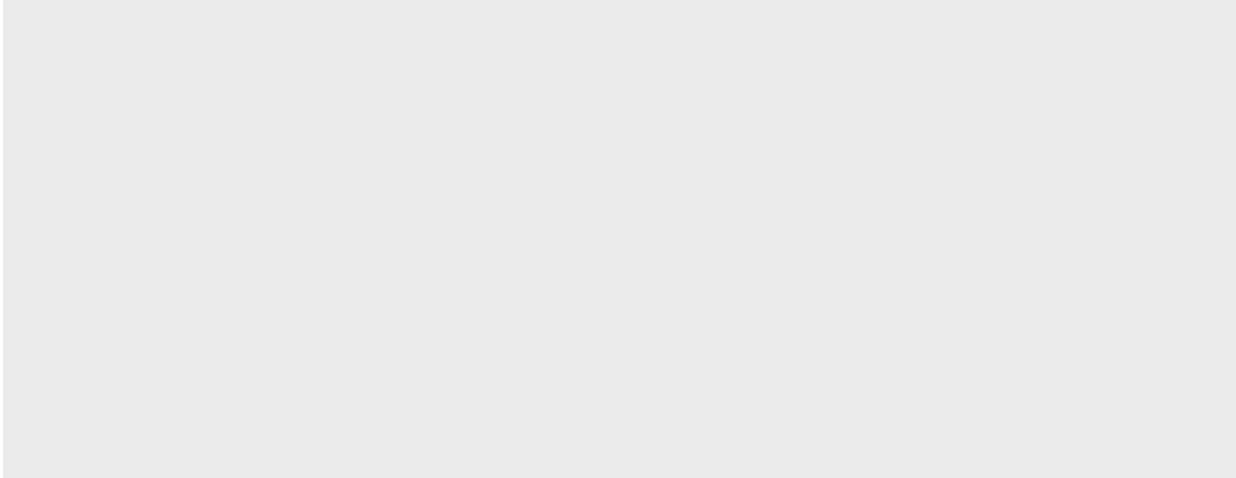
$$\begin{aligned} \textit{Resource} &= \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right) & \textit{Available} &= \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right) \\ \textit{Claim} &= \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \hline \end{array} \right) & \textit{Allocation} &= \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \hline \end{array} \right) \end{aligned}$$

Darf Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

☐ Ja

☐ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):



Resource Allocation Denial (10)

Verwenden Sie die Strategie des *Resource Allocation Denial*.

Beschreiben Sie für die gegebene Ausgangssituation mit Hilfe des Banker's-Algorithmus eine Lösung, bei der zuerst Auftrag 1, dann Auftrag 2 und zuletzt Auftrag 3 erfüllt werden.

Führen Sie alle Schritte bis zum Abschluss aller Aufträge durch und geben Sie zu jedem Schritt die zugehörige Claim- und Allocation-Matrix, sowie den Available-Vektor an.

Auftrag 1 wird durchgeführt:

(Alle für Auftrag 1 notwendigen Ressourcen sind in Verwendung):

$$C = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Alloc = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Avail = \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right)$$

Nachdem Auftrag 1 beendet ist und Auftrag 2 durchgeführt wird:

(Alle für Auftrag 2 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Alloc = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Avail = \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right)$$

Nachdem Auftrag 2 beendet ist und Auftrag 3 durchgeführt wird:

(Alle für Auftrag 3 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Alloc = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Avail = \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right)$$

Nachdem alle Aufträge durchgeführt worden sind:

$$C = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Alloc = \begin{pmatrix} \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} & Avail = \left(\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \end{array} \right)$$

4 Security (25)

Security Threats (6)

Beschreiben Sie die Security Threats? Geben Sie jeweils ein Beispiel an.

Data Oriented Access Control (3)

Beschreiben Sie die *Data Oriented Access Control*? Welche Elemente hat das Modell?

Design for Security (4)

Was bedeutet das Prinzip *Open Design*? Erklären Sie anhand eines Beispiels die Vorteile des Open Design Prinzips.

Verschlüsselung (5)

Beschreiben Sie das Prinzip des Public-Key Verschlüsselungsverfahrens, sowie die Vor- und Nachteile gegenüber symmetrischen Verschlüsselungsverfahren.

Program Related Threats (2)

Beschreiben Sie 4 Program Related Threats.

Reference Monitor (1)

Was ist ein Reference Monitor?

Intrusion Detection (4)

Was ist *Intrusion Detection*, und wozu wird es verwendet?

Beschreiben Sie 3 *Intrusion Detection* Methoden, und geben Sie jeweils ein Beispiel an.

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)
```

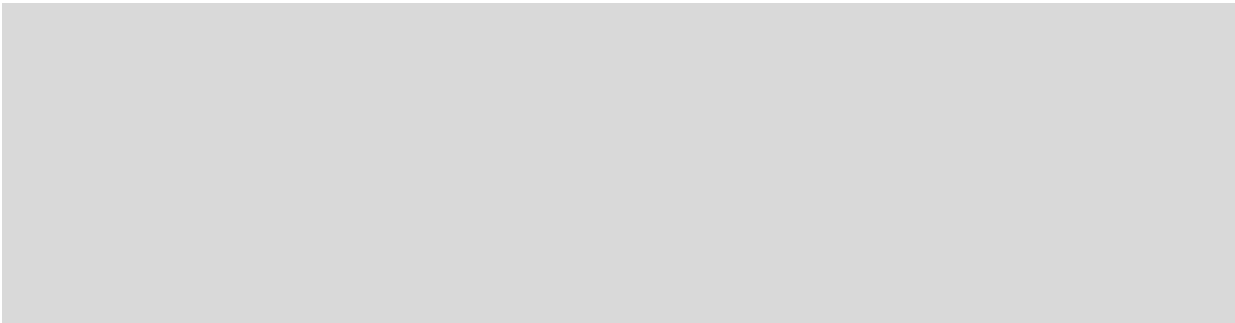
```
} else {   Behandlung fuer Schreibinstruktion (WRITE_INSTRUCTION)
```

```
    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.

	0	5	10	15	20
Exec.					
Ready Queue(s)					

Ersatzvorlage:

	0	5	10	15	20
Exec.					
Ready Queue(s)					

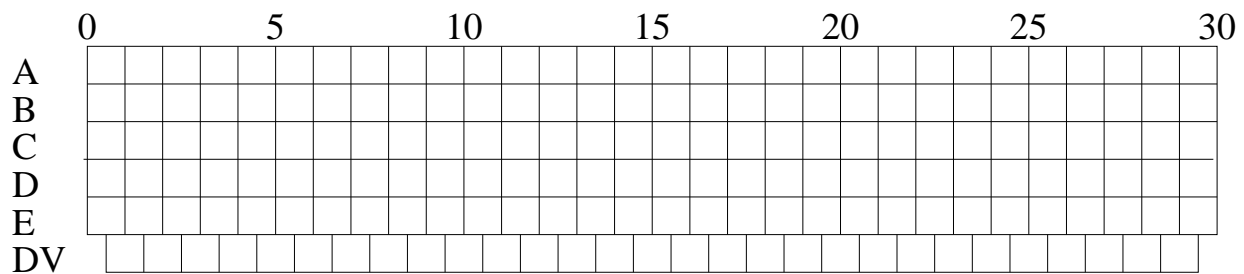
2.2 Earliest Deadline First Scheduling 12

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

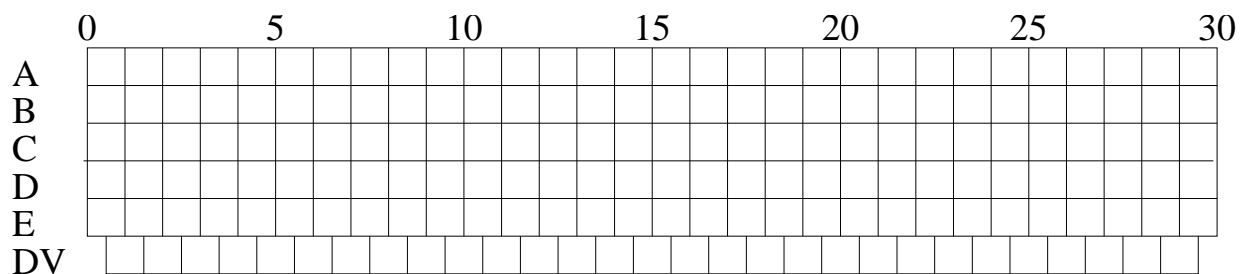
Task	Arrival Time (ms)	Execution Time (ms)	Deadline (ms)
A	0	10	20
B	2	3	10
C	3	4	9
D	6	1	7
E	16	3	19

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

Vervollständigen 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 Preemption 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.

$$Resource = \begin{pmatrix} I \\ T \\ P \end{pmatrix} \quad Claim = \begin{pmatrix} \\ \\ \end{pmatrix}$$

$$Available = \begin{pmatrix} I \\ T \\ P \end{pmatrix} \quad Allocation = \begin{pmatrix} \\ \\ \end{pmatrix}$$

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:

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

Nach der Ausführung des letzten Projektes:

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


Nächstes Projekt/fertig/Deadlock: 

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

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

Nach der Ausführung des letzten Projektes:

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

Nächstes Projekt/fertig/Deadlock: 

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

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

Nach der Ausführung des letzten Projektes:

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

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)

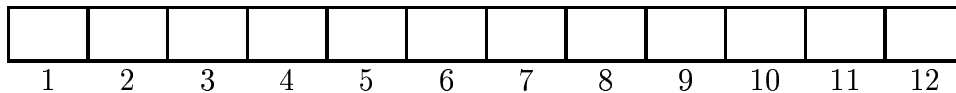


...Interne Fragmentierung (angegeben in % relativ zum allokierten Speicher)

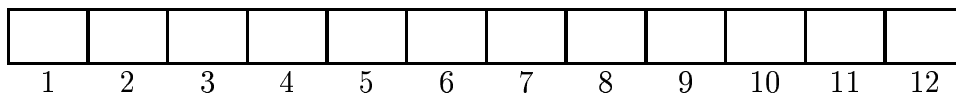


...Externe Fragmentierung (angegeben in % relativ zum Gesamtspeicher)

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

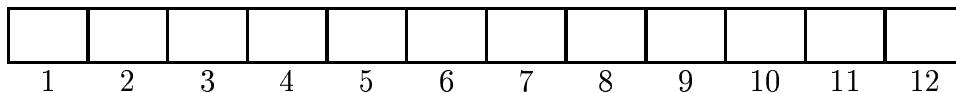


Ersatzvorlage:

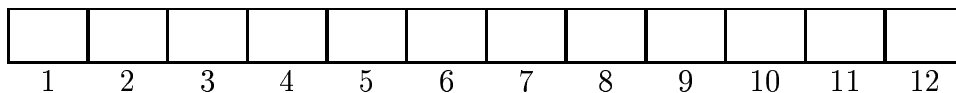


internal fragmentation: 33.3%, external fragmentation: 0%, Größe der

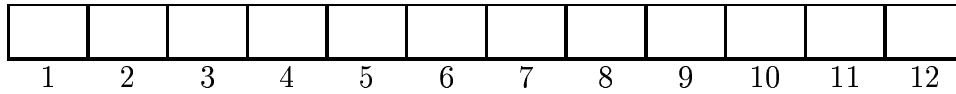
Allokationseinheit:  **Blöcke (frei wählbar)**



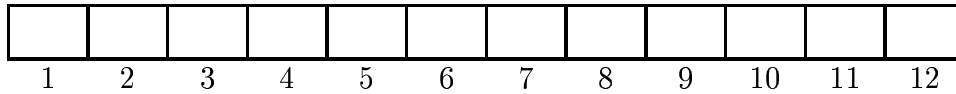
Ersatzvorlage:



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:

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)

1.1 Uniprocessor 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	1	5
B	2	10
C	2	7
D	1	6

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

A																	
B																	
C																	
D																	

0

5

10

15

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

A																	
B																	
C																	
D																	

0 5 10 15

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

A																	
B																	
C																	
D																	

0 5 10 15

1.2 Verständnisfragen (12)

Beurteilen Sie die folgenden Aussagen für Single-Prozessor Scheduling!

Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!

- ☐ Ja ☐ Nein Jedes Taskset, das mittels EDF gelöst werden kann, lässt auch mit RMS schedulen.
- ☐ Ja ☐ Nein Ein Taskset, das mittels EDF nicht gelöst werden kann, lässt eventuell mit RMS schedulen.
- ☐ Ja ☐ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.
- ☐ Ja ☐ Nein Bei Round Robin Scheduling kann das Problem der Starvation auftreten.
- ☐ Ja ☐ Nein Beim Scheduling nach dem Round-Robin-Verfahren werden I/O-intensive Prozesse benachteiligt.
- ☐ Ja ☐ Nein Wenn bei allen Tasks die Deadline gleich ihrer Periode ist, dann liefert RMS Scheduling dasselbe Ergebnis wie EDF Scheduling.
- ☐ Ja ☐ Nein Die First-Come-First-Serve-Strategie benachteiligt CPU-intensive Prozesse.
- ☐ Ja ☐ Nein Das Shortest-Remaining-Time-Verfahren begünstigt I/O-intensive Prozesse.
- ☐ Ja ☐ Nein Das Shortest-Process-Next-Verfahren ist für Echtzeitscheduling ungeeignet.
- ☐ Ja ☐ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die reziproke Periodendauer der Tasks.
- ☐ Ja ☐ Nein Das Shortest-Process-Next-Verfahren kommt im Vergleich zu Round Robin mit weniger Interruptaufrufen aus.



Synchronisieren Sie die Tätigkeiten der Mechaniker während eines Boxenstopps in der Formel 1. Ein Boxenstopp läuft folgendermaßen ab:

- Der Pilot fährt in die Boxengasse (`pit_stop()`). Ein Teammitglied (Signalgeber) gibt via der Signaltafel welche auf 'Brake on' gedreht ist, die Anweisung das Fahrzeug in der Box des Teams zum Stillstand zu bringen (`brake()`). Der Fahrer darf erst wieder Gas geben (`accelerate()`), wenn die Mechaniker fertig sind und die Signaltafel auf 'GO' gestellt ist. Damit ist der Boxenstopp beendet.
- Ein Teammitglied (Signalgeber) signalisiert dem Fahrer mit der Tafel 'Brake on/GO' (`signal_brake()` und `signal_go()`), ab wann das Fahrzeug wieder bereit ist, die Box zu verlassen. Der Signalgeber dreht das Signal auf 'GO', sobald das alle vier Reifen wieder Bodenkontakt haben.
- Um einen Reifenwechsel zu ermöglichen, muss das Auto angehoben werden. Diese Aufgabe übernehmen zwei Teammitglieder, sobald das Auto still steht und der Signalgeber dies mittels 'Break on' anzeigt. Einer bockt das Auto vorne (`lift_front()`) und einer hinten (`lift_back()`) auf. Erst dann können die Reifenmechaniker, die Tankwarte und der Visierreiniger mit ihrer Arbeit beginnen. Sobald diese fertig sind und die Hand gehoben haben, kann das Fahrzeug wieder abgesenkt werden (`down_front()` und `down_back()`) damit alle vier Reifen wieder Bodenkontakt haben.
- Insgesamt 12 Teammitglieder stehen bereit zum Reifenwechsel, je drei davon für ein Rad. Ein Mechaniker löst (`unscrew_wheel_nut()`) und befestigt die Radmutter (`fasten_wheel_nut()`) und hebt den Arm sobald er das Rad fertig gewechselt ist (`ready()`). Ein andere Mechaniker zieht das abgefahrene Rad ab (`remove_old_wheel()`) und der dritte steckt das neue Rad auf (`put_new_wheel()`).
- Zwei Teammitglieder sind für das Betanken des Wagens zuständig (`fill_up()`). Sobald das Fahrzeug fertig betankt ist, wird dies durch das Heben einer Hand signalisiert (`ready()`). Damit kann das Fahrzeug aus Sicht der Tankwarte wieder abgesenkt werden.

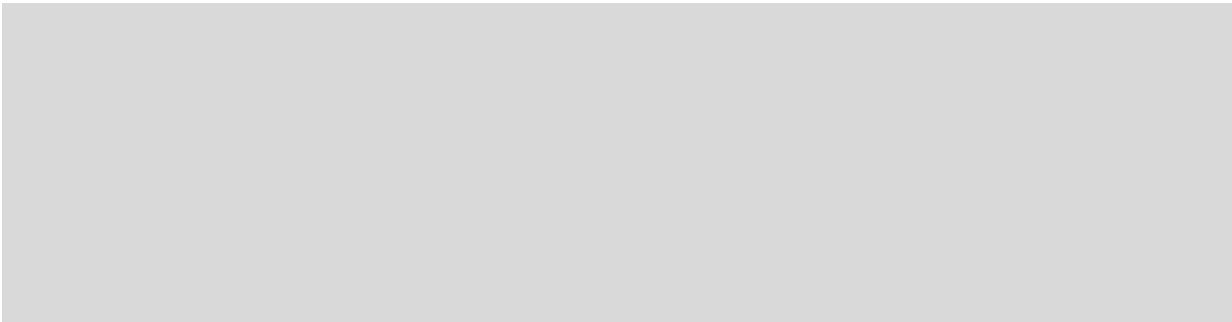
- Ein Teammitglied (Visierreiniger) säubert das Helmvisier des Piloten (`clean_visor()`). Sobald das Helmvisier gereinigt ist, wird dies durch das Heben einer Hand signalisiert (`ready()`). Damit kann das Fahrzeug aus Sicht des Visierreinigers wieder abgesenkt werden.

Ergänzen Sie den folgenden Code entsprechend. Beachten Sie dabei folgende Punkte

- Die Synchronisation hat durch Semaphore zu erfolgen, welche in der Funktion `Init()` zu initialisieren sind. Dafür steht die Funktion `initsem(semaphor,value)` zur Verfügung. Zur Synchronisation sind `P(semaphor)` und `V(semaphor)` zu verwenden. Verwenden Sie eine minimale Anzahl von Ressourcen!
- Achten Sie auf maximale Parallelität, um den Boxenstopp so schnell wie möglich zu beenden!

Initialisierung:

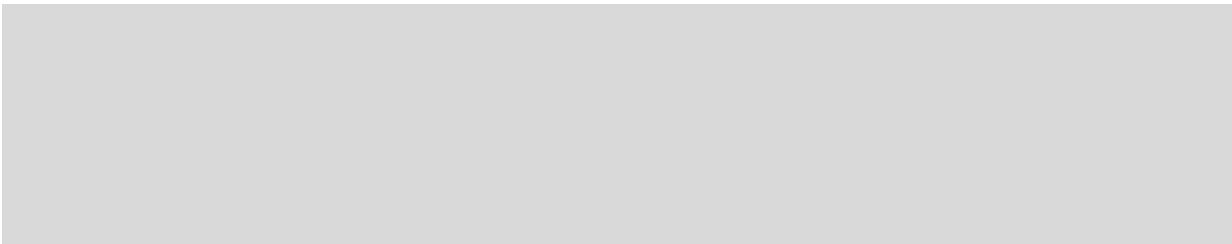
```
Init() {
    initsem(signal, 0);
```



```
}
```

Der Fahrer:

```
Driver() {
    pit_stop();
    break();
    V(signal);
```

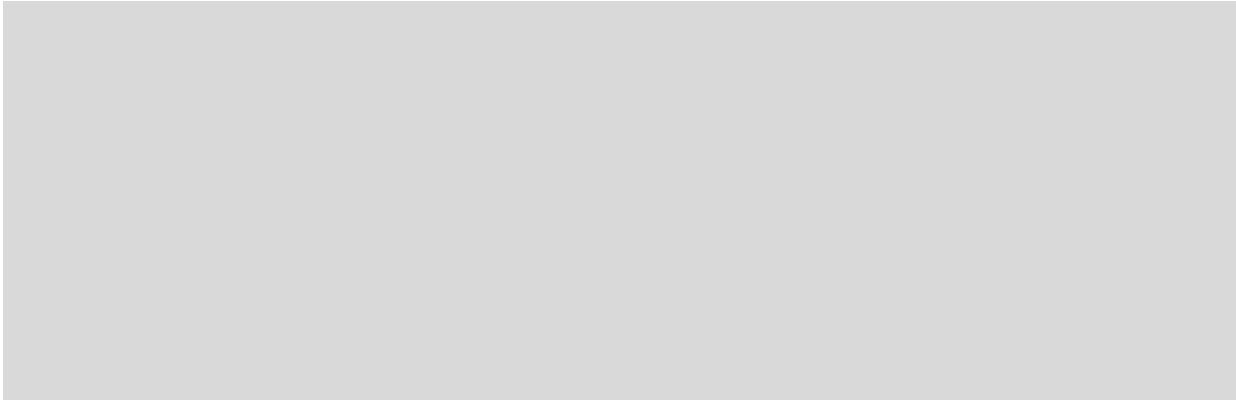


```
}
```

Mechaniker für den Reifenwechsel:

Obwohl diese Funktion viermal aufgerufen wird (für jedes Rad), braucht diese Funktion nur einmal programmiert werden. Diese Funktion implementiert die Tätigkeit von je drei Mechanikern pro Rad.

```
Mechanic_Wheel() {
```

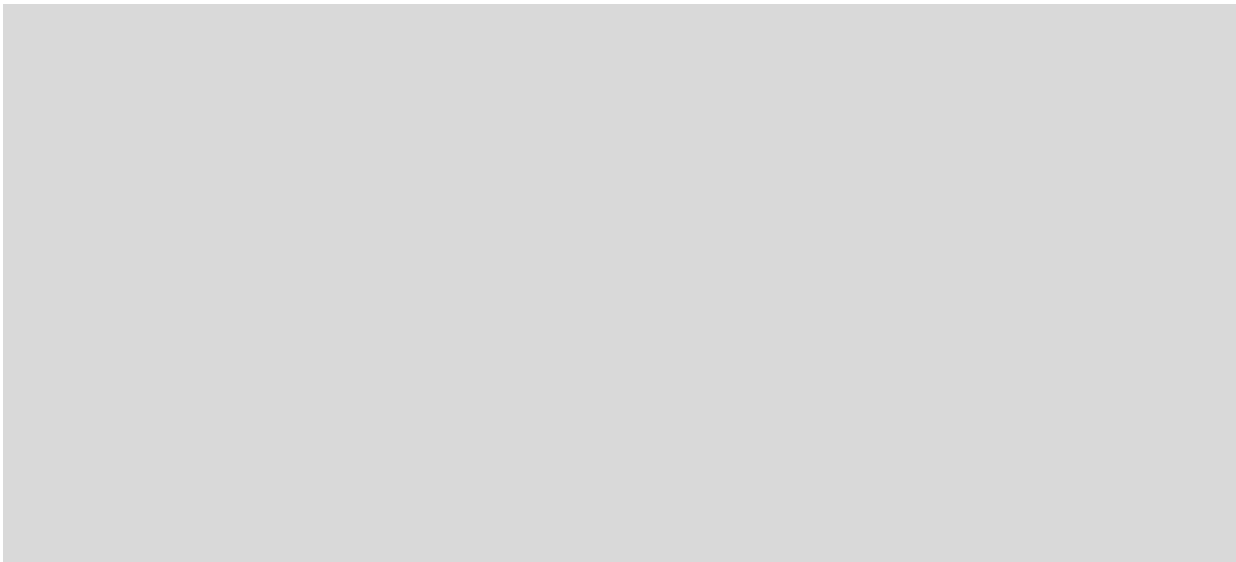


```
}
```

Mechaniker zum Aufbocken des Fahrzeugs:

Diese Funktion implementiert die Tätigkeit der zwei Mechaniker, welche das Auto aufbocken (vorne und hinten).

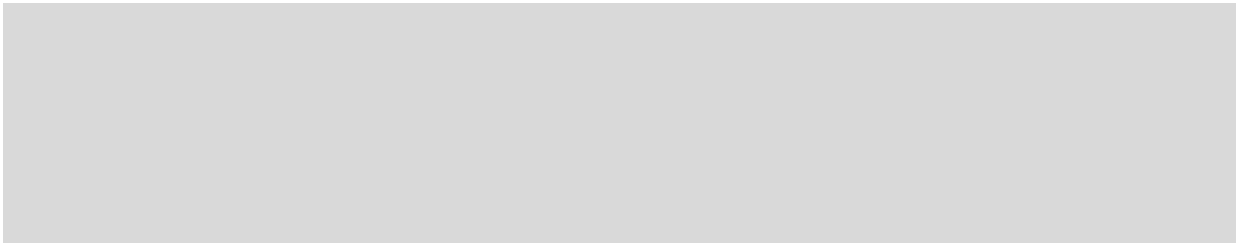
```
Mechanic_Lifter() {
```



```
}
```

Der Visierreiniger:

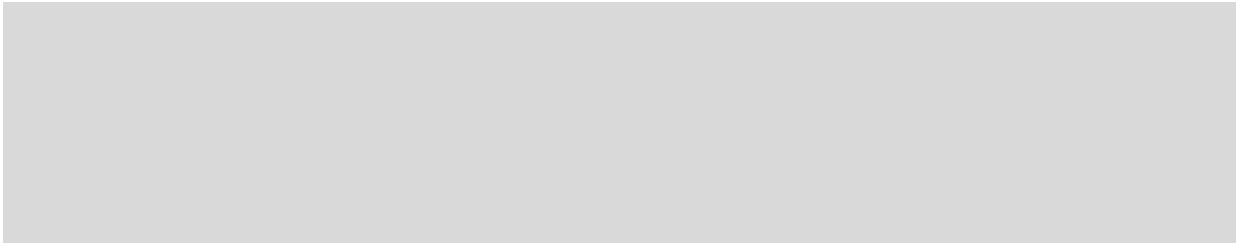
```
Visor() {
```



```
}
```

Betanken des Wagens:

```
Fuel() {
```



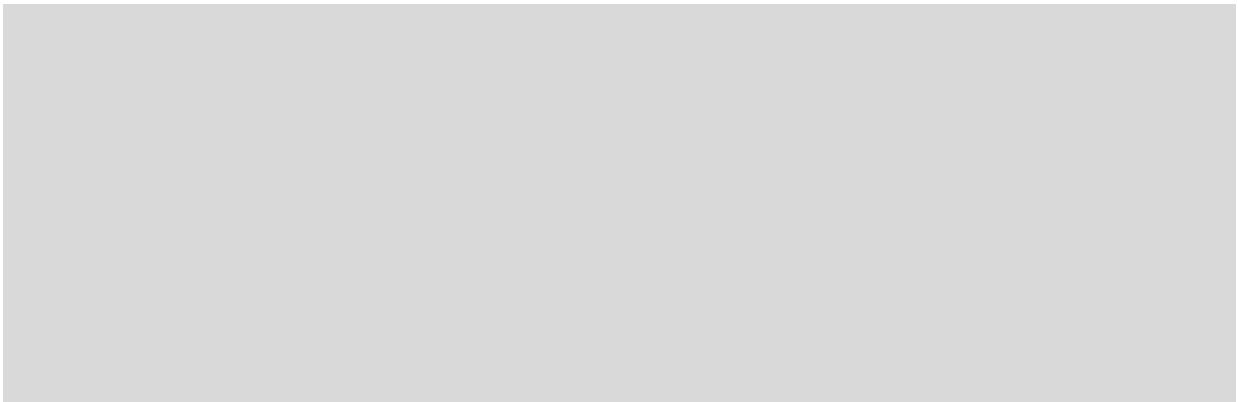
```
}
```

Der Signalgeber:

```
Signal() {
```

```
    signal_break(); //wait for standing car
```

```
    P(signal);
```



```
}
```

3 Security (25)

Security Threats (6)

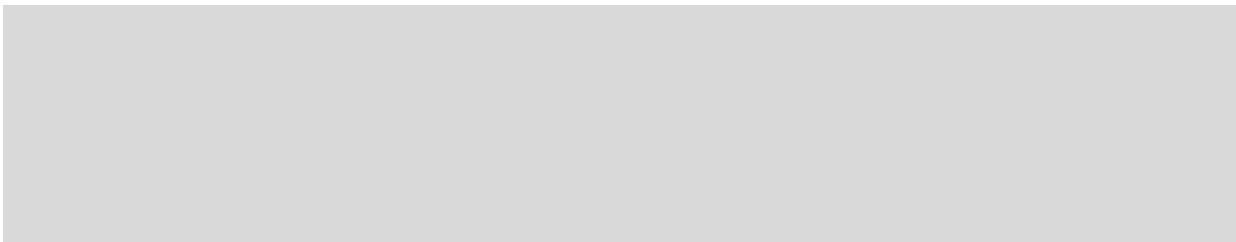
Beschreiben Sie die 3 grundsätzliche Arten von Security Threats? Geben Sie jeweils ein Beispiel an.



Verschlüsselung (4)

Beschreiben Sie das Prinzip des Public-Key Verschlüsselungsverfahrens, sowie die Vor- und Nachteile gegenüber symmetrischen Verschlüsselungsverfahren.

Prinzip:



Vorteile:

Nachteile:

Reference Monitor (2)

Was ist ein Reference Monitor?

Intrusion Detection (5)

Was ist *Intrusion Detection*, und wozu wird es verwendet?

Beschreiben Sie 3 *Intrusion Detection* Methoden, und geben Sie jeweils ein Beispiel an.

Authentication (2)

Was ist Authentication und wie wird das gemacht?

Verständnisfragen (6)

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

- ☐ Ja ☐ Nein *Network security* umfasst Maßnahmen zum Schutz der Daten während der Übertragung.
- ☐ Ja ☐ Nein Bei Asymmetrischen 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 keine Rechte per default hat.
- ☐ Ja ☐ Nein Bei *Masquerading* wird der Inhalt einer Message verändert.
- ☐ Ja ☐ Nein Eine *denial-of-service* Attacke ist eine aktive Attacke.

4 Deadlock (25)

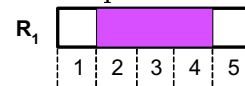
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehles.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Ressource 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)
```



- Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.
2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung 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.
 5. Beschriften Sie jeweils ein Kästchen im Diagramm
 - mit 'A', von welchem aus der Punkt *end* erreichbar ist
 - mit 'B', welches unweigerlich zu einen Deadlock führt
 - mit 'C', welches ein nicht erreichbarer Zustand ist

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

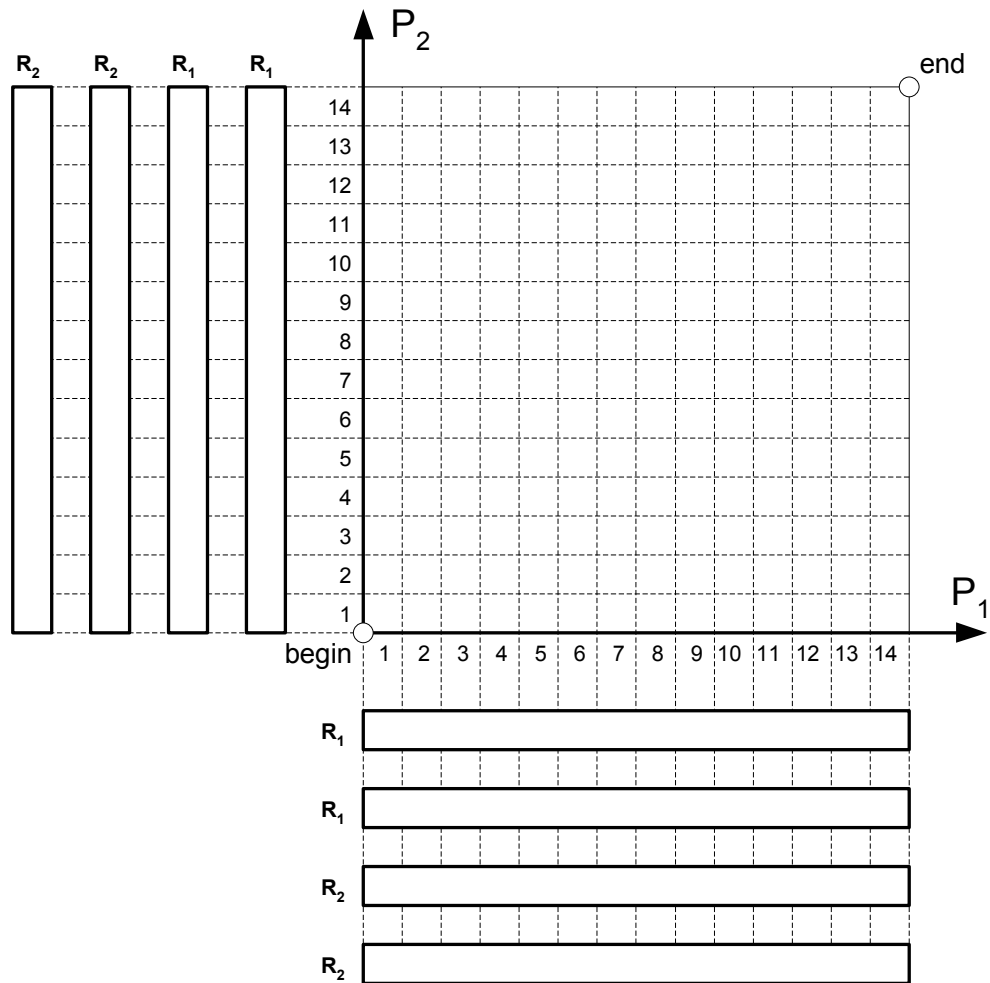
```

Program P_2 :

```

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

```



K.Nr.

M.Nr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(25)

3.)(25)

4.)(20)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

Beim Kartenspiel "Jag the Gitter" erhält jeder *Mitspieler* vom Kartengeber 5 Karten verdeckt vor sich auf den Tisch gelegt (Funktion `karte_geben(Spieler)`). Nachdem der Kartengeber (welcher selbst **nicht** am Spiel teilnimmt) die Karten **reihum einzeln** ausgeteilt hat, nehmen alle Spieler **gleichzeitig** ihre Karten auf (Funktion `karten_nehmen()`).



Nachdem alle Mitspieler ihre Karten aufgenommen haben, wird rundenweise jeweils eine Karte pro Mitspieler (Funktion `karte_spielen()`) gespielt. Dabei beginnt jeweils Spieler 1 mit dem Ausspielen, dann kommt Spieler 2 dran usw. Nachdem alle Karten ausgespielt wurden ist das Spiel beendet und der Kartengeber teilt erneut Karten für die nächste Runde aus.

Zu verwendende Funktionen (alle Funktionen blockieren so lange, bis der angegebene Task erledigt ist):

`karte_geben(Spieler)` teilt eine Karte an den angegebenen *Spieler* ($\in \{1, 2, 3, \dots\}$) aus. Diese Funktion wird nur vom Kartengeber verwendet.

`karten_nehmen()` lässt den Spieler alle seine Karten aufnehmen. Diese Funktion darf erst aufgerufen werden, wenn alle Karten dieses Spielers ausgeteilt sind.

`karte_spielen()` lässt den Spieler eine seiner Karten ausspielen.

`initS(semname, n)`, `initE(evname, n)` zum Anlegen und Initialisieren von Semaphoren und Eventcountern

`P()`, `V()`, `advance()`, `wait()` mit der üblichen Semantik für Semaphore und Eventcounter.

Weiters zu beachten:

- Die Verwendung von globalen Variablen zur Synchronisation ist verboten!
- Achten Sie auf maximale Parallelität!
- Verwenden Sie (unter Berücksichtigung der oberen beiden Punkte) nur so viele Ressourcen wie notwendig!

a) (14)

Zuerst soll das Kartenspiel für *2 Spieler* implementiert werden. Synchronisieren Sie die folgenden Prozesse (Kartengeber, Spieler 1 und Spieler 2) mit **Semaphoren**, sodass Sie das Spiel “Jag the Gitter” wie oben beschrieben spielen.

Initialisierungen		
Kartengeber	Spieler 1	Spieler 2
<div></div> <div>FOREVER() {</div> <div></div> <div>}</div>	<div></div> <div>FOREVER() {</div> <div></div> <div>}</div>	<div></div> <div>FOREVER() {</div> <div></div> <div>}</div>

c) (16)

Nun soll das Kartenspiel auf eine beliebige Anzahl N von Spielern erweitert werden (N ist konstant). Ergänzen Sie im folgenden Gerüst die Aufrufe von `await()`, um die Prozesse (Kartengeber, Spieler i) mit **einem Eventcounter** derartig zu synchronisieren, sodaß Sie das Spiel “Jag the Gitter” wie oben beschrieben spielen. Im Spielerprozess steht Ihnen die entsprechende Nummer des Spielers (zwischen 1 und N) in der Variablen i zur Verfügung.

Tipp: Sie brauchen zur Lösung dieses Beispiel *keine* Sequencer!

Initialisierungen	
Kartengeber	Spieler i
<div style="background-color: #f0f0f0; height: 20px; margin-bottom: 10px;"></div> <pre>FOREVER() {</pre> <div style="background-color: #f0f0f0; height: 400px; margin-top: 10px;"></div> <pre>}</pre>	<div style="background-color: #f0f0f0; height: 20px; margin-bottom: 10px;"></div> <pre>FOREVER() {</pre> <div style="background-color: #f0f0f0; height: 400px; margin-top: 10px;"></div> <pre>}</pre>

dem Task der als nächstes den Prozessor zugeteilt bekommt. Scheduling Sie das Task Set nach der Round-Robin Methode (time-slice = 1). Der Overhead für den Taskwechsel ist vernachlässigbar.

	0	5	10	15	20
A					
B					
C					
D					
E					

Abbildung 2: Round Robin Scheduling

Verständnisfragen (8)

Was bedeutet Starvation?

Geben Sie 3 Scheduling Methoden an bei denen es zu Starvation kommen kann:

Erklären Sie das **gemeinsame** Problem dieser 3 Methoden, wieso es zu Starvation kommen kann:

3 Deadlock (25)

Ein Computer-Server besitzt 5 Festplatten, 2 Speichermodule mit je 10 GByte und 3 Gigabit Ethernet Netzwerkkarten. Die Ressourcen, die das installierte Betriebssystem benötigt, brauchen Sie zur Lösung der folgenden Aufgaben nicht berücksichtigen.

Um die auf dem Server befindlichen Programme schedulen zu können, müssen die Ressourcen wie Festplatten (F), Speicher (S) und Netzwerkkarten (N) koordiniert werden.

Zurzeit befinden sich drei Programme am Server:

- **Programm 1** benötigt 4 Festplatten, 20 GByte Speicher und 2 Netzwerkkarten. Für dieses Programm ist bereits 1 Festplatte reserviert.
- **Programm 2** benötigt 4 Festplatten, 20 GByte Speicher und keine Netzwerkkarte. Für dieses Programm sind bereits 1 Festplatte und 10 GByte Speicher reserviert.
- **Programm 3** benötigt 2 Festplatten, 10 GByte Speicher und 3 Netzwerkkarten. Für dieses Programm sind bereits 1 Festplatte, 10 GByte Speicher und 1 Netzwerkkarte reserviert.

Process Initiation Denial (10)

Beschreiben Sie *Resource*- und *Available*-Vektor sowie *Claim*- und *Allocation*-Matrix. Die Matrizen sind so auszufüllen, dass die Ressourcen zeilenweise und die Programme (Prozesse) spaltenweise aufgezählt werden.

$$\begin{aligned} \text{Resource} &= \begin{pmatrix} \text{F} & \text{ } \\ \text{S} & \text{ } \\ \text{N} & \text{ } \end{pmatrix} & \text{Available} &= \begin{pmatrix} \text{ } \\ \text{ } \\ \text{ } \end{pmatrix} \\ \text{Claim} &= \begin{pmatrix} \text{F} & \text{ } & \text{ } \\ \text{S} & \text{ } & \text{ } \\ \text{N} & \text{ } & \text{ } \end{pmatrix} & \text{Allocation} &= \begin{pmatrix} \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \end{pmatrix} \end{aligned}$$

Darf mit Programm 2 gemäß *Process Initiation Denial* begonnen werden?

☐ Ja

☐ Nein


Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Bedingungen für Deadlock (5)

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

Deadlock Vermeidung (10)

Verwenden Sie nun von obigem Initialzustand ausgehend den *Bankier-Algorithmus* (Banker's Algorithm) zum Scheduling der Prozesse. Geben Sie für *jeden* Schritt die Claim- und Allocation-Matrix, sowie den Availability Vector und das als nächstes durchzuführende Programm an. Wenn kein Programm mehr auszuführen ist, dann schreiben sie 'fertig', falls ein Deadlock auftritt, schreiben Sie 'Deadlock' in den nächsten Schritt.


Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in

Verwendung:

$$Claim = \begin{pmatrix} F & \text{gray bar} & \text{gray bar} & \text{gray bar} \\ S & & & \\ N & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{gray bar} & \text{gray bar} & \text{gray bar} \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \text{gray bar} \\ & \\ & \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} F & \text{gray bar} & \text{gray bar} & \text{gray bar} \\ S & & & \\ N & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{gray bar} & \text{gray bar} & \text{gray bar} \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \text{gray bar} \\ & \\ & \end{pmatrix}$$


Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in

Verwendung:

$$Claim = \begin{pmatrix} F & \text{gray bar} & \text{gray bar} & \text{gray bar} \\ S & & & \\ N & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{gray bar} & \text{gray bar} & \text{gray bar} \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \text{gray bar} \\ & \\ & \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} F & \text{gray bar} & \text{gray bar} & \text{gray bar} \\ S & & & \\ N & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{gray bar} & \text{gray bar} & \text{gray bar} \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \text{gray bar} \\ & \\ & \end{pmatrix}$$

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in

Verwendung:

$$Claim = \begin{pmatrix} F \\ S \\ N \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} F \\ S \\ N \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} F \\ S \\ N \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} F \\ S \\ N \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

4 Security (20)

4.1 Begriffe (8)

Was versteht man unter *Confidentiality (Secrecy)*, *Integrity* und *Availability*? Erklären Sie die drei Begriffe.

- Confidentiality:

- Integrity:

- Availability:

Arten der Bedrohung (Types of Threats)

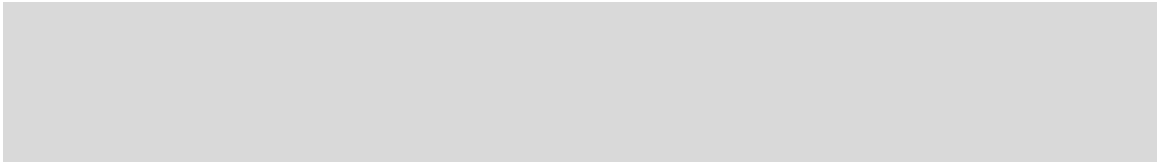
Füllen Sie folgende Tabelle derart aus, dass Sie jede angegebene Art der Bedrohung einem der Begriffe *Confidentiality (Secrecy)*, *Integrity* und *Availability* zuordnen (Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!):

Art der Bedrohung	bedroht
Interruption	
Interception	
Modification	
Fabrication	

4.2 Bedrohungen durch Programme oder Programmfragmente (8)

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

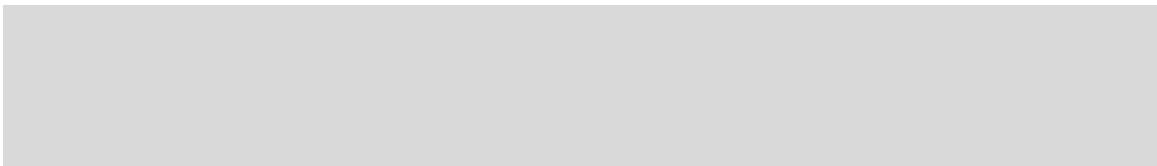
- Logic Bomb:



- Trojan Horse:



- Virus:

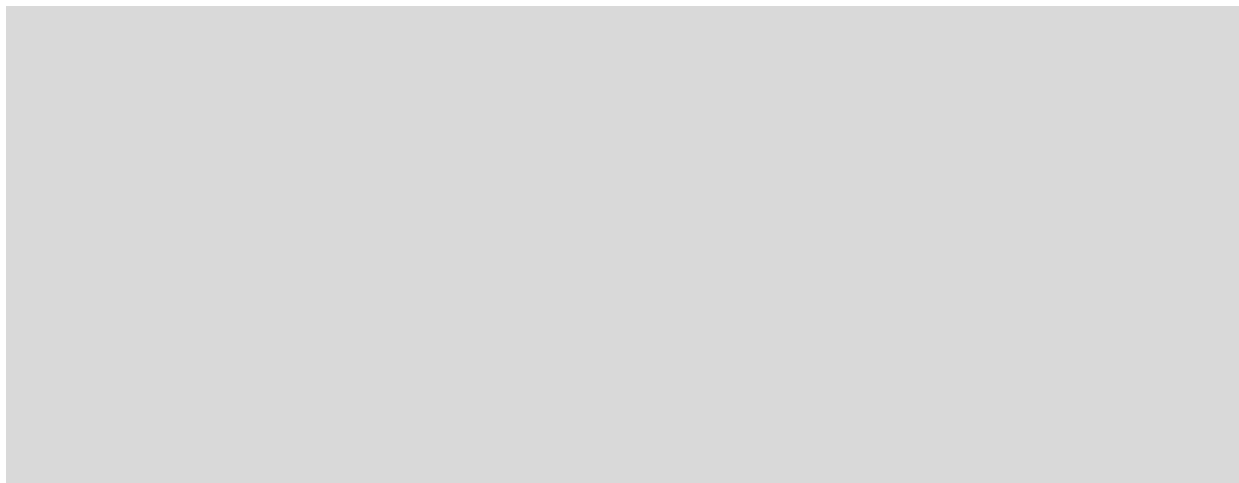


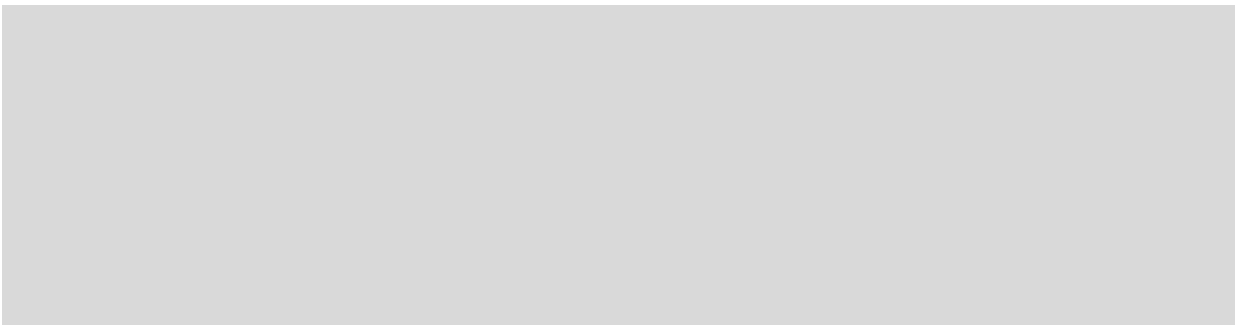
- Worm:



4.3 Security by Obscurity vs. Open Design (4)

Beschreiben Sie den Unterschied zwischen *Security by Obscurity* und *Open Design*. Geben Sie die Vor- und Nachteile an.





KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(25)

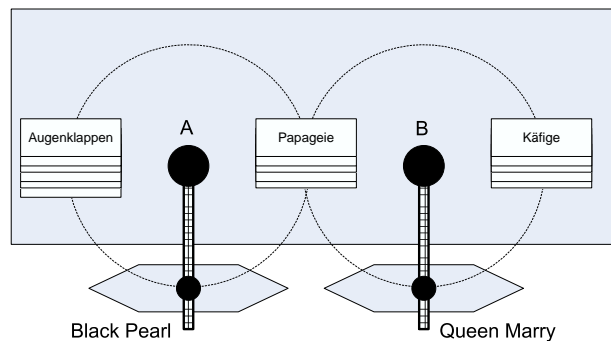
3.)(23)

4.)(22)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Die Reederei *Titanic Transports* besitzt zwei Frachtschiffe namens *Queen Marry* und *Black Pearl*. Beide Frachtschiffe haben einen eigenen Liegeplatz mit dazugehörigem Verladekran. Die Queen Marry liefert regelmäßig *Papageie* und dazugehörige *Käfige* an Tierliebhaber in ein weit entferntes Land. Die Black Pearl liefert an eine von Piraten bewohnte Insel *Papageie* und *Augenklappen* (Piraten tragen ihren Papagei auf der Schulter und brauchen daher keine Käfige).

Die Papageie, Käfige und Augenklappen sind in Containern verpackt, welche auf dem Hafen gestapelt stehen.

Das Beladen der Schiffe läuft nach folgenden Regeln ab:

- Die Queen Marry soll mit einem Container voller Papageie und einem Container voller Käfige beladen werden. Die Black Pearl soll mit einem Container voller Augenklappen und mit einem Container voller Papageie beladen werden.
- Zum Beladen werden die Kräne verwendet.
- Der Kran A, welcher der Black Pearl zugeordnet ist kann über die Container mit den Papageien, über die Container mit den Augenklappen und über die Black Pearl bewegt werden.
- Der Kran B, welcher der Queen Marry zugeordnet ist, kann über die Container mit den Papageien, über die Container mit den Käfigen und über die Queen Marry bewegt werden.

- Um ein Zusammenstoßen der Kräne zu verhindern dürfen niemals beide Kräne gleichzeitig über die Container mit den Papageien bewegt werden.
- Wenn ein Schiff fertig beladen ist, soll es sofort mit seiner Auslieferung beginnen.
- Mit dem Beladen eines Schiffes kann natürlich erst dann wieder begonnen werden wenn das Schiff zurückgekehrt ist. Ein Kran kann allerdings einen Container schon vom Stapel nehmen bevor das Schiff wieder zurückgekehrt ist.

Synchronisieren Sie den Arbeitsablauf der beiden Kräne und der beiden Schiffe mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`bewege(Richtung)` Bewegt den Kran um 90 Grad in die gewünschte *Richtung*. Als Richtung kann Uhrzeigersinn (UZS) oder Gegen_Uhrzeigersinn (GUZS) angegeben werden.

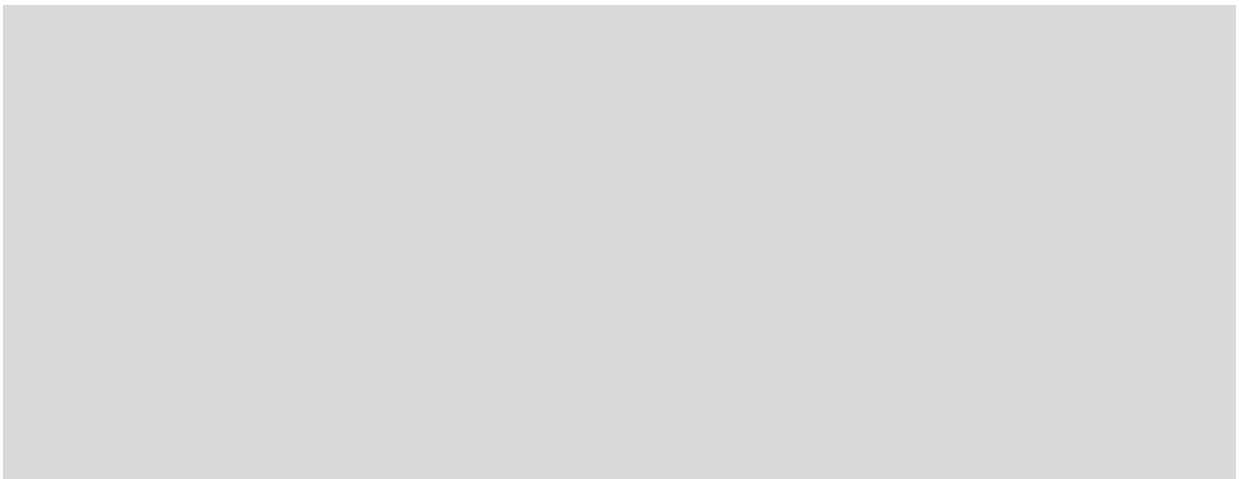
`nimm()` Lässt den Kran einen Container von dem Stapel über dem er sich gerade befindet aufnehmen. Funktioniert nur, wenn sich der Kran über einem Containerstapel befindet.

`lege_ab()` Mittels dieser Funktion legt ein Kran den Container, den er gerade hält, an der aktuellen Position ab

`lieferung()` Mit dieser Funktion liefert ein Schiff seine Ware aus, und kehrt anschließend wieder zurück.

a) Initialisierungen (8)

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** des Systems entspricht dem obigem Bild. Beide Schiffe sind unbeladen und beide Kräne stehen über den jeweiligen Schiffen.



b) (14)

Entwerfen Sie je einen Prozess für *Kran A* und *Kran B*.

Prozess *Kran A*:

```
do forever() {
```

```
}
}
```

Prozess *Kran B*:

```
do forever() {
```

```
}
}
```

c) (8)

Entwerfen Sie die Prozesse für die *Black Pearl* und die *Queen Marry*:

Prozess *Black Pearl*:

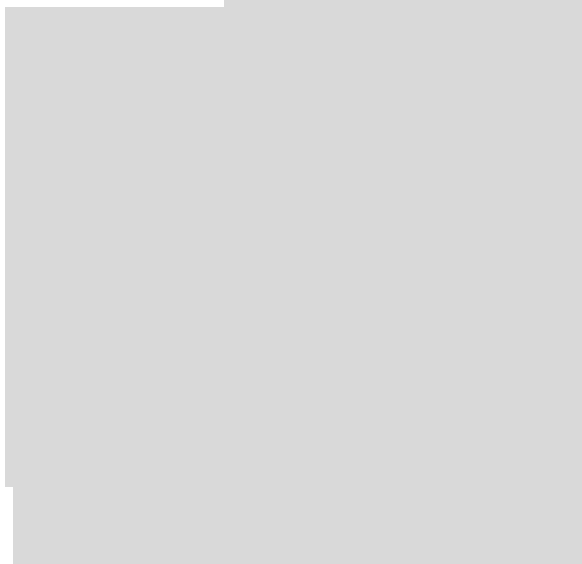
```
do forever() {
```



```
}
```

Prozess *Queen Marry*:

```
do forever() {
```



```
}
```

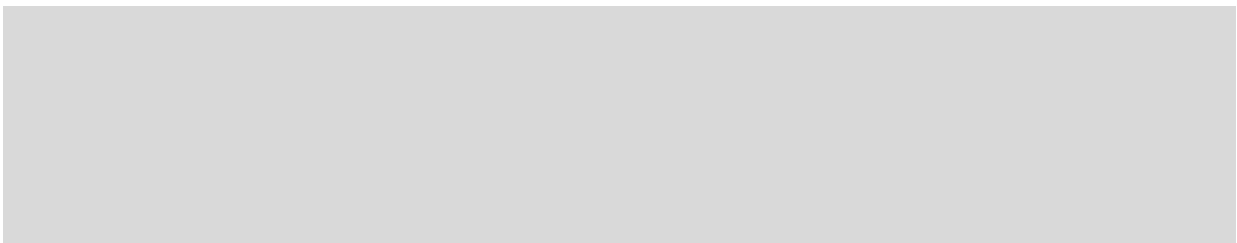
2 Scheduling (25)

2.1 Uniprocessor 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	1	10
B	1	5
C	1	4
D	2	7

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)!



Benutzen Sie dazu folgende Funktionstabelle:

n	1	2	3	4	5	6
$n \cdot (2^{(1/n)} - 1)$	1.00	0.82	0.77	0.75	0.74	0.73

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

A																	
B																	
C																	
D																	

0

5

10

15

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																	
B																	
C																	
D																	

0

5

10

15

0

2.2 Verständnisfragen (12)

Beurteilen Sie die folgenden Aussagen für Single-Prozessor Scheduling!

Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!

- ☐ Ja ☐ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.
- ☐ Ja ☐ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die *Processor Utilization* der Tasks.
- ☐ Ja ☐ Nein Earliest Deadline First Scheduling findet in Single-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 Das EDF Scheduling ist non-preemptive.
- ☐ Ja ☐ Nein Bei Round Robin Scheduling kann das Problem der Starvation auftreten.
- ☐ Ja ☐ Nein Beim Scheduling nach dem Round-Robin-Verfahren werden I/O-intensive Prozesse benachteiligt.
- ☐ Ja ☐ Nein Die First-Come-First-Serve-Strategie benachteiligt CPU-intensive Prozesse.
- ☐ Ja ☐ Nein Das Round-Robin-Verfahren ist preemptive.
- ☐ Ja ☐ Nein Das Shortest-Process-Next-Verfahren ist für Echtzeitscheduling ungeeignet.
- ☐ Ja ☐ Nein Das Shortest-Process-Next-Verfahren kommt im Vergleich zu Round Robin mit weniger Interruptaufrufen aus.
- ☐ Ja ☐ Nein Ein Scheduler nach dem RMS Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Round-Robin-Verfahren.

3 Deadlock (23)

Das Catering Service *Fein und Gesund* bekommt von verschiedenen Kunden Aufträge um auf diversen Veranstaltungen für das leibliche Wohl zu sorgen.

Um die Aufträge erfolgreich erfüllen zu können müssen bestimmte Ressourcen wie Kellner (K), Teller (T) und Sektkgläser (S) koordiniert werden. Dem Unternehmen stehen 15 Kellner, 200 Teller und 500 Sektkgläser zur Verfügung.

Im Augenblick sind drei Aufträge in Arbeit:

- Für **Auftrag 1** werden 11 Kellner, 150 Teller und 300 Sektkgläser benötigt. Reserviert sind für diesen Auftrag bereits 10 Kellner, 130 Teller und 200 Sektkgläser.
- Für **Auftrag 2** werden 5 Kellner, 70 Teller und 300 Sektkgläser benötigt. Dieser Auftrag hat noch nicht begonnen.
- Für **Auftrag 3** werden 7 Kellner, 10 Teller und 80 Sektkgläser benötigt. Reserviert sind für diesen Auftrag bereits 1 Kellner, 1 Teller und 1 Sektkglas.

Process Initiation Denial

Beschreiben Sie *Resource*- und *Available*-Vektor sowie *Claim*- und *Allocation*-Matrix. Die Matrizen sind so auszufüllen, dass die Ressourcen zeilenweise und die Aufträge spaltenweise aufgezählt werden.

$$\begin{array}{l}
 \text{Resource} = \begin{pmatrix} \text{K} & \boxed{} \\ \text{T} & \boxed{} \\ \text{S} & \boxed{} \end{pmatrix} \qquad \text{Available} = \begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} \\
 \\
 \text{Claim} = \begin{pmatrix} \text{K} & \boxed{} & \boxed{} & \boxed{} \\ \text{T} & \boxed{} & \boxed{} & \boxed{} \\ \text{S} & \boxed{} & \boxed{} & \boxed{} \end{pmatrix} \qquad \text{Allocation} = \begin{pmatrix} \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \end{pmatrix}
 \end{array}$$

Darf mit Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

☐ Ja

☐ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Deadlock Vermeidung

Verwenden Sie nun von obigem Initialzustand ausgehend den *Bankier-Algorithmus* (Banker's Algorithm) zum Scheduling der Aufträge. Geben Sie für *jeden* Schritt die Claim- und Allocation-Matrix, sowie den Available Vector und den als nächstes durchzuführenden Auftrag an. Wenn kein Auftrag mehr auszuführen ist, dann schreiben sie 'fertig' in den nächsten Schritt, falls ein Deadlock auftritt, schreiben Sie 'Deadlock' .

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in

Verwendung:

$$Claim = \begin{pmatrix} \text{K} & \text{ } & \text{ } \\ \text{T} & \text{ } & \text{ } \\ \text{S} & \text{ } & \text{ } \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \end{pmatrix} \quad Avail = \begin{pmatrix} \text{ } \\ \text{ } \\ \text{ } \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} K & \text{Bar} & \text{Bar} \\ T & \text{Bar} & \text{Bar} \\ S & \text{Bar} & \text{Bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in

Verwendung:

$$Claim = \begin{pmatrix} K & \text{Bar} & \text{Bar} \\ T & \text{Bar} & \text{Bar} \\ S & \text{Bar} & \text{Bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

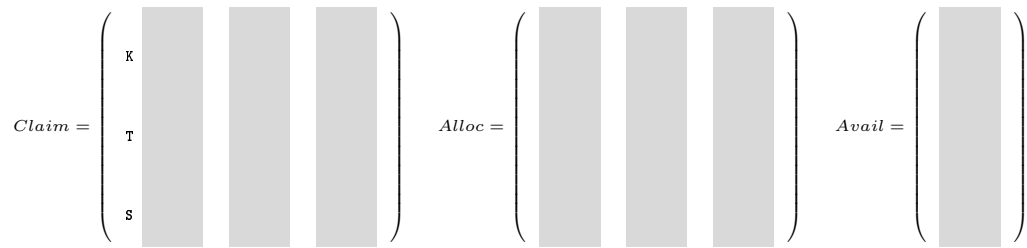
$$Claim = \begin{pmatrix} K & \text{Bar} & \text{Bar} \\ T & \text{Bar} & \text{Bar} \\ S & \text{Bar} & \text{Bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in

Verwendung:

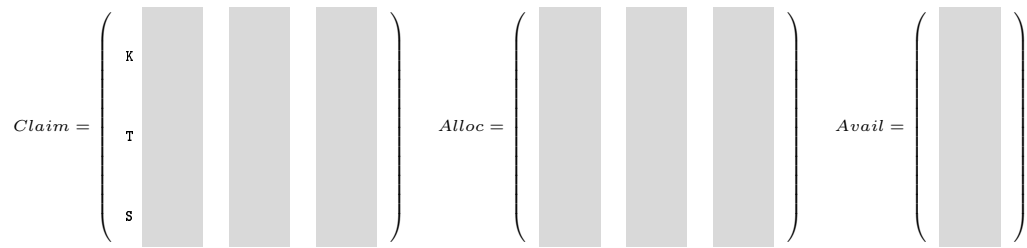
$$Claim = \begin{pmatrix} K & \text{Bar} & \text{Bar} \\ T & \text{Bar} & \text{Bar} \\ S & \text{Bar} & \text{Bar} \end{pmatrix} \quad Alloc = \begin{pmatrix} \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} & \text{Bar} \end{pmatrix} \quad Avail = \begin{pmatrix} \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \\ \text{Bar} & \text{Bar} \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

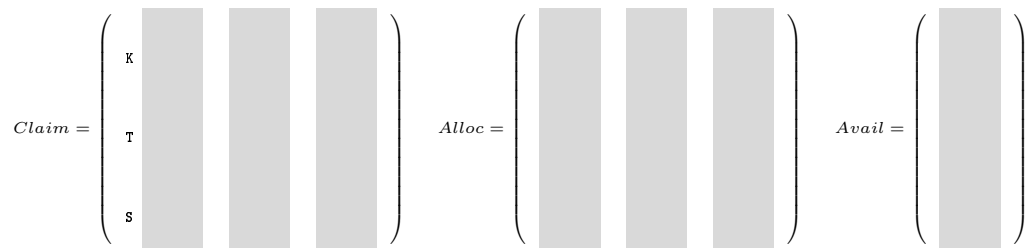


Nächster Auftrag: . Alle für den Auftrag benötigten

Ressourcen sind in Verwendung:



Nach der Ausführung des zuletzt markierten Auftrags:



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 angegebenen 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. 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
Length	Länge des Segmentes
Frame#	Seitenrahmennummer (im physischen Speicher)
Page#	Seitennummer (im virtuellen Speicher)

a) Paging — Assoziativer Zugriff (associative mapping)

Adressübersetzungstabelle:		Zu berechnende Adressen:	
Page#	Frame#	Virtuelle Adresse	Physikalische Adresse
0x11	0x33	0x083A	
0x01	0x17	0x01FF	
0x17	0x20	0x0505	
0x42	0x08		
0x05	0x05	0x063A	

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:		Zu berechnende Adressen:	
Entry	Frame#	Virtuelle Adresse	Physikalische Adresse
0	0x42	0x0406	
1	0x06		
2	0x04	0x0611	
3	0x11		
4	0x01	0x11FA	
5	0xFA		
6	0x02	0xFA11	

c) Segmentierung — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:			Zu berechnende Adressen:	
Entry	Base	Length	Virtuelle Adresse	Physikalische Adresse
0	0x1830	0x020	0x1832	
1	0x0810	0x0FF	0x0402	
2	0x42AC	0x100	0x4222	
3	0x0400	0x004	0x0010	
4	0x01FE	0x0F0	0x0305	
5	0x0010	0x030	0x01F2	
6	0x3302	0x050	0x0650	
7	0x4220	0x010		

d) Verständnisfragen

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

- Bei Paging kann das Problem der externen Fragmentierung auftreten.
 - ☐ richtig
 - ☐ falsch
- Beim *Fixed Partitioning* sind die Partionen *immer* von gleicher Größe.
 - ☐ richtig
 - ☐ falsch
- In einem fehlerfreien System können zwei oder mehr virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
 - ☐ richtig
 - ☐ falsch
- Zu welchen Effekten kann es bei Segmentierung kommen?
 - ☐ Unterschiedliche Segmentlängen
 - ☐ Internal Fragmentation
 - ☐ External Fragmentation

Je größer die “page size” (Seitengröße), desto mehr “pages” (Seiten) und “page frames” (Seitenrahmen) gibt es und desto größer müssen die “page tables” (Seitentabellen) sein.

- ☐ richtig
- ☐ falsch

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(25)

3.)(23)

4.)(22)

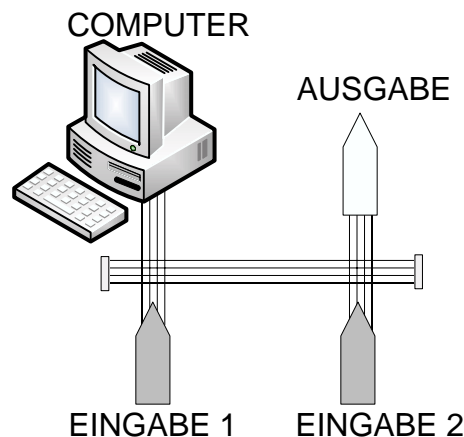
Zusatzblätter:

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 Netzwerk frei von Nachrichten ist.
- Der Computerprozess liest die empfangenen 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 Eingabeprozesse 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 Semaphore, und

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

a) Initialisierungen

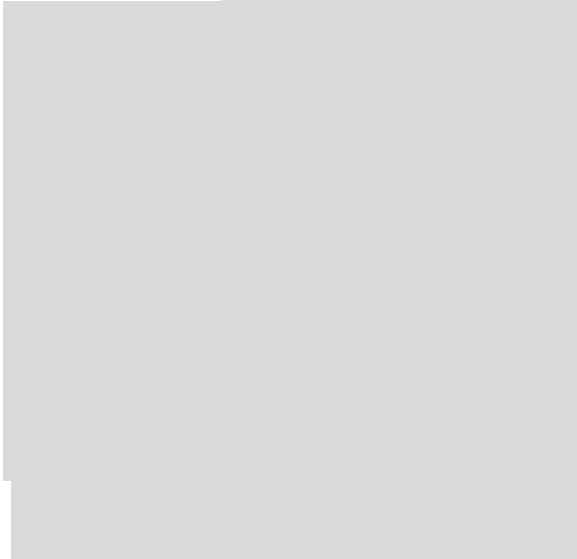
Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** ist wie folgt:

- keine Daten im Eingabegerät,
- 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*:

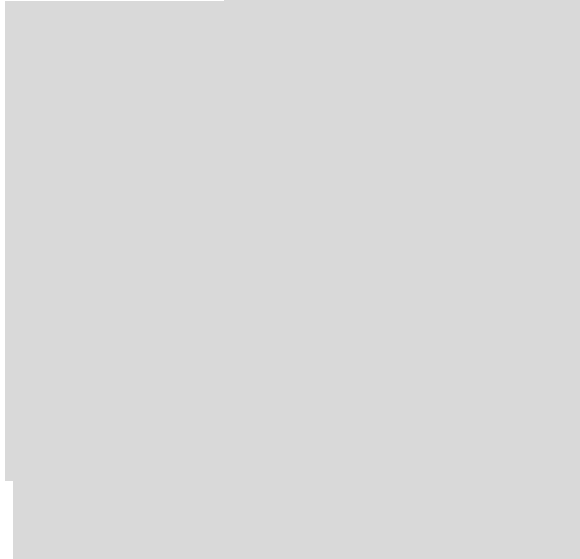
```
do forever() {
```



```
}
```

Prozess *Eingabegerät 2*:

```
do forever() {
```



```
}
```

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

Prozess *Computer*:

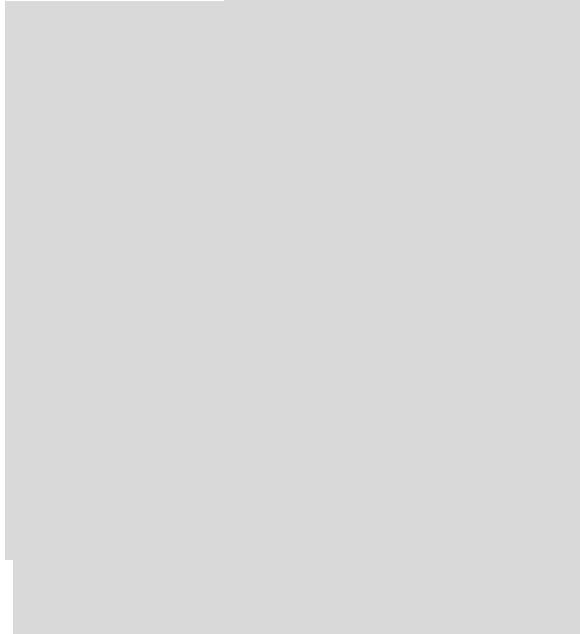
```
do forever() {
```



```
}
```

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

A																		
B																		
C																		
D																		

0

5

10

15

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

5

10

15

Ersatzvorlage: Scheduling nach dem

-Verfahren: Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

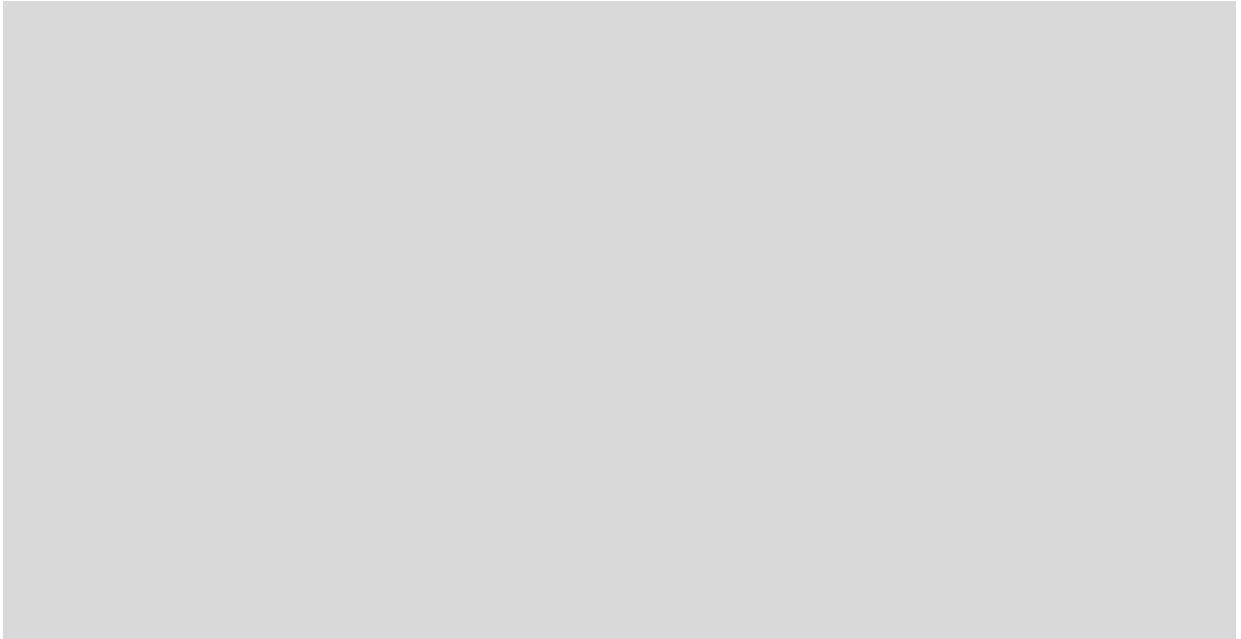
5

10

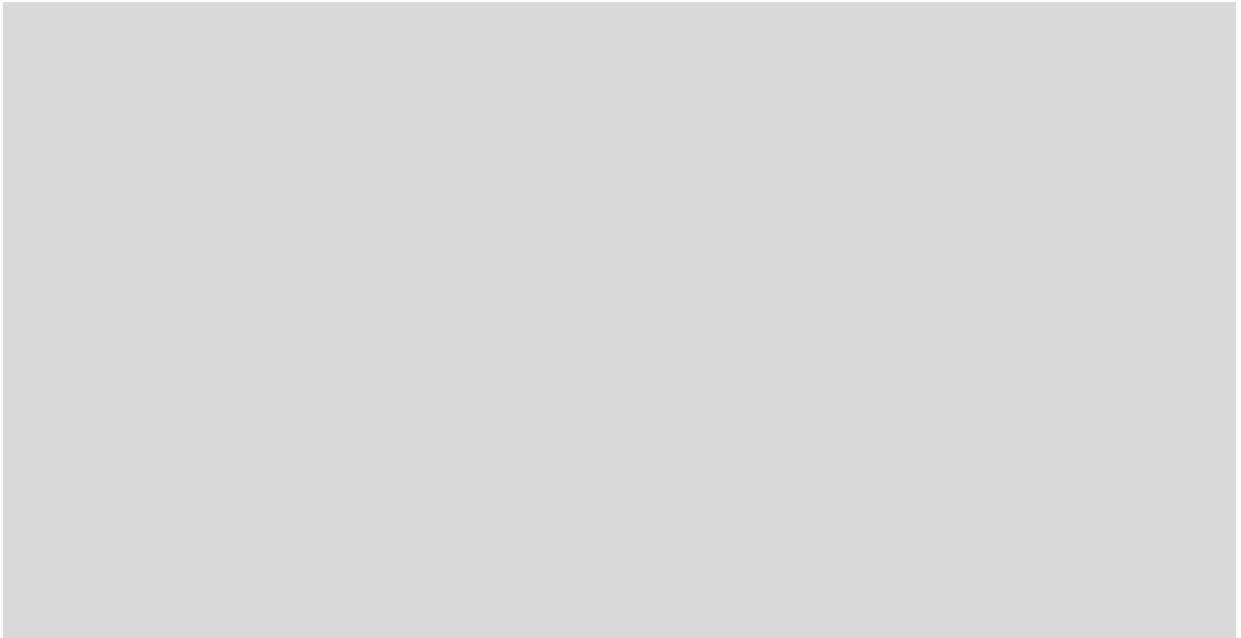
15

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 angegebenen 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:		Zu berechnende Adressen:	
Page#	Frame#	Virtuelle Adresse	Physikalische Adresse
00000101b	0x11	0x1541	
00100011b	0x01	0x0110	
00000001b	0x30	0x3105	
00110000b	0x41		
00010001b	0x02	0x0502	

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:		Zu berechnende Adressen:	
Entry	Frame#	Virtuelle Adresse	Physikalische Adresse
0	0x13	0x1306	
1	0x08		
2	0x34	0x0511	
3	0x31		
4	0x01	0x028A	
5	0x68		
6	0xA0	0x0401	

c) Segmentierung — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:			Zu berechnende Adressen:	
Entry	Base	Length	Virtuelle Adresse	Physikalische Adresse
0	0x2840	0x004	0x04A0	
1	0x0563	0x0FF	0x4222	
2	0x72AB	0x100	0x0010	
3	0x0300	0x010	0x0305	
4	0x11FD	0x0F0	0x01F2	
5	0x4444	0x030	0x0650	
6	0x3112	0x060		
7	0x4220	0x010		

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

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(20)

3.)(18)

4.)(32)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

a) Definition von Semaphoreoperationen

Geben Sie eine Implementierung der Semaphoreoperationen *init()*, *wait()* und *signal()* für *Counting Semaphores* an, indem Sie die folgenden Codeskelette mit Pseudocode ergänzen.

```
/* *** Semaphore init operation *** */
```

```
init( )
```

```
{
```

```
}
```

```
/* *** Semaphore wait operation *** */
```

```
wait( )
```

```
{
```

```
}
```

```

/* *** Semaphore signal operation *** */
signal(
{
}

```

b) Verwendung von Semaphoren

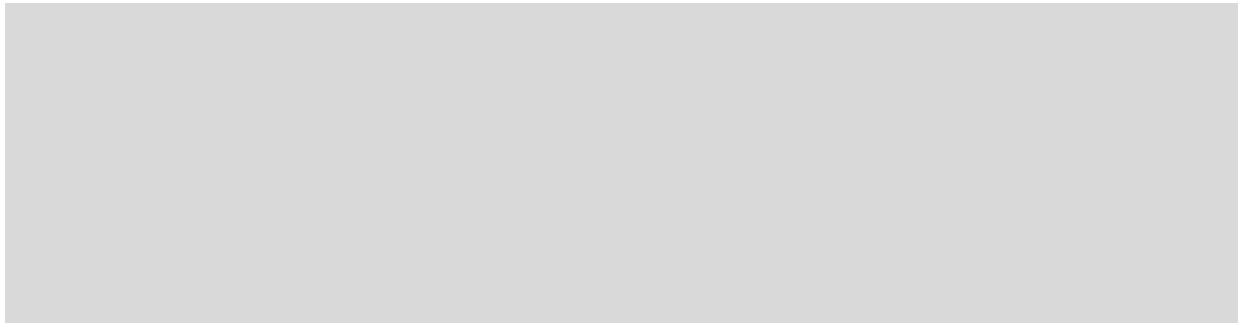
In einem Multiprozessor-System gibt es vier Prozessoren $P1, \dots, P4$ mit eigenem, privatem Speicher, sowie weiters zwei getrennte Shared Memory Blöcke $ShM1$ bzw. $ShM2$, über die die Prozesse, die auf den vier Prozessoren laufen, kommunizieren.

Datenobjekte (*data*) werden mit den Operationen $read(ShMx, data)$ bzw. $write(ShMx, data)$, wobei $ShMx$ für $ShM1$ oder $ShM2$ steht, von einem der Shared Memory Blöcke gelesen bzw. auf einen Shared Memory Block geschrieben.

Ergänzen Sie den Code von vier Tasks $T1, \dots, T4$, die jeweils in einer Endlosschleife auf einem der Prozessoren laufen, mit geeigneten Operationen zum Lesen bzw. Schreiben der Shared Memory Blöcke sowie Semaphoroperationen. Erfüllen Sie dabei die geforderten Kommunikations- und Synchronisationsanforderungen.

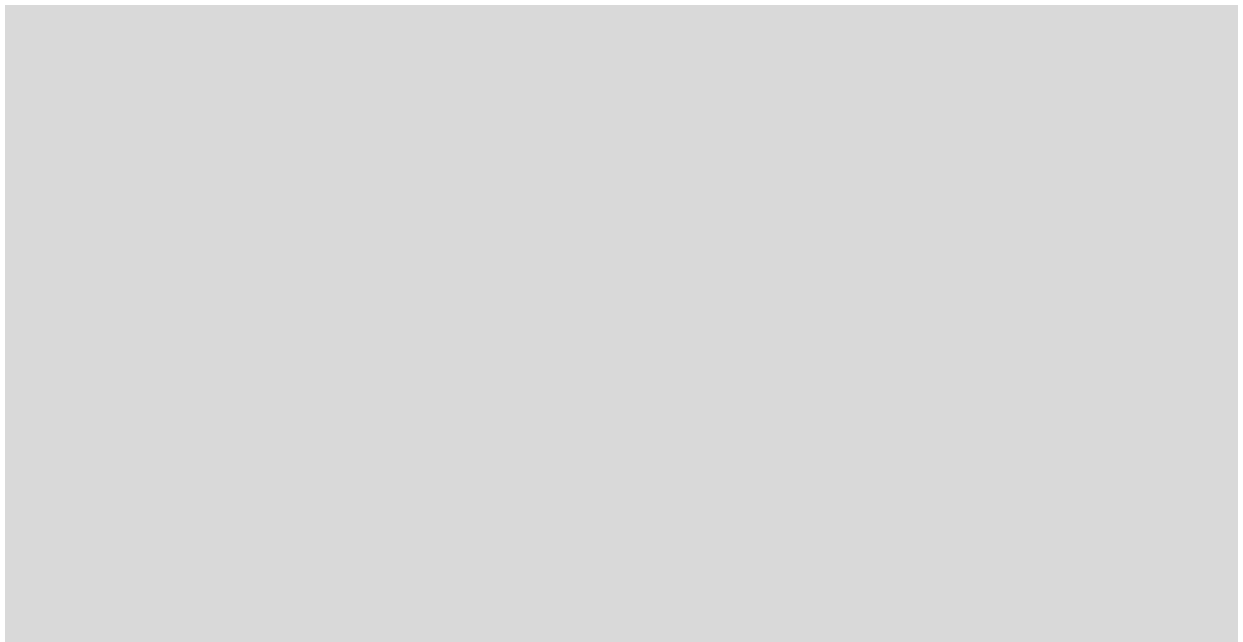
- $T1$ generiert mit der Funktion $generate(data)$ Daten und schreibt diese auf $ShM1$, von wo sie von $T2$ und $T3$ gelesen werden. Jeder von $T1$ generierte Datensatz soll von $T2$ und $T3$ genau einmal gelesen werden bevor $T1$ den nächsten Datensatz generiert.
- $T2$ und $T3$ haben identisches Verhalten. Nach jedem Auslesen von $ShM1$ (siehe voriger Punkt) lesen sie den gerade aktuellen Inhalt von $ShM2$ und verarbeiten die gelesenen Inhalte mit der Funktion $process(data1, data2)$ weiter.
- $T4$ kopiert den aktuellen Inhalt von $ShM1$ auf $ShM2$.

Initialisierungen:



Code für Task $T1$:

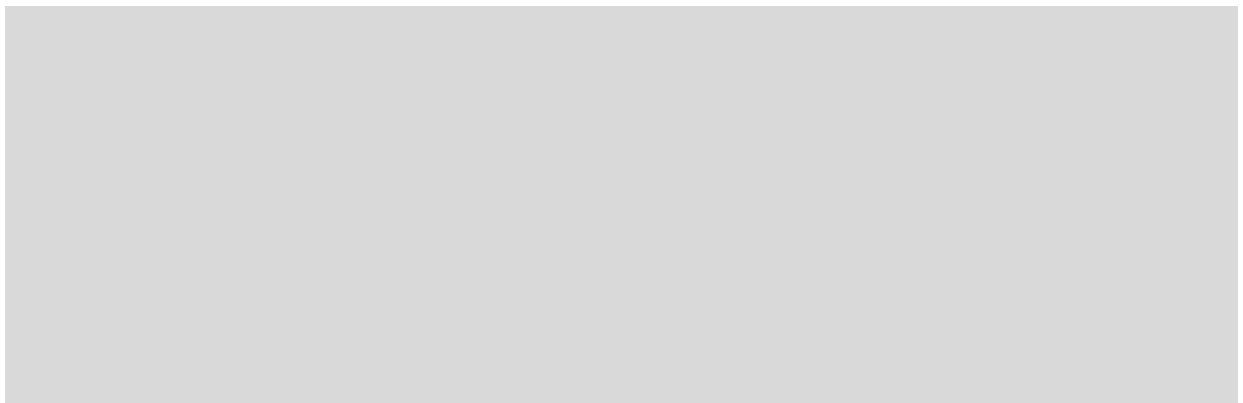
loop forever

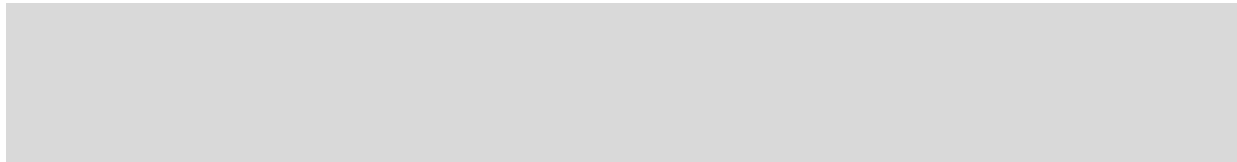


end loop

Code für Task $T2$:

loop forever

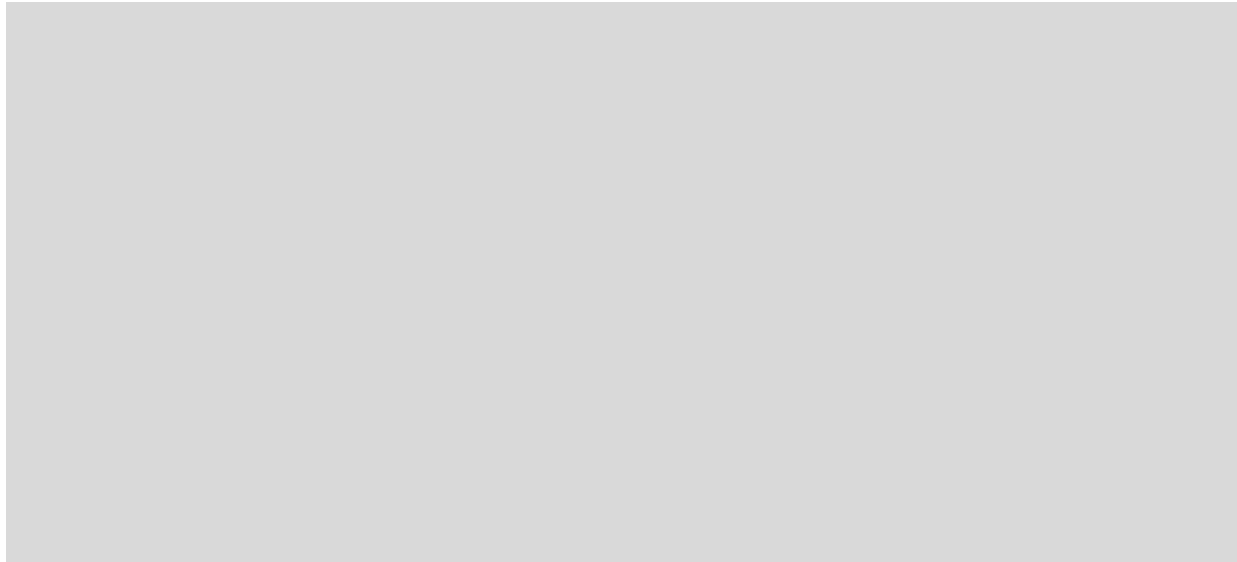




```
end loop
```

Code für Task T_3 :

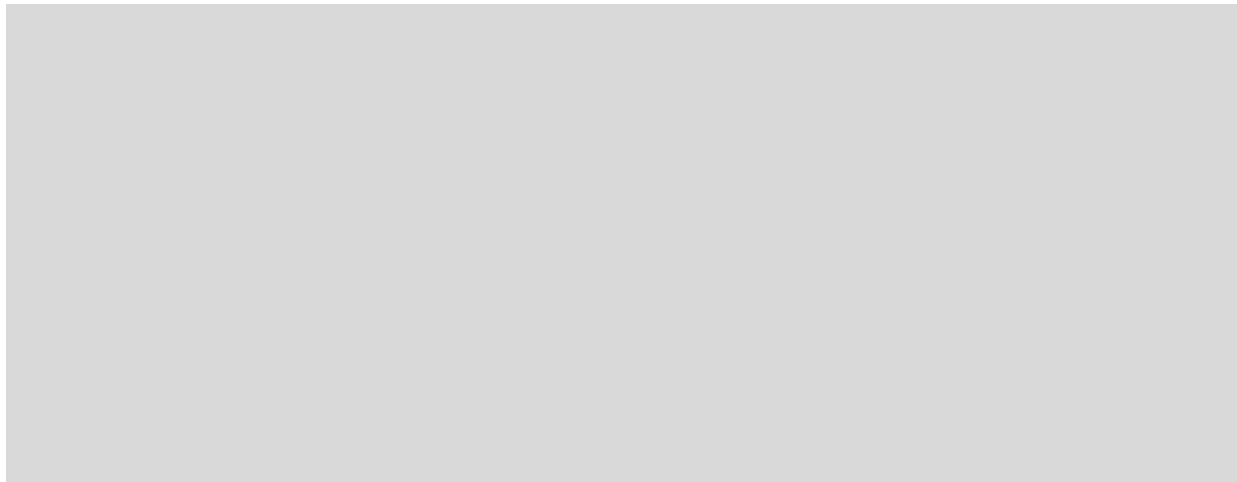
```
loop forever
```



```
end loop
```

Code für Task T_4 :

```
loop forever
```



```
end loop
```


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	1	7
B	1	6
C	2	4
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 ($\sqrt[2]{2} \approx 1,41$ $\sqrt[3]{2} \approx 1,26$ $\sqrt[4]{2} \approx 1,19$ $\sqrt[5]{2} \approx 1,15$ $\sqrt[6]{2} \approx 1,12$).

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

A																		
B																		
C																		
D																		

0

5

10

15

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

5

10

15

Ersatzvorlage: Scheduling nach dem -Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

5

10

15

3 Deadlock (18)

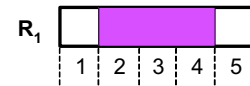
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *begin* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

2: `get(R1)`
3: ...
4: `free(R1)`



Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht möglich ist.
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.
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.
5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

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

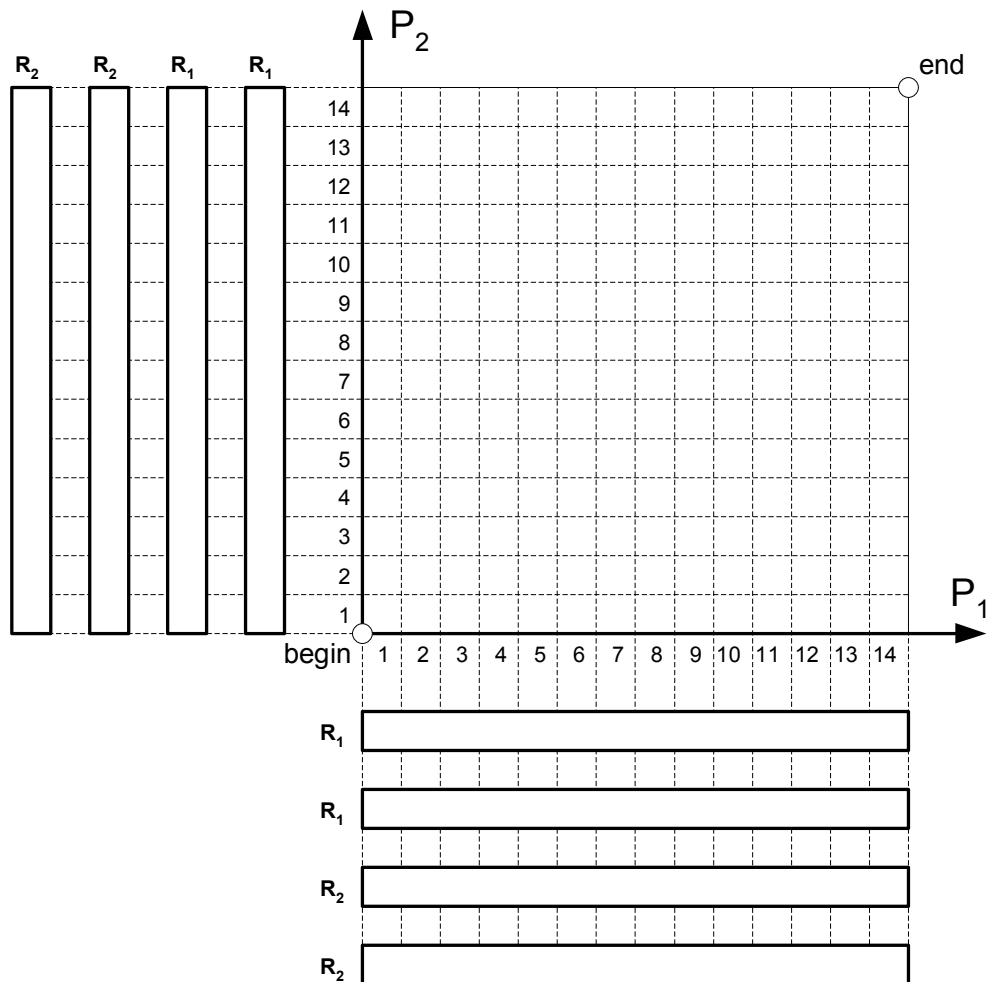
```

Program P_2 :

```

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

```



4 Memory Management (32)

a) Virtueller Speicher mit Kombination aus Segmentierung und Paging 18

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 "invalid segment nr." bzw. "invalid page nr." in das entsprechende Feld.

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle:

Segmenttabelle		
Seg#	Base	Length
0x00	0x0102 7234	0x03
0x01	0x0300 6073	0x01
0x02	0xC9C0 8054	0x02
0x03	0x8A9F 23AA	0x10
0x04	0x2001 B578	0x0A

und folgende Seitentabelle:

Seitentabelle	
Address	Frame#
...	...
0x0102 7234	0x6752
0x0102 7235	0x7613
0x0102 7236	0x258A
0x0102 7237	0xB3CA
...	...
0x0300 6073	0x56EF
0x0300 6074	0x4567
...	...
0x2001 B57F	0x5014
0x2001 B580	0x5109
0x2001 B581	0xA145
0x2001 B582	0x2000
0x2001 B583	0x3234
...	...
0xC9C0 8054	0x7353
0xC9C0 8055	0xBABA
0xC9C0 8056	0x9789
0xC9C0 8057	0xAFFE
...	...
0x8A9F 23B7	0x5400
0x8A9F 23B8	0x5401
0x8A9F 23B9	0x0945
0x8A9F 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	
0x8A9F 23B9	
0x0409 1025	
0x0409 0102	
0x04B0 1019	
0x0539 0715	
0x0407 294A	
0x0310 9728	

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

b) Verständnisfragen (14)

9

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

- Bei folgender Speicherverwaltungstechnik kann das Problem der internen Fragmentierung auftreten.
 - ☐ fixed partitioning
 - ☐ simple segmentation
 - ☐ virtual-memory paging
 - ☐ dynamic partitioning
 - ☐ simple paging
 - ☐ virtual memory with combination of paging and segmentation

- Beim *Fixed Partitioning* sind die Partionen *immer* von gleicher Größe.
 - ☐ richtig
 - ☐ falsch

- Bei folgender Speicherverwaltungstechnik tritt sowohl interne also auch externe Fragmentierung auf.
 - ☐ fixed partitioning
 - ☐ simple segmentation
 - ☐ virtual memory with combination of paging and segmentation
 - ☐ virtual-memory paging
 - ☐ dynamic partitioning
 - ☐ simple paging

- In einem fehlerfreien System können zwei oder mehr virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
 - ☐ richtig
 - ☐ falsch

VO 182.022

28. Jänner 2008

Prüfung Betriebssysteme

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(25)

2.)(31)

3.)(20)

4.)(24)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 File Management (25)

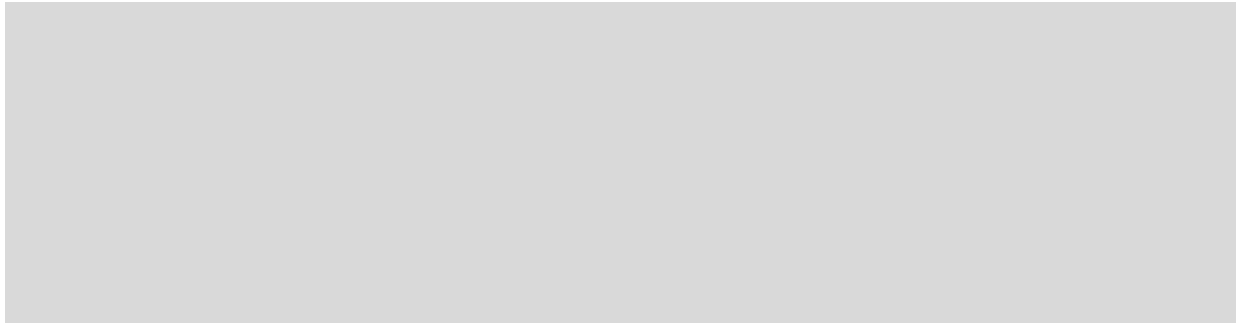
1.1 Allokation (6)

Welche Arten der Allokation kennen Sie zur Implementierung einer Datei im sekundären Speicher? Erklären Sie diese und erläutern Sie Vor- und Nachteile!



1.2 File Management Operationen (4)

Nennen und erklären Sie vier typische Operationen beim File Management!



1.3 File Management Operationen (10)

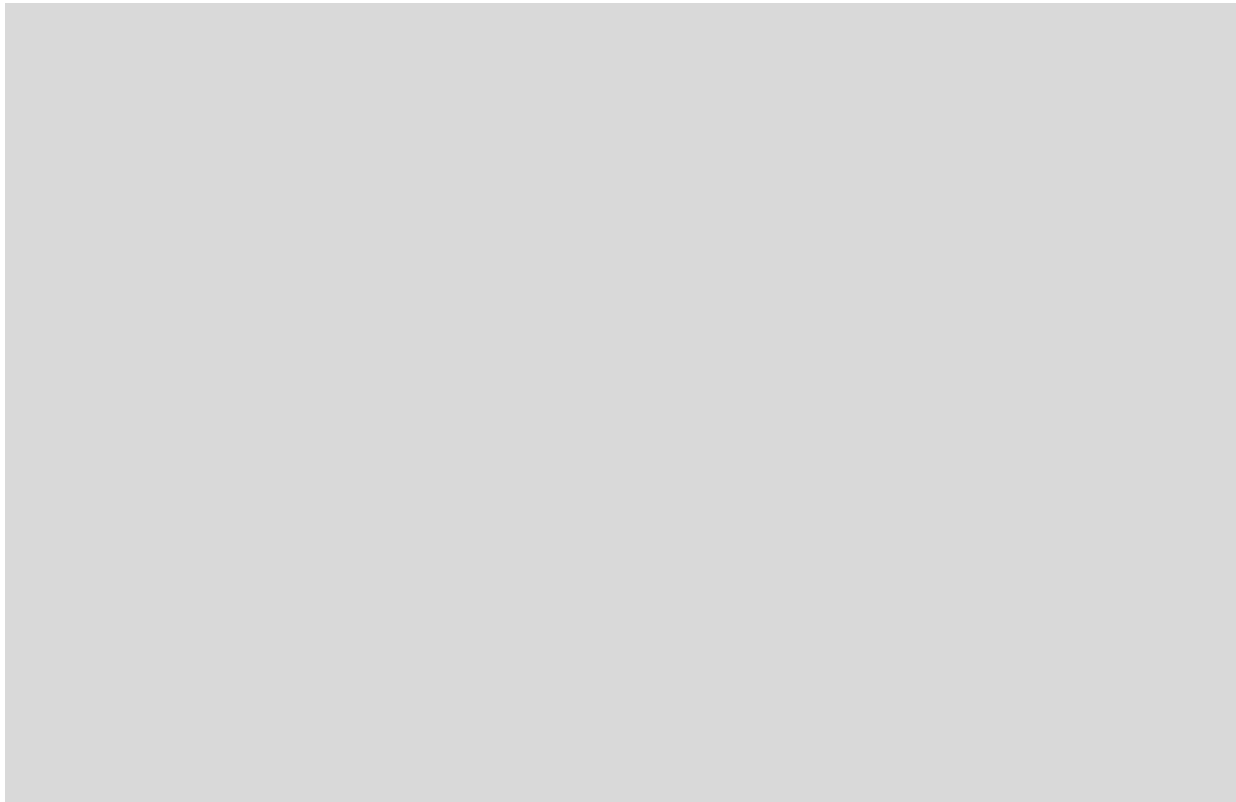
Nennen und erklären Sie fünf fundamentale Arten der Datei-Organisation!



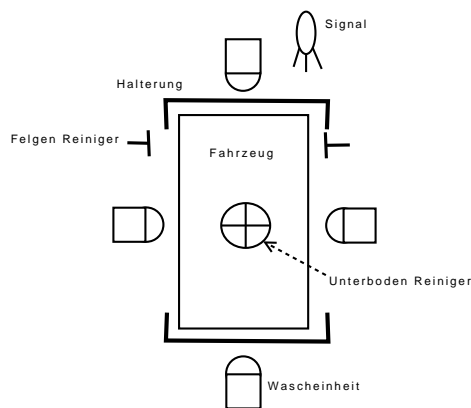


1.4 Block-Verwaltung (5)

Welche Arten der Block-Verwaltung kennen Sie? Erklären Sie ein Verfahren der Block-Verwaltung anhand eines Beispiels!



2 Synchronization (31)



Synchronisieren Sie die Tätigkeiten der Komponenten einer Waschanlage während einer Autowäsche in der Waschbox. Ein Waschvorgang läuft folgendermaßen ab:

- Der Fahrer fährt in die Waschbox (`enter_box()`). Ein Signalgeber gibt mittels eines Lichtsignals, welches auf 'Rot/Red' steht, die Anweisung das Fahrzeug in der Box zum Stillstand zu bringen (`brake()`). Der Fahrer darf erst wieder Gas geben (`accelerate()`), wenn die Wascheinheiten fertig sind und das Lichtsignal auf 'Grün/Green' gestellt ist. Damit ist der Waschvorgang beendet.
- Das Lichtsignal signalisiert dem Fahrer mit der Anzeige 'Rot/Red' bzw. 'Grün/Green' (`signal_red()` und `signal_green()`), ab wann das Fahrzeug wieder bereit ist, die Waschbox zu verlassen. Das Lichtsignal stellt sich auf 'Grün/Green' um, sobald der Waschvorgang beendet ist und das Fahrzeug nicht mehr fixiert ist.
- Um den Waschvorgang zu ermöglichen, muss das Auto in der Waschbox fixiert werden. Diese Aufgabe übernehmen zwei Halterungen, sobald das Auto still steht und das Lichtsignal dies mittels 'Rot/Red' anzeigt. Eine Halterung fixiert das Auto vorne (`lock_front()`) und eine weitere hinten (`lock_back()`). Erst dann können die Wascheinheiten, die Felgenreiniger und die Unterbodenwascheinheit mit dem Waschvorgang beginnen. Sobald diese fertig sind und dies melden, kann das Fahrzeug wieder von der Haltevorrichtung befreit werden (`unlock_front()` und `unlock_back()`) damit es die Waschbox verlassen kann.
- Insgesamt stehen 4 Wascheinheiten bereit für die Fahrzeugreinigung, je eine für jede Seite (vorne, hinten, rechts, links). Jede Wascheinheit besteht aus 4 Funktionen: Eine Düse sprüht Wasser (`sprinkle_water()`) bzw. verteilt Luft nach dem Waschvorgang (`sprinkle_air()`) und gibt ein Signal sobald der Waschvorgang beendet ist (`ready()`). Eine andere Düse verteilt Waschmittel (`sprinkle_detergent()`). Nach dem Versprühen von Wasser und dem Einbringen von Waschmittel werden die Waschbürsten bewegt (`activate_brushes()`).
- Zwei Wascheinheiten sind für die Reinigung der Felgen (engl.: Rim) zuständig (`clean_rim()`). Sobald die Felgen gereinigt sind, wird das mittels eines Signals gemel-

det (`ready()`). Damit kann das Fahrzeug seitens der Felgenreinigungseinheit wieder aus der Halterung gelöst werden.

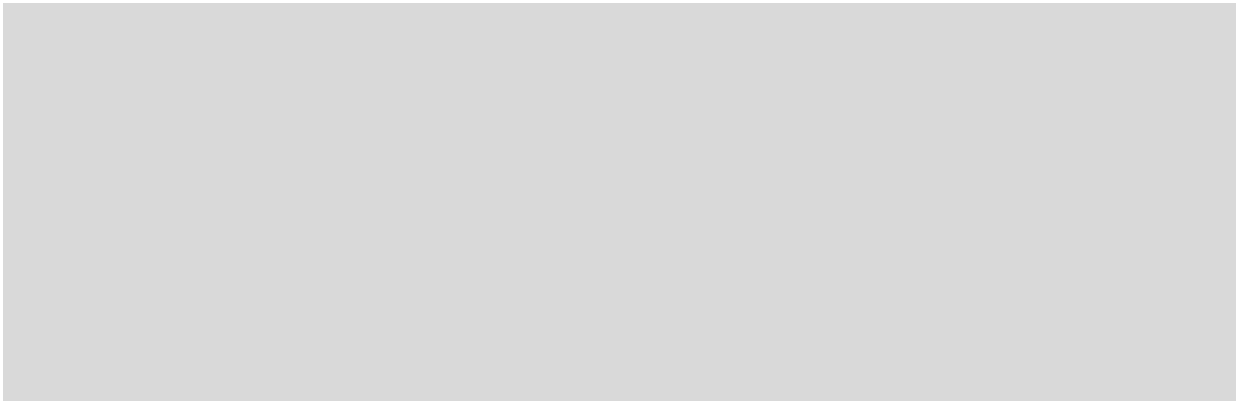
- Eine Unterbodenreinigungseinheit säubert den Unterboden des Fahrzeugs (`clean_underbody()`). Sobald der Unterboden gereinigt ist, wird ein Signal gegeben (`ready()`). Damit kann das Fahrzeug von Seiten der Unterbodenreinigungseinheit freigegeben werden.

Ergänzen Sie den folgenden Code entsprechend. Beachten Sie dabei folgende Punkte

- Die Synchronisation hat durch Semaphore zu erfolgen, welche in der Funktion `Init()` zu initialisieren sind. Dafür steht die Funktion `initsem(semaphor,value)` zur Verfügung. Zur Synchronisation sind `P(semaphor)` und `V(semaphor)` zu verwenden. Verwenden Sie eine minimale Anzahl von Ressourcen!
- Achten Sie auf maximale Parallelität, um den Boxenstopp so schnell wie möglich zu beenden!

Initialisierung:

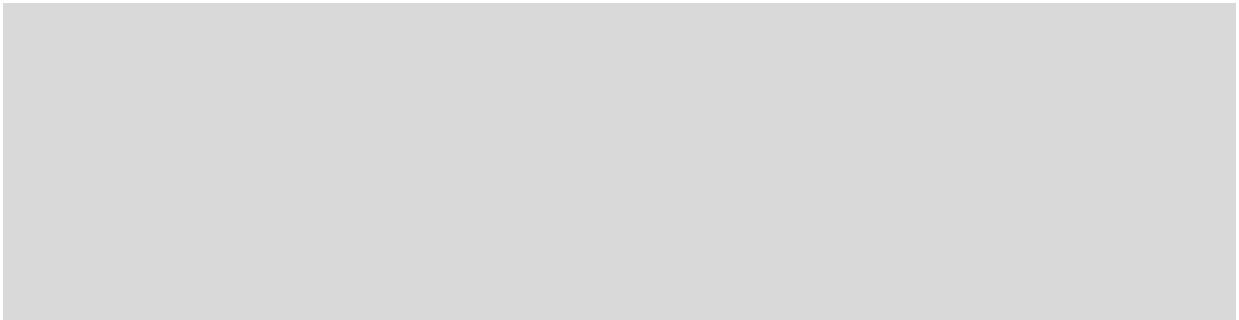
```
Init() {
```



```
}
```

Der Fahrer:

```
Driver() {
```



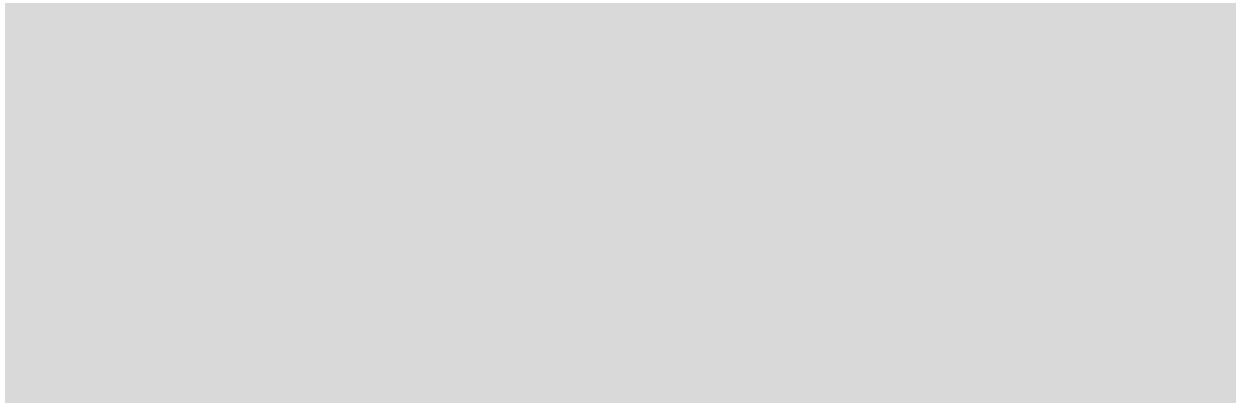


```
}
```

Wascheinheiten:

Obwohl diese Funktion viermal aufgerufen wird (für jede Seite), braucht diese Funktion nur einmal programmiert werden. Diese Funktion implementiert die Tätigkeit von je drei Wascheinheiten pro Fahrzeugseite.

```
Cleaning_Units() {
```

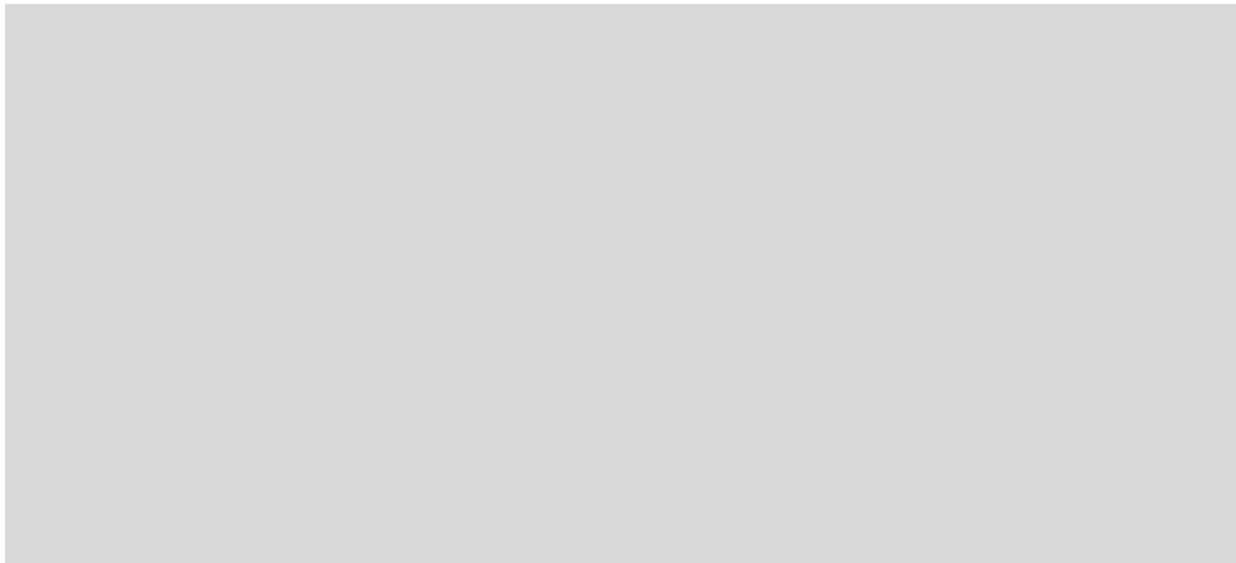


```
}
```

Halterungen zur Fixierung des Fahrzeugs:

Diese Funktion implementiert die Tätigkeit der zwei Halterungen, welche das Auto fixieren (vorne und hinten).

```
Lock_Car() {
```



```
}
```

Die Unterbodenreinigungseinheit:

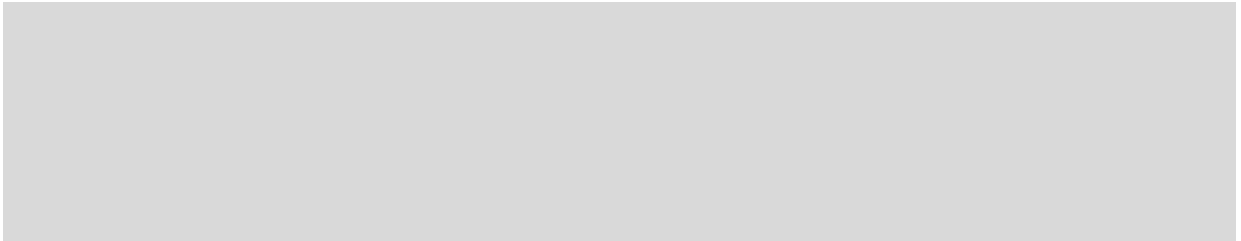
```
Underbody_Unit() {
```



```
}
```

Reinigung der Felgen:

```
Rim_Unit() {
```



```
}
```

Das Signallicht:

```
Signal() {
```



```
}
```

3 Security (20)

3.1 Begriffe (8)

Was versteht man unter *Confidentiality (Secrecy)*, *Integrity* und *Availability*? Erklären Sie die drei Begriffe.

- 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 Design Principles for Security (8)

Nennen Sie mindestens 4 Design Prinzipien für Security und beschreiben Sie diese kurz:

3.3 Maßnahmen gegen Passwortattacken (4)

Nennen Sie mindestens 4 Maßnahmen gegen Passwortattacken:

4 Deadlock (24)

Bedingungen für Deadlock (8)

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

Behandlung von Deadlocks (8)

Erklären Sie den Begriff der Deadlock Prevention. Führen Sie zu jedem der beiden Arten von Deadlock Prevention ein Beispiel für eine Strategie zur Behandlung von Deadlocks an.

Worin liegt der Unterschied zwischen Deadlock Prevention und Deadlock Avoidance?

Process Initiation Denial (8)

Gegeben ist ein System von 3 Prozessen (P1, P2 und P3) und deren Gesamtanforderung an Ressourcen, dargestellt durch die *Claim-Matrix* C . Die Prozesse verwenden 3 Ressourcenkategorien (R1, R2 und R3), deren insgesamt zur Verfügung stehende Gesamtanzahl durch den Vektor R abgebildet wird.

Im aktuellen Zustand des Systems werden die Prozesse P1 und P3 bereits ausgeführt. Beschreiben Sie anhand dieses Systemzustandes die gegenwärtige Ressourcenbelegung durch Ausfüllen der *Allocation-Matrix* A und des *Available-Vektors* V (in den Matrizen repräsentiert jede Spalte eine Ressource und jede Zeile einen Prozess).

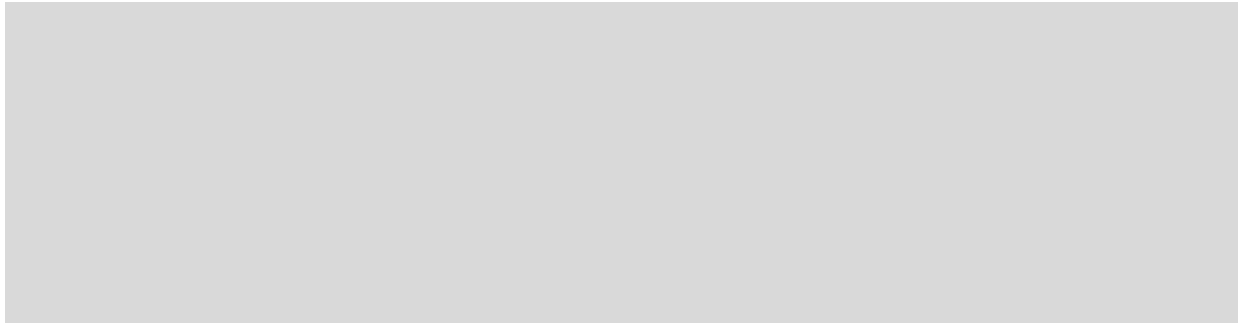
$$R = \begin{pmatrix} 80 & 5 & 6 \end{pmatrix} \quad V = \begin{pmatrix} \square & \square & \square \end{pmatrix}$$
$$C = \begin{pmatrix} 10 & 1 & 2 \\ 40 & 1 & 4 \\ 16 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}$$

Darf mit Programm 2 gemäß *Process Initiation Denial* begonnen werden?

☐ Ja

☐ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):



KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(20)

3.)(18)

4.)(32)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

a) Definition von Semaphoreoperationen

Geben Sie eine Implementierung der Semaphoreoperationen *init()*, *wait()* und *signal()* für *Counting Semaphores* an, indem Sie die folgenden Codeskelette mit Pseudocode ergänzen.

```
/* *** Semaphore init operation *** */
```

```
init( )
```

```
{
```

```
}
```

```
/* *** Semaphore wait operation *** */
```

```
wait( )
```

```
{
```

```
}
```

```

/* *** Semaphore signal operation *** */
signal(
{
}

```

b) Verwendung von Semaphoren

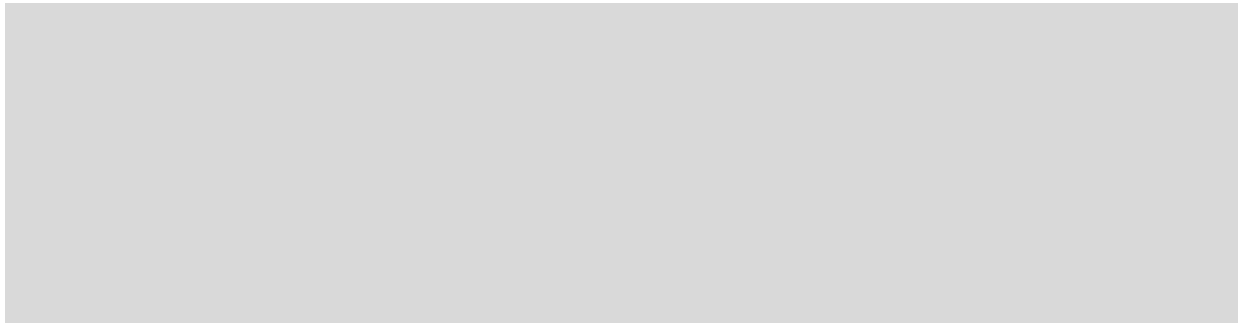
In einem Multiprozessor-System gibt es vier Prozessoren $P1, \dots, P4$ mit eigenem, privatem Speicher, sowie weiters zwei getrennte Shared Memory Blöcke $ShM1$ bzw. $ShM2$, über die die Prozesse, die auf den vier Prozessoren laufen, kommunizieren.

Datenobjekte (*data*) werden mit den Operationen $read(ShMx, data)$ bzw. $write(ShMx, data)$, wobei $ShMx$ für $ShM1$ oder $ShM2$ steht, von einem der Shared Memory Blöcke gelesen bzw. auf einen Shared Memory Block geschrieben.

Ergänzen Sie den Code von vier Tasks $T1, \dots, T4$, die jeweils in einer Endlosschleife auf einem der Prozessoren laufen, mit geeigneten Operationen zum Lesen bzw. Schreiben der Shared Memory Blöcke sowie Semaphoroperationen. Erfüllen Sie dabei die geforderten Kommunikations- und Synchronisationsanforderungen.

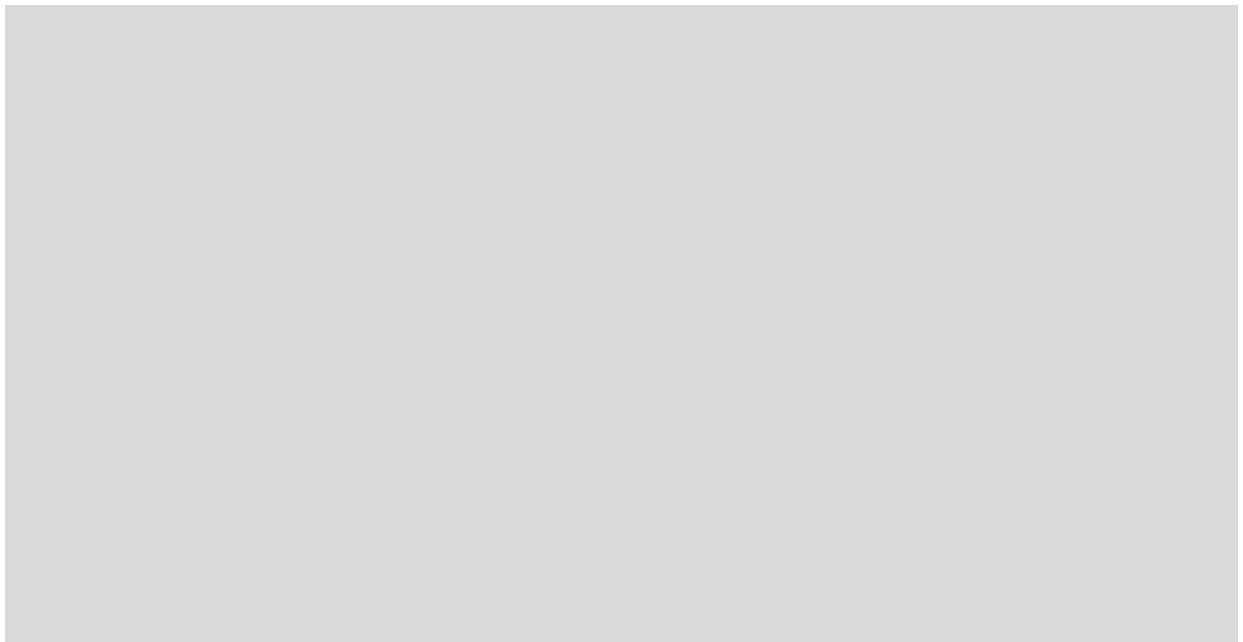
- $T1$ generiert mit der Funktion $generate(data)$ Daten und schreibt diese auf $ShM1$, von wo sie von $T2$ und $T3$ gelesen werden. Jeder von $T1$ generierte Datensatz soll von $T2$ und $T3$ genau einmal gelesen werden bevor $T1$ den nächsten Datensatz generiert.
- $T2$ und $T3$ haben identisches Verhalten. Nach jedem Auslesen von $ShM1$ (siehe voriger Punkt) lesen sie den gerade aktuellen Inhalt von $ShM2$ und verarbeiten die gelesenen Inhalte mit der Funktion $process(data1, data2)$ weiter.
- $T4$ kopiert den aktuellen Inhalt von $ShM1$ auf $ShM2$.

Initialisierungen:



Code für Task $T1$:

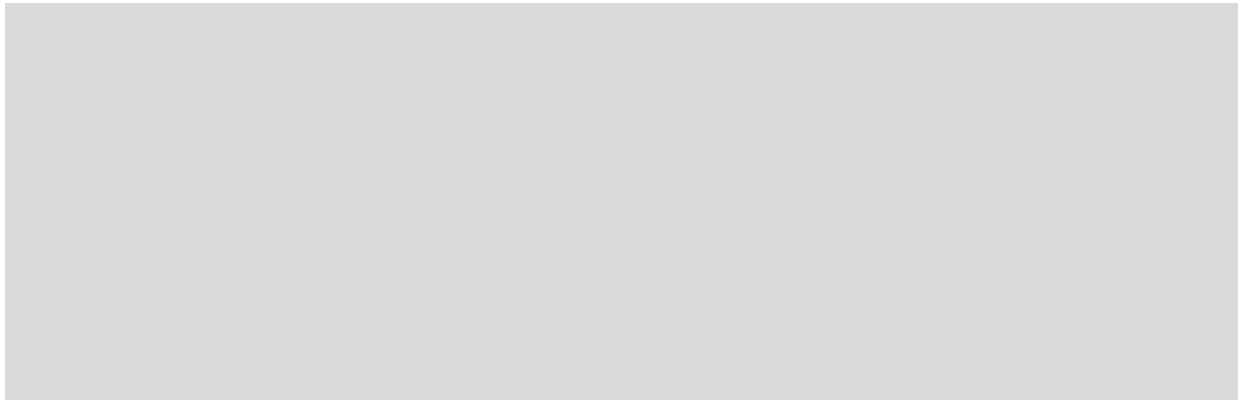
loop forever

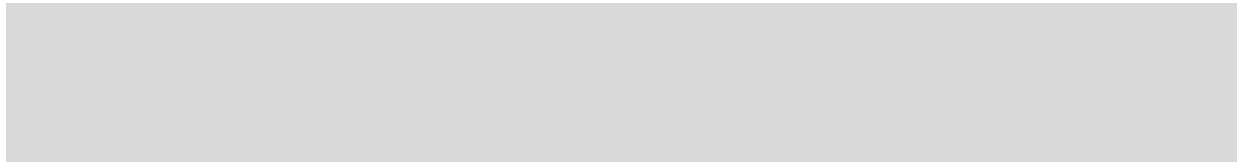


end loop

Code für Task $T2$:

loop forever

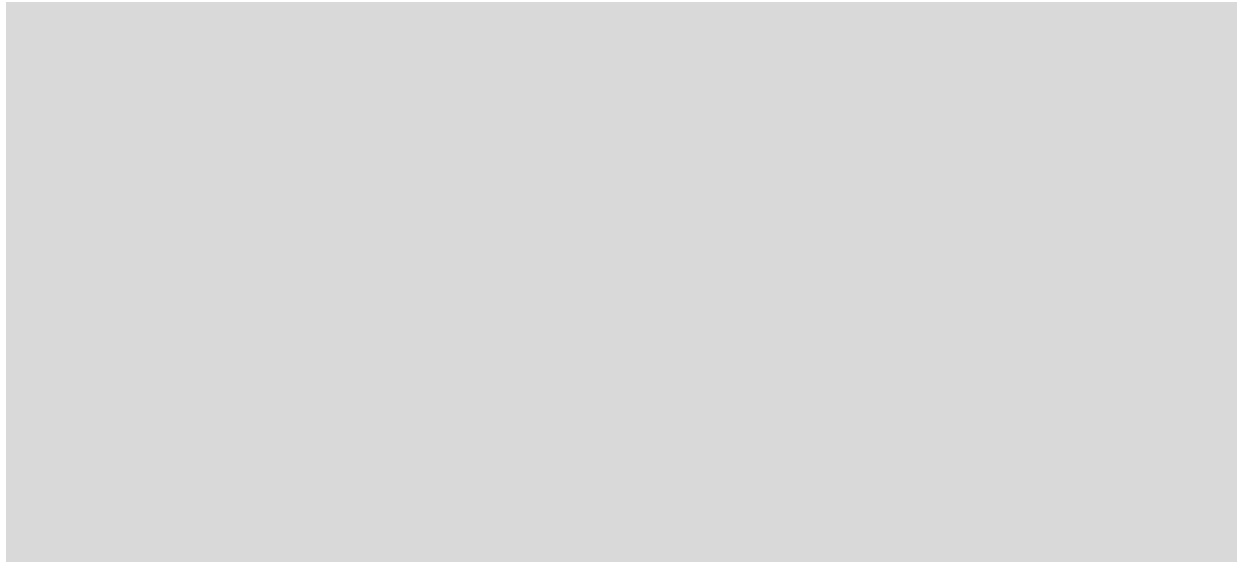




```
end loop
```

Code für Task T_3 :

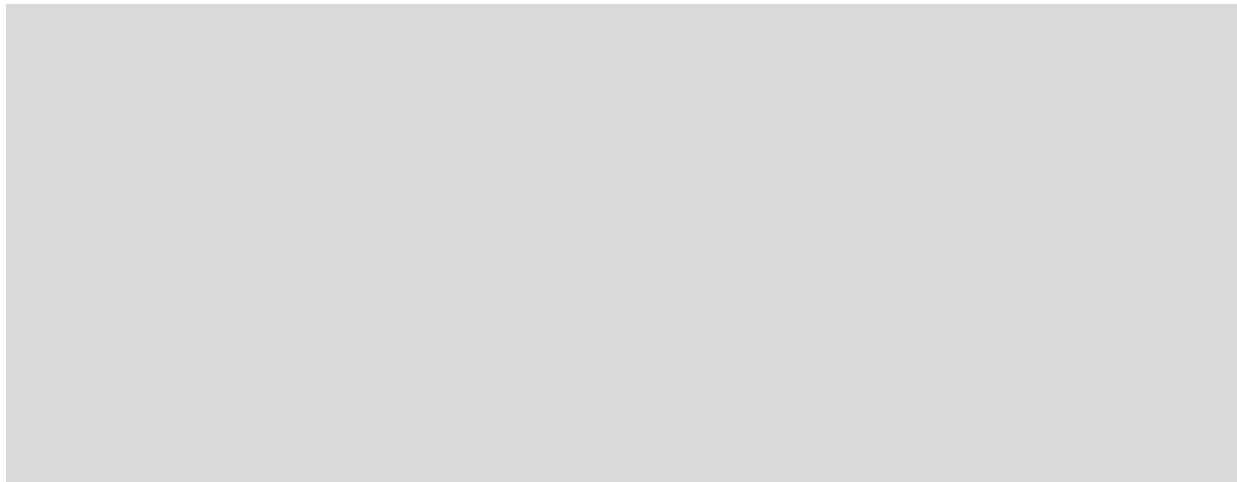
```
loop forever
```



```
end loop
```

Code für Task T_4 :

```
loop forever
```



```
end loop
```

2 Deadlock (18)

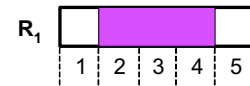
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *begin* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

```
2: get(R1)
3: ...
4: free(R1)
```



Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht möglich ist.
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.
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.
5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

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

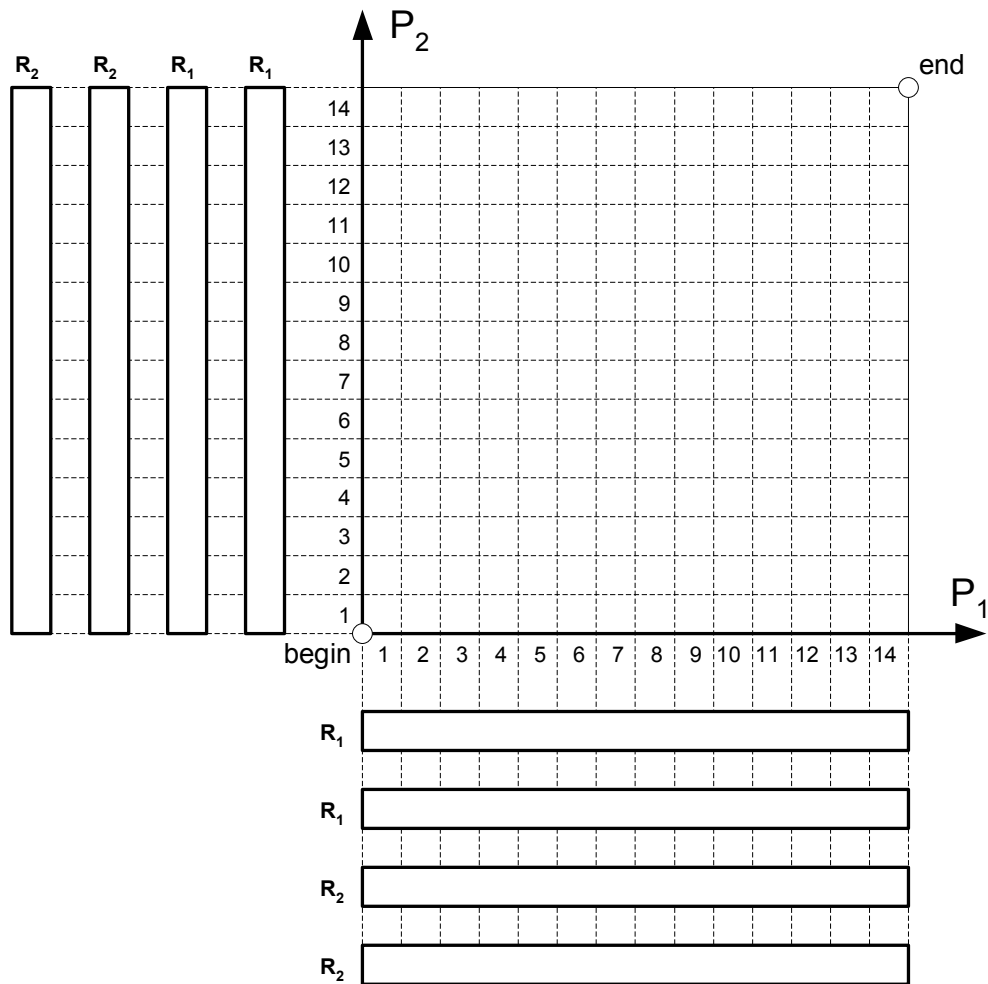
```

Program P_2 :

```

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

```



3 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	8
B	1	4
C	2	9
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** überschlagsmäßig ($\sqrt[2]{2} \approx 1,41$ $\sqrt[3]{2} \approx 1,26$ $\sqrt[4]{2} \approx 1,19$ $\sqrt[5]{2} \approx 1,15$ $\sqrt[6]{2} \approx 1,12$).

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

A																		
B																		
C																		
D																		
	0				5					10						15		

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		
	0					5				10						15		

Ersatzvorlage: Scheduling nach dem -Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		
	0						5				10						15	

4 Memory Management (32)

a) Virtueller Speicher mit Kombination aus Segmentierung und Paging 18

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 "invalid segment nr." bzw. "invalid page nr." in das entsprechende Feld.

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle:

Segmenttabelle		
Seg#	Base	Length
0x00	0x0102 7234	0x03
0x01	0x0300 6073	0x01
0x02	0xC9C0 8054	0x02
0x03	0x8A9F 23AA	0x10
0x04	0x2001 B578	0x0A

und folgende Seitentabelle:

Seitentabelle	
Address	Frame#
...	...
0x0102 7234	0x6752
0x0102 7235	0x7613
0x0102 7236	0x258A
0x0102 7237	0xB3CA
...	...
0x0300 6073	0x56EF
0x0300 6074	0x4567
...	...
0x2001 B57F	0x5014
0x2001 B580	0x5109
0x2001 B581	0xA145
0x2001 B582	0x2000
0x2001 B583	0x3234
...	...
0xC9C0 8054	0x7353
0xC9C0 8055	0xBABA
0xC9C0 8056	0x9789
0xC9C0 8057	0xAFFE
...	...
0x8A9F 23B7	0x5400
0x8A9F 23B8	0x5401
0x8A9F 23B9	0x0945
0x8A9F 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	
0x8A9F 23B9	
0x0409 1025	
0x0409 0102	
0x04B0 1019	
0x0539 0715	
0x0407 294A	
0x0310 9728	

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

b) Verständnisfragen (14)

9

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

- Bei folgender Speicherverwaltungstechnik kann das Problem der internen Fragmentierung auftreten.
 - ☐ fixed partitioning
 - ☐ simple segmentation
 - ☐ virtual-memory paging
 - ☐ dynamic partitioning
 - ☐ simple paging
 - ☐ virtual memory with combination of paging and segmentation

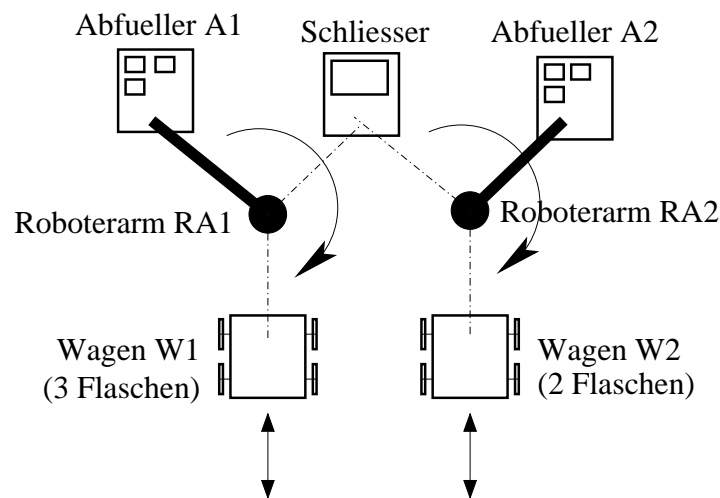
- Beim *Fixed Partitioning* sind die Partionen *immer* von gleicher Größe.
 - ☐ richtig
 - ☐ falsch

- Bei folgender Speicherverwaltungstechnik tritt sowohl interne also auch externe Fragmentierung auf.
 - ☐ fixed partitioning
 - ☐ simple segmentation
 - ☐ virtual memory with combination of paging and segmentation
 - ☐ virtual-memory paging
 - ☐ dynamic partitioning
 - ☐ simple paging

- In einem fehlerfreien System können zwei oder mehr virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
 - ☐ richtig
 - ☐ falsch

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Die Molkerei *Lolatte* besitzt zwei Abfüllanlagen sowie eine gemeinsame Schließanlage für die Flaschen nach dem Befüllen. Je ein Roboterarm nimmt fertig gefüllte Flaschen von der jeweiligen Abfüllanlage und bringt sie zum Schließer, wo die Flasche luftdicht abgeschlossen wird. Anschließend wird die Flasche in einen Wagen gestellt, welcher regelmäßig ausgeleert wird. Der Wagen “W1” kann drei Flaschen aufnehmen, bevor er geleert werden muss. Der Wagen “W2” kann nur zwei Flaschen aufnehmen, bevor er wieder geleert werden muss. Die Roboterarme haben die Eigenheit, dass sie sich beide **nur im Uhrzeigersinn drehen** lassen, wie auch in obiger Abbildung dargestellt.

Das Beladen der beiden Wagen läuft nach folgenden Regeln ab:

- Jeder Roboterarm kann direkt ohne Zeitverzögerung von seiner Abfüllanlage eine Flasche holen.
- Wenn ein Roboterarm sich gerade zum Schließer hinbewegt oder bereits wieder wegbewegt, wäre die Gefahr gegeben, dass er mit dem zweiten Roboterarm kollidiert. Daher muss dieser Abschnitt entsprechend gesichert werden.
- Der *Schließer* wartet bis ein Roboterarm direkt vor ihm eine Flasche platziert hat und verschließt diese dann.

- Der Drehwinkel für den Roboterarm ist zwischen dem Schließer und der Abfüllanlage bzw. dem Wagen genau 120 Grad.
- Wenn ein Wagen fertig beladen ist, soll er sofort mit seiner Auslieferung beginnen.
- Mit dem Anfüllen eines Wagens kann natürlich erst dann wieder begonnen werden wenn der Wagen zurückgekehrt ist. Der Roboterarm kann allerdings schon die nächste Fläche verschließen bevor der Wagen zurueck ist.

Synchronisieren Sie den Arbeitsablauf der beiden Roboterarme, des Schließers und der beiden Wagen mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Allgemeine zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen `P(Semaphor)` und `V(Semaphor)` auf den Semaphor angewendet werden.

Zu verwendende Funktionen für den Schließer:

`close()` Mit dieser Funktion wird eine Flasche verschlossen. Funktioniert nur, wenn einer der Roboterarme zum Abfüller ausgerichtet ist und eine Flasche hält. Die Flasche braucht zum Schließen vom Roboterarm nicht losgelassen werden.

Zu verwendende Funktionen für die Roboterarme:

`turn(Winkel)` Bewegt den Roboterarm im Uhrzeigersinn um *Winkel* Grad weiter (Winkel ist ein Wert im Bereich von 0-360 Grad). Beachten Sie, dass Roboterarm RA2 komplett baugleich wie RA1 ist und sich daher ebenfalls ausschließlich im Uhrzeigersinn dreht!

`get()` Nimmt von der Abfüllanlage eine Flasche. Funktioniert nur, wenn der Roboterarm zur Abfüllanlage ausgerichtet ist.

`put()` Stellt die Flasche in den Wagen. Funktioniert nur, wenn der Roboterarm zum jeweiligen Wagen ausgerichtet ist.

Zu verwendende Funktionen für die Wagen:

`deliver()` Mit dieser Funktion fährt der Wagen in die Vertriebshalle und kommt anschließend wieder leer zurück.

a) Initialisierungen (4)

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** des Systems entspricht dem obigem Bild. Beide Wagen sind unbeladen und beide Roboterarme sind zur jeweiligen Abfüllanlage platziert.



b) (4)

Entwerfen Sie einen Prozess für *Schliesser*.

Prozess *Schliesser*:

```
do forever() {
```



```
}
```

c) (16)

Entwerfen Sie je einen Prozess für *Roboterarm RA1* und *Roboterarm RA2*.

Prozess *Roboterarm RA1*:

```
do forever() {
```

Prozess *Roboterarm RA2*:

```
do forever() {
```

```
}
```

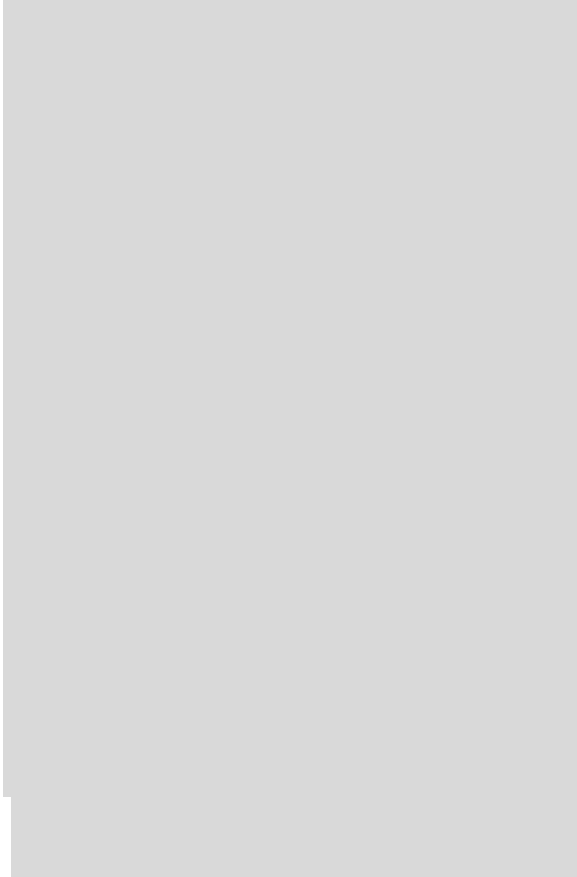
```
}
```


d) (6)

Entwerfen Sie die Prozesse für *Wagen W1* und *Wagen W2*:

Prozess *Wagen W1*:

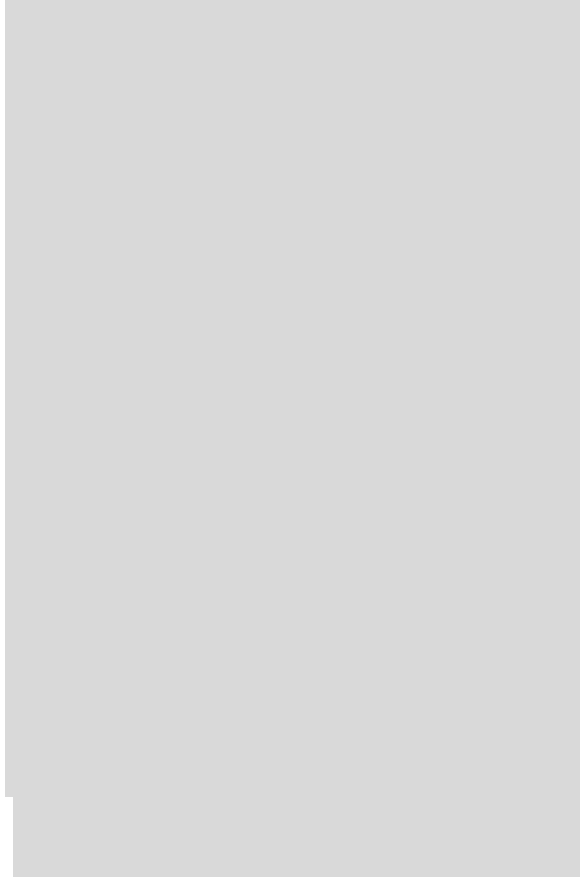
```
do forever() {
```



```
}
```

Prozess *Wagen W2*:

```
do forever() {
```

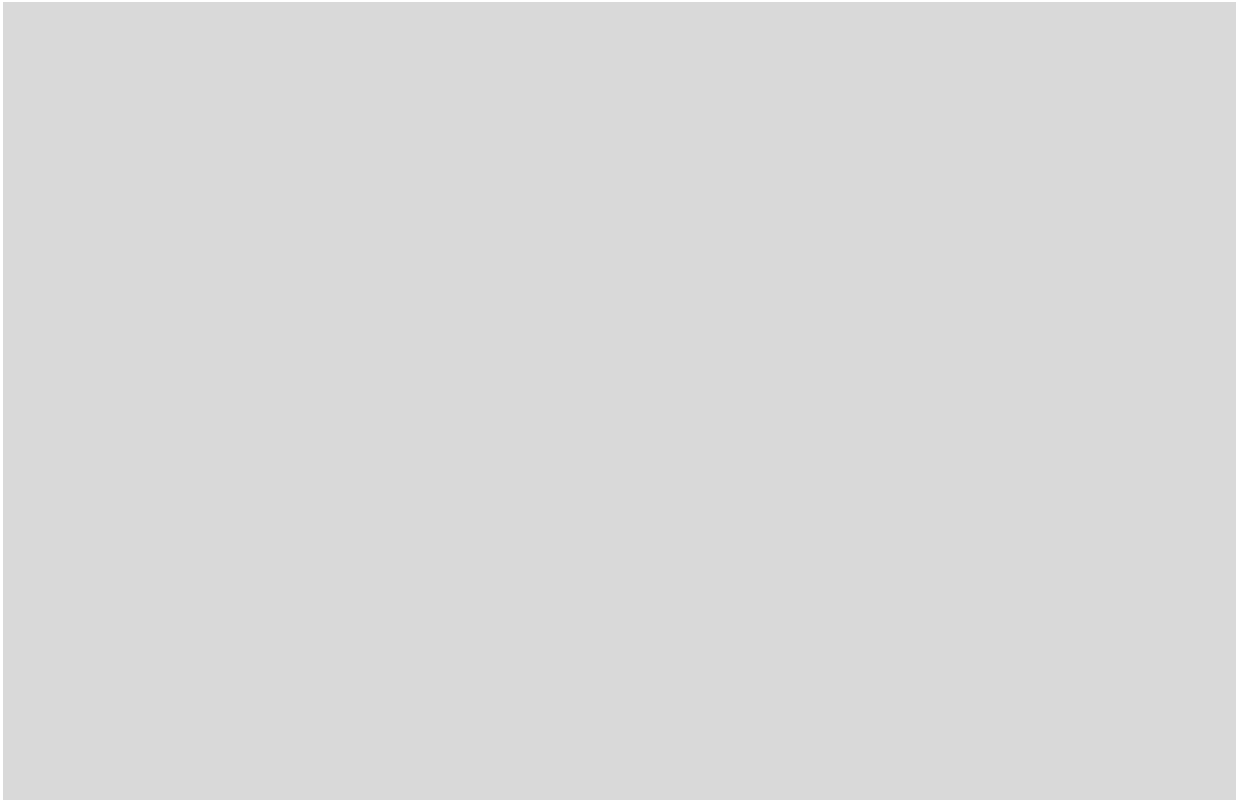


```
}
```

2 Security (20)

2.1 Security by Obscurity vs. Open Design (4)

Beschreiben Sie den Unterschied zwischen *Security by Obscurity* und *Open Design*. Geben Sie die Vor- und Nachteile an.



2.2 Verständnisfragen (4)

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

- ☐ 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 *denial-of-service* Attacke ist eine passive Attacke.

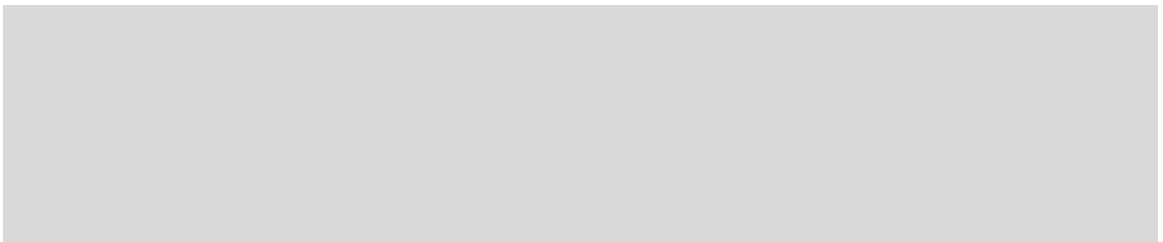
2.3 Begriffe (8)

Was versteht man unter *Confidentiality (Secrecy)*, *Integrity* und *Availability*? Erklären Sie die drei Begriffe.

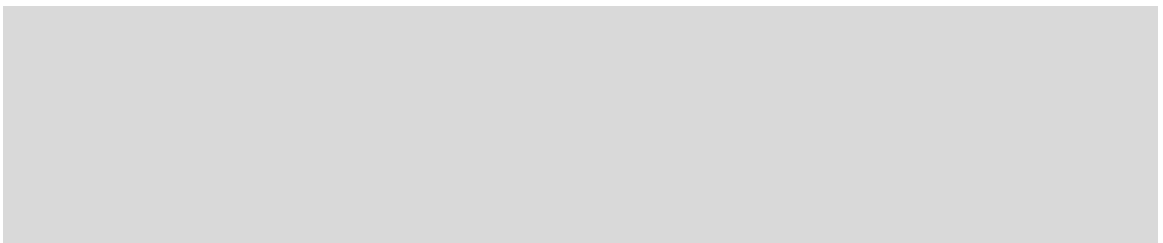
- Confidentiality:



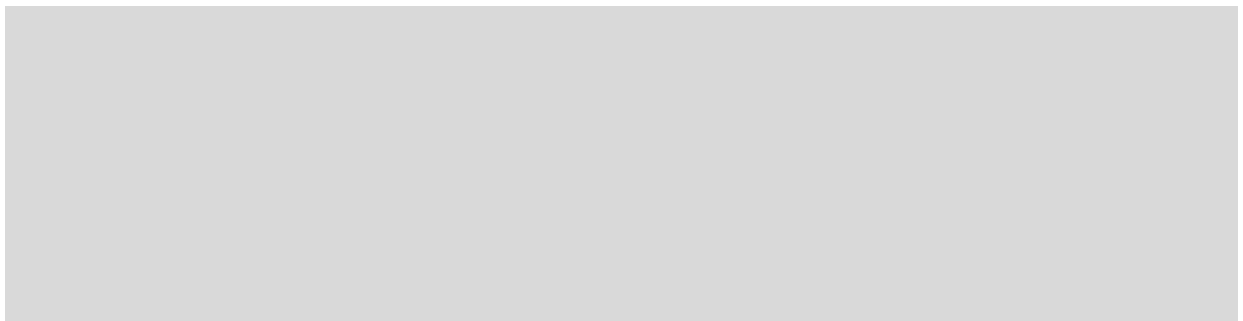
- Integrity:



- Availability:



2.4 Wie funktioniert Public Key Encryption? (4)



3 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	1	7
B	2	6
C	2	8
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 ($\sqrt[2]{2} \approx 1,41$ $\sqrt[3]{2} \approx 1,26$ $\sqrt[4]{2} \approx 1,19$ $\sqrt[5]{2} \approx 1,15$ $\sqrt[6]{2} \approx 1,12$).

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

A																		
B																		
C																		
D																		

0

5

10

15

Scheduling nach dem **EDF**-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

5

10

15

Ersatzvorlage: Scheduling nach dem -Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																		
B																		
C																		
D																		

0

5

10

15

4 BS-Abstraktionen und Prozesse (30)

Nennen Sie drei Abstraktionen, die ein Betriebssystem zur Verfügung stellt. Geben Sie für jede dieser Abstraktionen einerseits die Benutzerwahrnehmung andererseits die zur Realisierung der Abstraktion notwendigen Betriebssystemmechanismen an.

Welche Schritte muss ein Betriebssystem bei einem Process Switch (Context Switch) durchführen?

Nennen Sie die prozessspezifischen Informationen, die ein Betriebssystem im Process Control Block verwaltet.

Erklären Sie den Unterschied zwischen den Begriffen *Prozess* und *Thread*. In welcher Beziehung stehen diese beiden Konzepte?

In der Vorlesung wurden zwei Möglichkeiten der Implementierung von Threads genannt. Um welche Implementierungen handelt es sich dabei? Wie machen sich die Unterschiede in der Implementierung für Applikationen bemerkbar?

Erklären Sie, was man unter (a) einem *monolithischen Kernel* und (b) einem *Microkernel* versteht.

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(36)

2.)(21)

3.)(18)

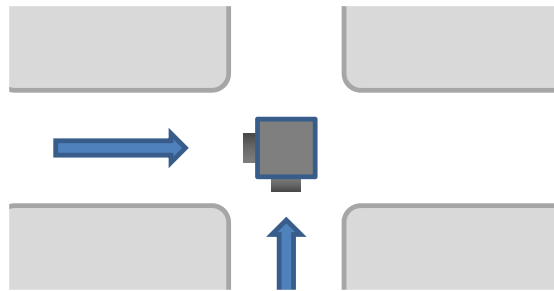
4.)(25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (36)

Der Autoverkehr einer Kreuzung von zwei Einbahnstraßen wird mit einer einfachen Ampel geregelt (siehe Skizze). Die Ampel hat auf jeder der beiden relevanten Seiten eine rote und eine grüne Lampe, um anzuzeigen, dass die Autos anhalten müssen (rot) bzw. passieren dürfen (grün). Die Autos der beiden Straßen erhalten jeweils abwechselnd eine Grünphase von 45 Sekunden. Am Anfang bzw. Ende jeder Grünphase wird ohne Verzögerung zwischen rotem und grünem Licht (bzw. umgekehrt) umgeschaltet.



Schreiben Sie drei Prozess-Templates zur Simulation des Verkehrs an der Ampel. Der Prozess *Ampel* soll die Ampel simulieren. Das Prozess-Template *Auto-W* simuliert ein von Westen und das Template *Auto-S* ein von Süden über die Kreuzung fahrendes Auto.

Die Mutual Exclusion und das korrekte Passieren der Ampel bei grün soll durch Semaphore geregelt werden. Der zyklische Prozess *Ampel* gibt die Kreuzung abwechselnd für die Autos der beiden Straßen frei und wartet zwischen den Umschaltvorgängen mit dem Funktionsaufruf *warte(45)* für 45 Sekunden.

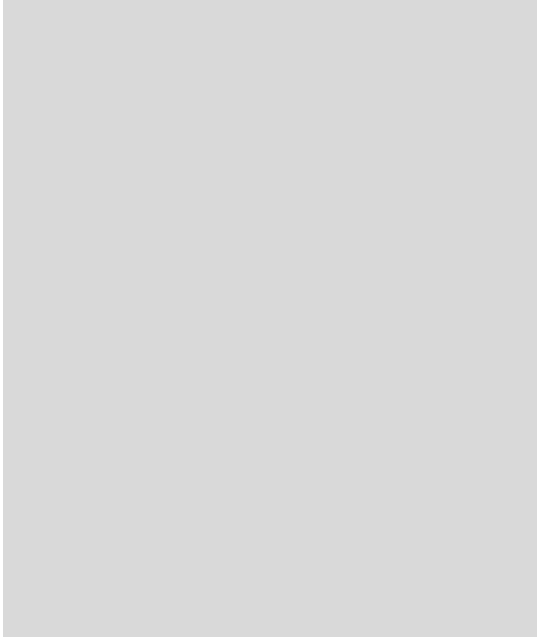
Das Template *Auto-W* simuliert die Fahrt eines von Westen kommenden Autos, das die Kreuzung passiert. Mehrere von Westen kommende Autos werden durch das Starten mehrerer Kopien von *Auto-W* simuliert. Entsprechendes gilt für das Template *Auto-S*, das ein von Süden kommendes Auto simuliert.

Zum Durchfahren der Kreuzung rufen Auto-Prozesse die Funktion *kreuzung_passieren()* auf. Dabei gilt: Ein Auto darf nur in die Kreuzung einfahren, wenn diese frei ist. Die Synchronisation muss sicherstellen, dass ein Auto nur bei grünem Ampelsignal in die Kreuzung einfährt und dass das Umschalten der Ampel nicht durch ankommende Autos verzögert wird. Schaltet die Ampel um, darf das gerade in der Kreuzung befindliche Auto die Kreu-

zung noch verlassen, dann müssen die Autos der anderen Fahrtrichtung in die Kreuzung einfahren können.

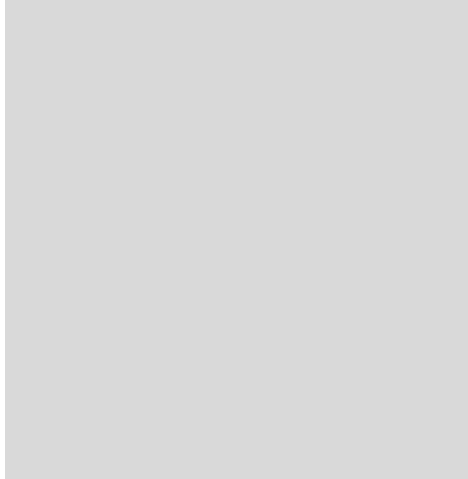
Schreiben Sie Codestücke für *Ampel*, *Auto-W* und *Auto-S* und geben Sie Initialisierungen für die benötigten Semaphore an.

Initialisierungen



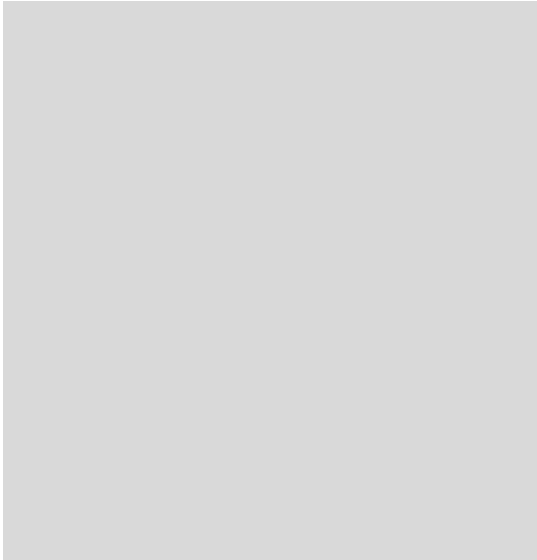
Ampel

```
forever{
```

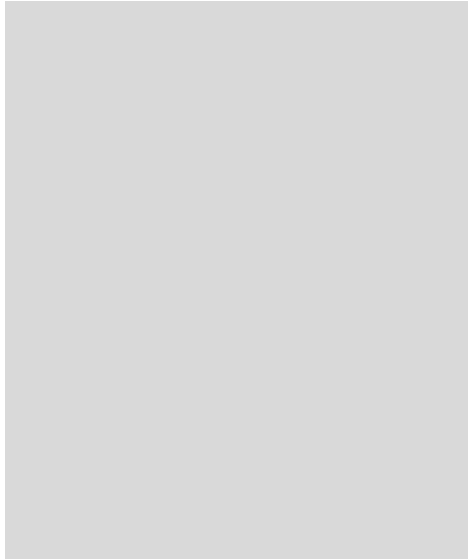


```
}
```

Auto-W



Auto-S



2 Security (21)

2.1 Begriffe (6)

Was versteht man unter *Confidentiality (Secrecy)*, *Integrity* und *Availability*? Erklären Sie die drei Begriffe.

- Confidentiality:

- Integrity:

- Availability:

2.2 Security Threats (4)

Füllen Sie folgende Tabelle derart aus, dass Sie jede angegebene Art der Bedrohung einem der Begriffe *Confidentiality (Secrecy)*, *Integrity* und *Availability* zuordnen:

Art der Bedrohung	bedroht
Interruption	
Interception	
Modification	
Fabrication	

2.3 Design Principles for Security (7)


Nennen Sie *sieben* Designprinzipien für Security!



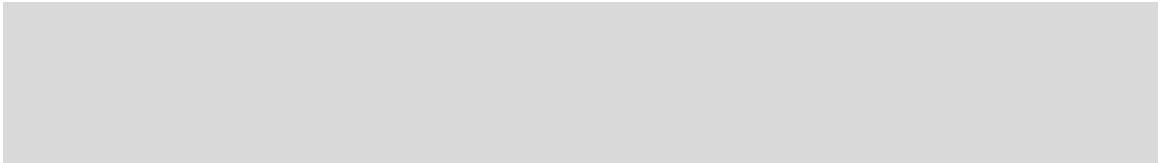
2.4 Bedrohungen durch Malware (4)

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

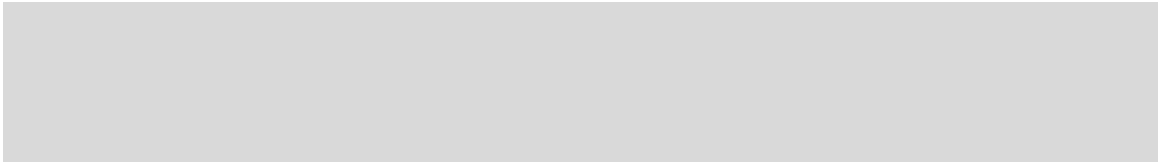
- Logic Bomb:



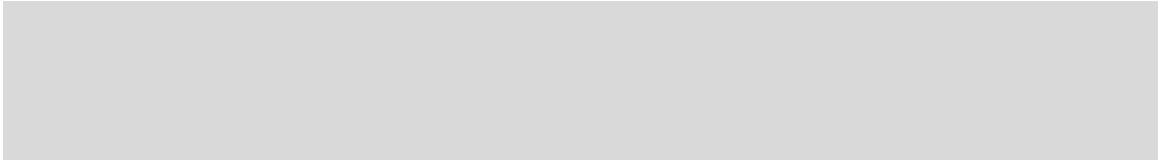
- Trojan Horse:



- Virus:



- Worm:



3 Deadlock (18)

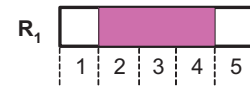
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *begin* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

```
2: get(R1)
3: ...
4: free(R1)
```



Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht möglich ist.
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.
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.
5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

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

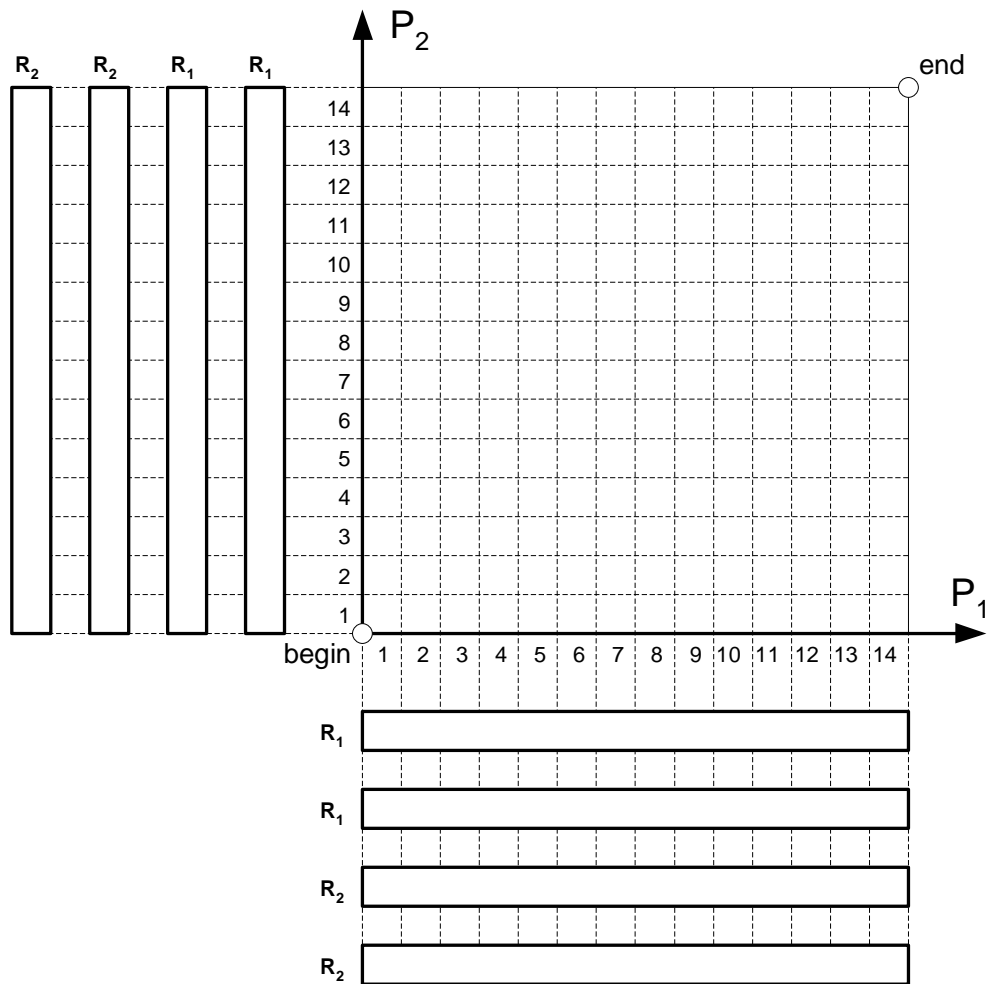
```

Program P_2 :

```

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

```



4 Memory Management (25)

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 angegebenen 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:		Zu berechnende Adressen:	
Page#	Frame#	Virtuelle Adresse	Physikalische Adresse
00010110b	0x41	0x5E41	
00001000b	0x5A	0x24AB	
01011110b	0x13	0x108A	
00100100b	0xAF		
00010001b	0x20	0x16B2	

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:		Zu berechnende Adressen:	
Entry	Frame#	Virtuelle Adresse	Physikalische Adresse
0	0x24	0x0311	
1	0xB1	0x14AC	
2	0x77	0x0512	
3	0x86		
4	0xF4	0x08A8	
5	0xCC		
6	0x01		

c) Segmentierung — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:			Zu berechnende Adressen:	
Entry	Base	Length	Virtuelle Adresse	Physikalische Adresse
0	0x2840	0x004	0x01F2	
1	0x0563	0x0FF	0x0305	
2	0x72AB	0x100	0x0010	
3	0x0300	0x010	0x04A0	
4	0x11FD	0x0F0	0x4222	
5	0x4444	0x030		
6	0x3112	0x060		
7	0x4220	0x010		

Bewertung: 1 Pluspunkt pro richtiger Lösung, 1 Punkt Abzug pro falscher Lösung.

d) Verständnisfragen (12)

Kreuzen Sie bitte die richtigen Antworten an. Achtung! Falsche Antworten werden negativ gewertet. (Bewertung: 2 Pluspunkte pro richtiger Antwort, 2 Punkte Abzug pro falscher Antwort.)

- Bei folgender Speicherverwaltungstechnik tritt sowohl *interne* als auch *externe* Fragmentierung auf.
 - ☐ fixed partitioning
 - ☐ simple segmentation
 - ☐ virtual memory with combination of paging and segmentation
 - ☐ virtual memory paging
 - ☐ dynamic partitioning
 - ☐ simple paging
- Zu welchen Effekten kann es bei *Segmentierung* kommen?
 - ☐ Internal Fragmentation
 - ☐ External Fragmentation
- In einem fehlerfreien System können zwei oder mehrere virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
 - ☐ richtig
 - ☐ falsch
- Eine virtuelle Adresse verweist immer auf eine Seite auf dem Sekundärspeicher.
 - ☐ richtig
 - ☐ falsch
- Um die externe Fragmentierung zu reduzieren, muss man die *Page Size* vergrößern.
 - ☐ richtig
 - ☐ falsch
- Ein virtueller Speicher kann sowohl mit Paging als auch mit Segmentierung implementiert werden.
 - ☐ richtig
 - ☐ falsch

KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(25)

2.)(25)

3.)(25)

4.)(25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (25)

Gegeben sei ein System mit einer beliebigen Anzahl von parallel laufenden Prozessen vom Typ A und vom Typ B. Weiters ist in dem System ein Gerät vom Typ G 4-fach vorhanden. Die zu produzierende und konsumierende Ressource vom Typ R ist initial 0-mal vorhanden. Ein Prozess vom Typ A benötigt für die Abarbeitung einer Iteration exklusiven Zugriff auf 3 Geräte vom Typ G und produziert dabei eine Ressource vom Typ R. Ein Prozess vom Typ B benötigt für die Abarbeitung exklusiven Zugriff auf ein Gerät vom Typ G und konsumiert zusätzlich zwei Ressourcen vom Typ R.

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

Verwenden Sie folgende Funktionen für Operationen auf Semaphoren:

$\text{initS}(\text{Sem}, \text{init})$ legt einen Semaphor mit dem angegebenen symbolischen Namen *Sem* an und initialisiert ihn mit der Zahl *init*.

$\text{P}(\text{Sem})$ implementiert ein *wait* auf dem Semaphore, und

$\text{V}(\text{Sem})$ implementiert ein *signal* auf dem Semaphore.

a) Initialisierungen

Initialisieren Sie die notwendigen Semaphore.

b) Entwerfen Sie die Prozesse vom Typ A und B

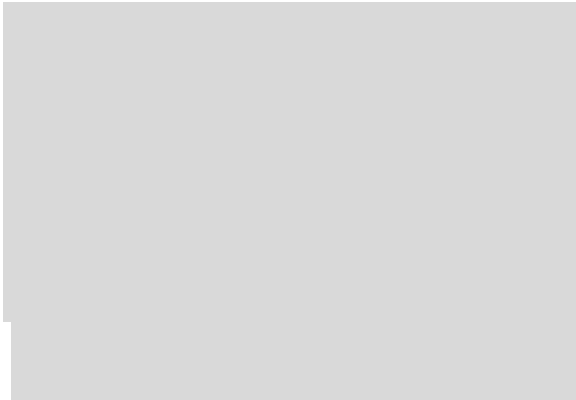
Die Prozesse A und B benötigen in der Funktion *do_the_work_A()* bzw. *do_the_work_B()* jeweils exklusiven Zugriff auf die jeweilige Anzahl an Geräten G.

Prozess Typ A:

```
do forever() {
```



```
do_the_work_A();
```



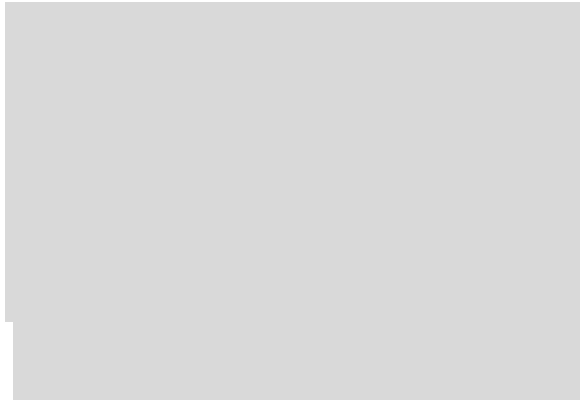
```
}
```

Prozess Typ B:

```
do forever() {
```



```
do_the_work_B();
```



```
}
```

2 Scheduling (25)

2.1 Uniprocessor Rate Monotonic Scheduling (10)

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	7
B	2	6
C	1	9
D	1	7

Ermitteln Sie für dieses Taskset die *notwendige* und die *hinreichende* Bedingung für das *Rate Monotonic Scheduling* (RMS) Verfahren. Berechnen Sie die **Zahlenwerte** überschlagsmässig ($\sqrt[2]{2} \approx 1,41$ $\sqrt[3]{2} \approx 1,26$ $\sqrt[4]{2} \approx 1,19$ $\sqrt[5]{2} \approx 1,15$ $\sqrt[6]{2} \approx 1,12$).

Ist die notwendige Bedingung erfüllt? ☐ Ja ☐ Nein

Ist die hinreichende Bedingung erfüllt? ☐ Ja ☐ Nein

Versuchen Sie das Taskset mit dem RMS 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 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

A																	
B																	
C																	
D																	

0

5

10

15

Ersatzvorlage: Scheduling nach dem RMS-Verfahren:

Erfolgreich: ☐ Ja ☐ Nein

A																	
B																	
C																	
D																	

0

5

10

15

2.2 Uniprocessor Round Robin Scheduling (7)

Scheduling nach dem Round Robin Verfahren: Versuchen Sie das Taskset nach dem *Round Robin* (RR) Verfahren zu schedulen. Fügen Sie zuerst *Tasks mit abgelaufener Zeitscheibe vor neu ankommenden Tasks* in die Ready Queue. Verwenden Sie die nachstehenden Vorlagen. Tragen Sie in die Vorlage die aktiven Taskzeiten ein und bezeichnen Sie deutlich (mit einem Pfeil →) die Arrival Time der Tasks. Eine Zeitscheibe entspricht einer *Service Time* von eins. Eine der Vorlagen dient als Ersatz, streichen Sie gegebenenfalls eine falsch ausgefüllte Vorlage deutlich durch.

Process	Arrival Time	Service Time
P1	0	5
P2	1	7
P3	3	4
P4	6	2
P5	10	1

Scheduling nach dem **RR**-Verfahren:

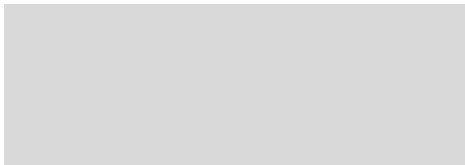
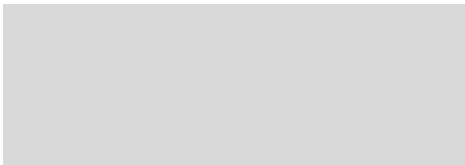
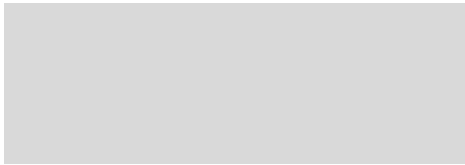
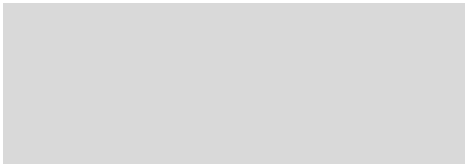
P1																		
P2																		
P3																		
P4																		
P5																		
	0				5					10					15			

Ersatzvorlage: Scheduling nach dem RR-Verfahren:

P1																		
P2																		
P3																		
P4																		
P5																		
	0					5					10					15		

2.3 Scheduling allgemein (8)

Welche Scheduling-Kriterien kennen Sie? Vervollständigen Sie nachstehende Tabelle.

	User-oriented	System-Oriented
Performance		
Other		

3 Deadlock (25)

Deadlock-Bedingungen (6)

Erklären Sie die Deadlock-Bedingungen *Hold and Wait* und *No Preemption*.

Hold and Wait:

No Preemption:

Banker's Algorithmus (6)

Wozu wird der Banker's Algorithmus verwendet?

Was bedeutet es, wenn der Banker's Algorithmus einen *Safe State* bzw. einen *Unsafe State* diagnostiziert?

Safe State:

Unsafe State:

Deadlock-Erkennung (13)

In einem Computersystem laufen fünf Prozesse, die gemeinsame Ressourcen aus vier Ressourcenkategorien verwenden. Die vorhandenen Ressourcen sind durch den Vektor $R = (6, 5, 9, 6)$ gegeben. Die unten gegebenen Matrizen beschreiben die aktuellen Ressourcenanforderungen und -allokation der fünf Prozesse. Führen Sie den Algorithmus zur Deadlock-Erkennung durch, um festzustellen, ob im gegebenen System ein Deadlock vorliegt.

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 1 & 3 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 4 & 2 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad V = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right)$$

$$Q = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right) \quad A = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right) \quad V = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right)$$

$$Q = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right) \quad A = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right) \quad V = \left(\begin{array}{c} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{array} \right)$$

$$Q = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad A = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad V = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right)$$

$$Q = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad A = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad V = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right)$$

$$Q = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad A = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right) \quad V = \left(\begin{array}{c} \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \\ \text{[Bar]} \end{array} \right)$$

Liegt ein Deadlock vor? Falls ein Deadlock vorliegt, beschreiben Sie diesen.

4 Filemanagement (25)

4.1 Allokation (12)

Welche Strategien zur Block-Allokation kennen Sie zur Implementierung einer Datei im sekundären Speicher? Erklären Sie diese und erläutern Sie Vor- und Nachteile!



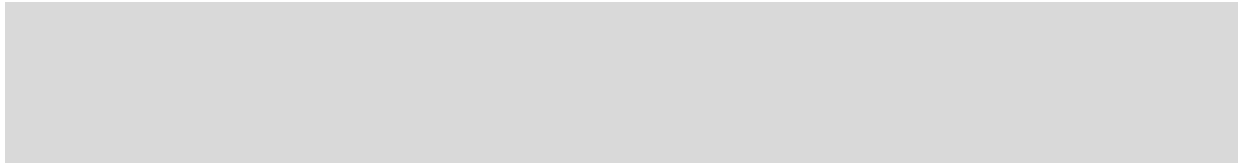


4.2 Datei-Organisation und -Zugriff (10)

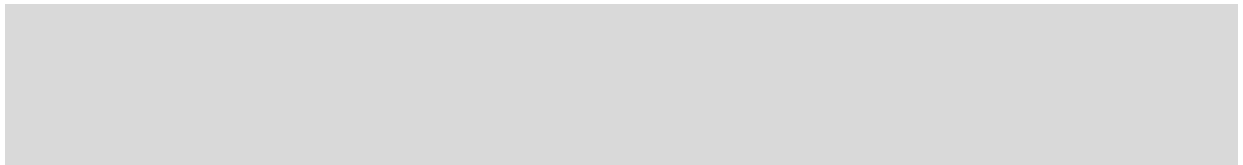
Nennen und erklären Sie fünf fundamentale Arten der Datei-Organisation!

4.3 Disk- und File-System-Layout (3)

In welchem Sektor der Disk befindet sich der Master Boot Record (MBR) und welche Informationen enthält er?



Aus welcher Partition einer Disk wird gebootet?



KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(30)

2.)(20)

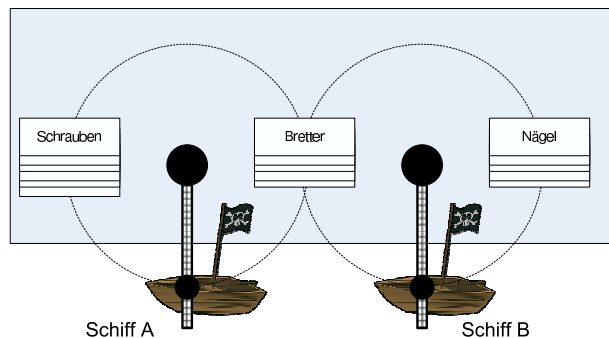
3.)(25)

4.)(25)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Eine Reederei besitzt zwei Frachtschiffe, *Schiff A* und *Schiff B*. Beide Frachtschiffe haben einen eigenen Liegeplatz mit dazugehörigem Verladekran. Das *Schiff A* transportiert *Schrauben* und *Bretter*, das *Schiff B* transportiert *Bretter* und *Nägel*.

Die Schrauben, Bretter und Nägel sind in Containern verpackt, welche auf dem Hafen gestapelt stehen.

Das Beladen der Schiffe läuft nach folgenden Regeln ab:

- Schiff A soll mit einem Container voller Schrauben und einem Container voller Bretter beladen werden. Schiff B soll mit einem Container voller Bretter und mit einem Container voller Nägel beladen werden. Die Schrauben bzw. die Nägel müssen vor den Brettern geladen werden.
- Zum Beladen werden die Kräne verwendet.
- Der Kran A, welcher Schiff A zugeordnet ist kann über die Container mit den Schrauben, über die Container mit den Brettern und über das Schiff A bewegt werden.
- Der Kran B, welcher Schiff B zugeordnet ist, kann über die Container mit den Brettern, über die Container mit den Nägeln und über das Schiff B bewegt werden.
- Um ein Zusammenstoßen der Kräne zu verhindern dürfen niemals beide Kräne gleichzeitig über die Container mit den Brettern bewegt werden.
- Wenn ein Schiff fertig beladen ist, soll es sofort mit seiner Auslieferung beginnen.

- Mit dem Beladen eines Schiffes kann natürlich erst dann wieder begonnen werden wenn das Schiff zurückgekehrt ist. Ein Kran kann allerdings einen Container schon vom Stapel nehmen bevor das Schiff wieder zurückgekehrt ist.

Synchronisieren Sie den Arbeitsablauf der beiden Kräne und der beiden Schiffe mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`bewege(Richtung)` Bewegt den Kran um 90 Grad in die gewünschte *Richtung*. Als Richtung kann Uhrzeigersinn (UZS) oder Gegen_Uhrzeigersinn (GUZS) angegeben werden.

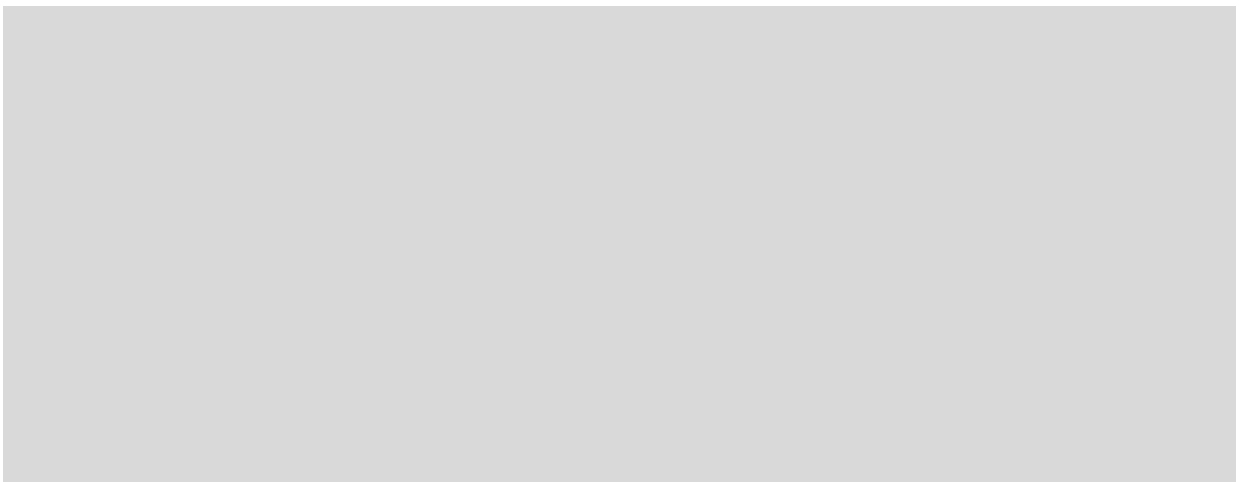
`nimm()` Lässt den Kran einen Container von dem Stapel über dem er sich gerade befindet aufnehmen. Funktioniert nur, wenn sich der Kran über einem Containerstapel befindet.

`lege_ab()` Mittels dieser Funktion legt ein Kran den Container, den er gerade hält, an der aktuellen Position ab

`lieferung()` Mit dieser Funktion liefert ein Schiff seine Ware aus, und kehrt anschließend wieder zurück.

a) Initialisierungen (8)

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** des Systems entspricht dem obigem Bild. Beide Schiffe sind unbeladen und beide Kräne stehen über den jeweiligen Schiffen.



b) (14)

Entwerfen Sie je einen Prozess für *Kran A* und *Kran B*.

Prozess *Kran A*:

```
do forever() {
```

Prozess *Kran B*:

```
do forever() {
```

```
}
```

```
}
```

c) (8)

Entwerfen Sie die Prozesse für *Schiff A* und *Schiff B*:

Prozess *Schiff_A*:

```
do forever() {
```

```
    // ...  
}
```

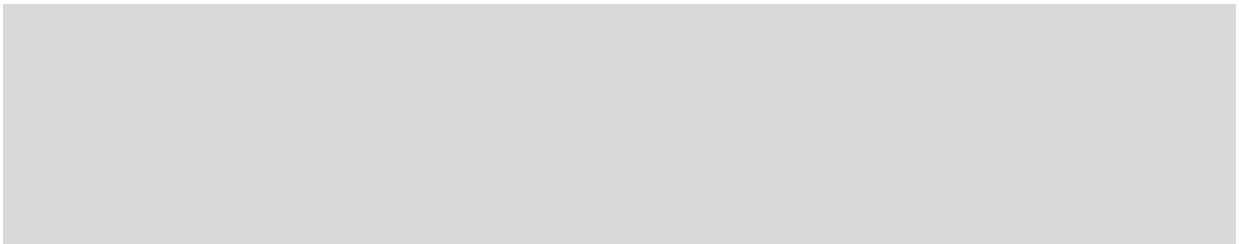
Prozess *Schiff B*:

```
do forever() {
```

```
    // ...  
}
```

2 Memory Management (20)

Was ist der Unterschied zwischen lokalem und globalem Page Replacement?



Die Page Table eines kleinen Computersystems ist als Inverted Page Table (IPT) mit acht Einträgen realisiert. Beim Auftreten von Page Faults wird in diesem Computersystem der Clock Algorithmus als globale Page Replacement Strategie eingesetzt.

Im zu lösenden Beispiel ist eine Ausgangsbelegung der Page Table (links), sowie eine Folge von Zugriffen auf den virtuellen Speicher, charakterisiert durch Prozessnummer (PID), adressierte Seite (page) und Offset, gegeben. Simulieren Sie die Ausführung der Zugriffsfolge von links nach rechts und tragen Sie für jeden Zugriff auf den virtuellen Speicher (a) die entsprechende physikalische Adresse, (b) den Zustand der IPT und (c) den Wert des Replacement Pointers (rep. pointer) für den Clock Algorithmus nach dem Speicherzugriff an (Sie brauchen bei jedem Schritt nur die Werte in der IPT anzugeben, die sich beim aktuellen Zugriff geändert haben).

#vielleicht
beim lesen
wird 0 gesetzt
look book

	PID	page	offset	phys. address
	1	0	00a0	page fault
	1	0	00a2	0x100a2
	1	4	0002	0x60002
	2	3	fffe	0x2fffe
	2	4	0000	page fault

	PID	page	use
0	1	5	1
1	3	1	0
2	2	3	0
3	2	1	0
4	1	2	1
5	1	3	1
6	1	4	1
7	3	3	1

	PID	page	use
0			0
1	1	0	1
2			0
3			0
4			1
5			0
6			0
7			0

	PID	page	use
0			0
1	1	0	1
2			0
3			0
4			1
5			0
6			0
7			0

	PID	page	use
0			0
1			1
2			0
3			0
4			1
5			0
6	1	4	1
7			0

	PID	page	use
0			0
1			1
2	2	3	1
3			0
4			1
5			0
6			1
7			0

	PID	page	use
0			
1			
2			0
3	2	4	1
4			
5			
6			
7			

	rep. pointer
	5
	2
	2
	2
	2
	2
	4

3 Deadlock (25)

Im folgenden Beispiel konkurrieren 5 Prozesse um 4 verschiedene Ressourcen. Zum betrachteten Zeitpunkt sind 2 Ressourcen vom Typ 1, und keine Ressource des Typs 2,3 oder 4 verfügbar.

Die *allocation Matrix* A gibt die gegenwärtig von jedem der 5 Prozesse allokierten Ressourcen an, die Matrix Q beschreibt die momentan von den Prozessen angeforderten und noch nicht gewährten Ressourcen (in den Matrizen repräsentiert jede Spalte eine Ressource und jede Zeile einen Prozess).


Kann in dieser Situation ein Deadlock vorliegen ? Führen Sie den *deadlock detection* Algorithmus durch, um diese Frage zu beantworten.

$$A = \begin{pmatrix} 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 3 & 3 & 0 & 4 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 2 & 0 \end{pmatrix}$$

Ein Deadlock kann aufgehoben werden, in dem den Prozessen einige Ressourcen entzogen werden. Wie kann im vorangegangenen Beispiel durch das Entziehen genau einer allokierten Ressource sichergestellt werden, dass kein Deadlock mehr vorliegt ?



Welche anderen Strategien zur Aufhebung eines Deadlocks kennen Sie ?



Beschreiben Sie eine Möglichkeit, um das Auftreten der notwendigen Deadlock Bedingung “*circular wait*” auszuschliessen (*direct deadlock prevention*). Die gleichzeitige Nutzung unterschiedlicher Ressourcen soll weiterhin möglich sein.



Welche weiteren Bedingungen sind für einen Deadlock notwendig ? Erklären Sie deren Bedeutung.

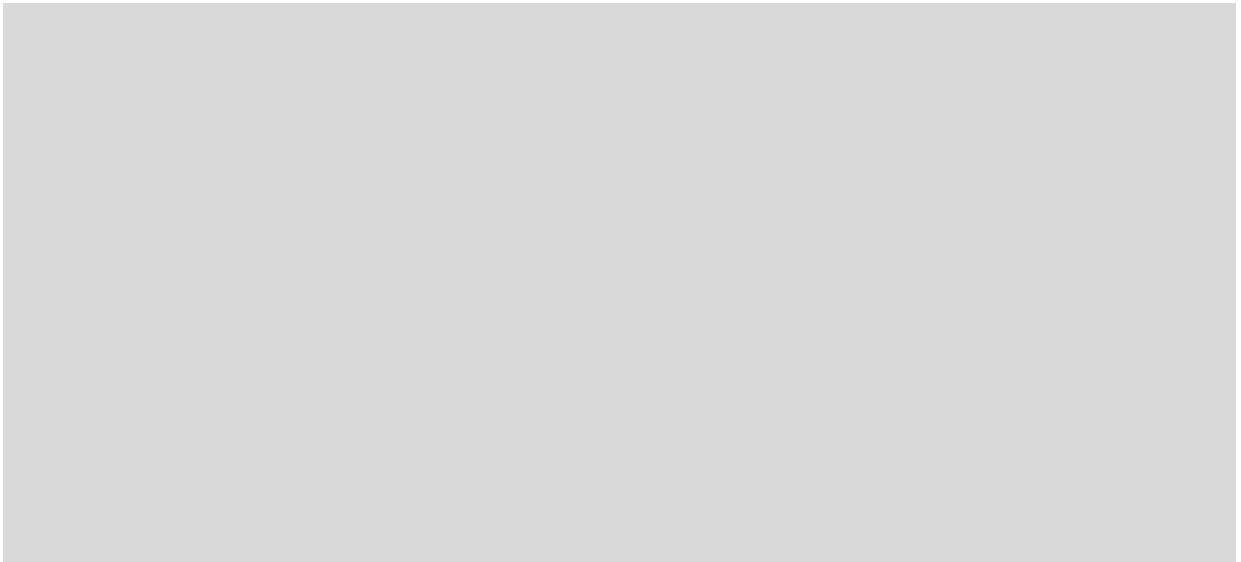
4 Input–Output (25)

Nennen Sie die beiden widersprechenden Hauptziele, die bei der Realisierung eines I/O-Systems für ein General Purpose Betriebssystem verfolgt werden. Geben Sie für jedes der beiden Ziele an, welche Auswirkungen es auf das Design des Betriebssystems hat.

Beschreiben Sie die hierarchische Ebenenstruktur, die bei der Realisierung von I/O-Funktionen Anwendung findet. Geben Sie Name und Funktion für jede Ebene an.

Was sind die Vor- und Nachteile des Pufferens von I/O-Anfragen?

Nennen Sie zwei Verfahren, die beim Disk-Scheduling zur “Minimierung der Seek Time” verwendet werden (nicht FIFO, LIFO, und Prioritätsverfahren). Beschreiben Sie für jedes der von Ihnen genannten Verfahren kurz Funktionsweise, Vor- und Nachteile.



KNr.

MNr.

Zuname, Vorname

Ges.)(100)

1.)(22)

2.)(25)

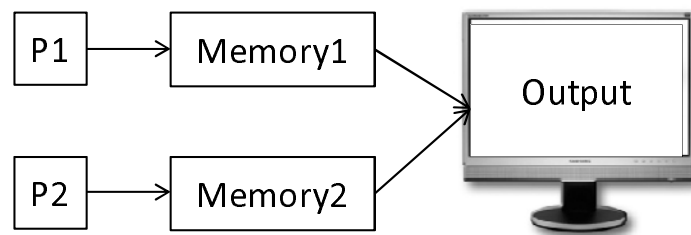
3.)(30)

4.)(23)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (25)



In einem Chatprogramm gibt es 2 Prozesse (P1 bzw. P2), die die eingehenden Nachrichten in jeweils einen Speicher (Memory1 und Memory2) schreiben. Sobald Daten in einem der beiden Speicher oder in beiden Speichern zum Lesen bereitgestellt sind, sollen diese Daten gelesen und ausgegeben werden (Output). Es darf erst wieder in einen Speicher geschrieben werden, nachdem die Daten gelesen und ausgegeben wurden.

Synchronisieren Sie den Arbeitsablauf der beiden Schreibprozesse und des Ausgabeprozesses mittels **Semaphoren**. Achten Sie auf maximale Parallelität und vermeiden Sie Starvation der Schreibprozesse. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Integrieren Sie in Ihre Lösung eine Variable, die angibt in welchem der beiden Speicher Daten zum Lesen bereitgestellt sind.

Allgemeine zu verwendende Funktionen:

`initS(Semaphor, init)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

Zu verwendende Funktionen für die Schreibprozesse:

`write_mem1()` Mit dieser Funktion wird in den Speicher 1 geschrieben.

`write_mem2()` Mit dieser Funktion wird in den Speicher 2 geschrieben.

Zu verwendende Funktionen für den Ausgabeprozess:

`read_mem1()` Liest die Daten vom Speicher 1 und gibt sie aus.

`read_mem2()` Liest die Daten vom Speicher 2 und gibt sie aus.

a) Initialisierungen (5)

Initialisieren Sie die notwendigen Semaphoren und die geforderte Variable.

d) Schreibprozesse (10)

Entwerfen Sie die beiden Schreibprozesse $P1$ und $P2$:

Prozess $P1$:

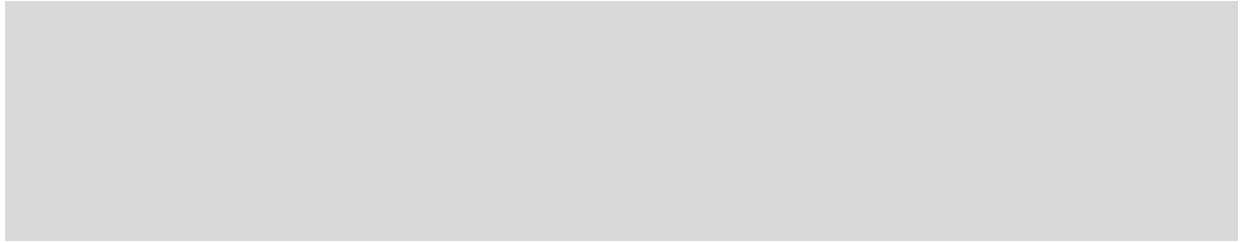
Prozess $P2$:

do forever() {

do forever() {

b) Ausgabeprozess (10)

Definition von Hilfsvariablen:

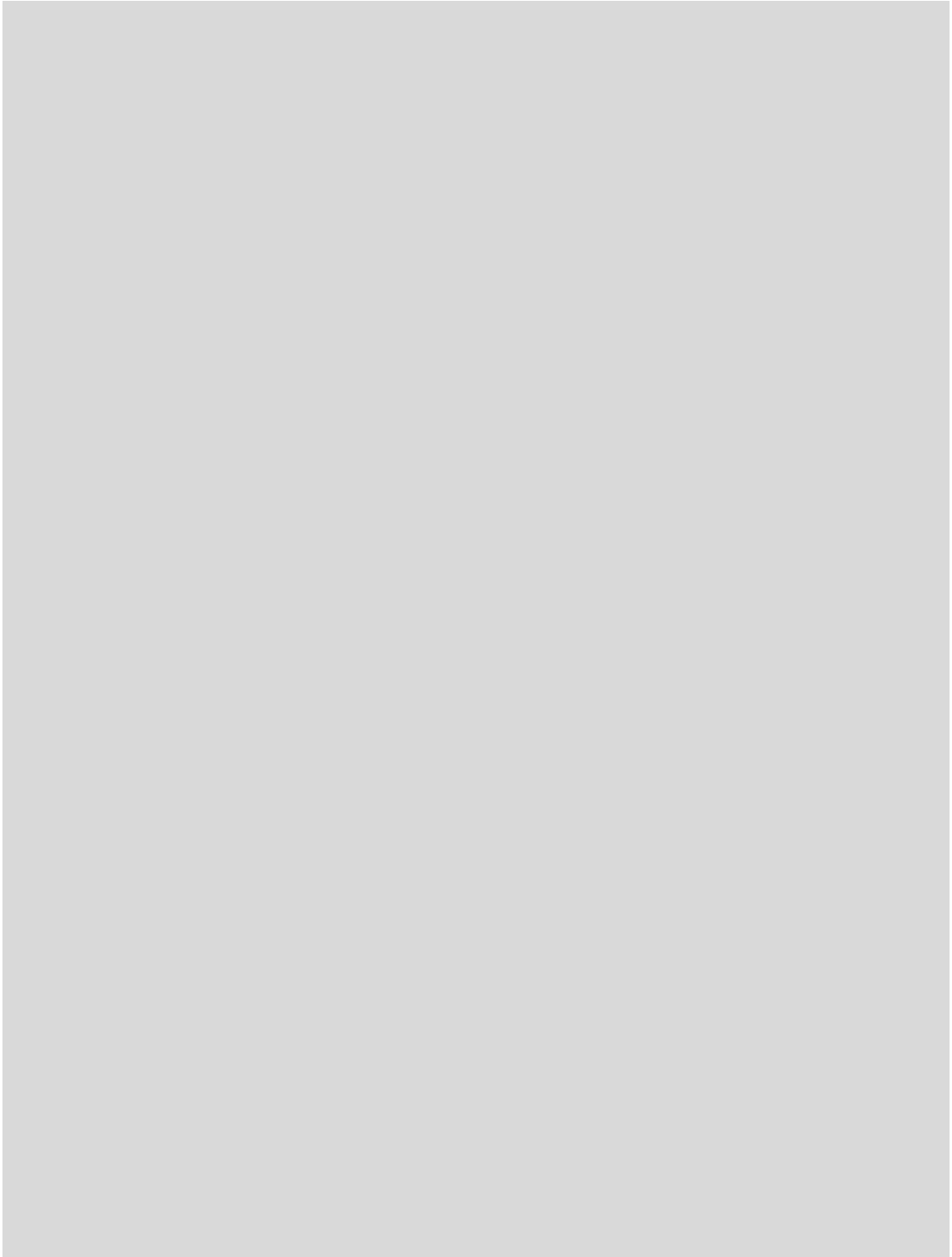


Entwerfen Sie den *Ausgabeprozess*:

Prozess *Output*:

```
do forever() {
```





}

2 Scheduling (25)

RR und SRT Scheduling (10)

Schedulen Sie das nebenstehende Taskset mit den Verfahren *Round Robin* und *Shortest Remaining Time* (SRT). Der Overhead für den Taskwechsel ist vernachlässigbar.

Task	Arrival Time	Service Time
P1	0	5
P2	2	2
P3	3	6
P4	8	4
P5	10	2

Beim Round Robin Verfahren soll die *Zeitscheibenlänge* 2 verwendet werden. Fügen Sie zuerst Tasks mit abgelaufener Zeitscheibe vor neu ankommenden Tasks in die Ready Queue ein. Wird ein Task beendet, bevor seine Zeitscheibe abgelaufen ist, wird sofort einem neuen Task die Ressource für 2 Zeiteinheiten zugewiesen.

Beim *SRT* Verfahren ist zu beachten, dass Tasks zum jeweiligen Ankunftszeitpunkt sofort gescheduled werden können. Stehen mehrere Tasks mit der selben verbliebenen Service Time zur Auswahl, soll der Task mit der niedrigsten Nummer gewählt werden (also beispielsweise *P4* vor *P5*).

Verwenden Sie die nachstehenden Vorlagen. Markieren Sie in die Vorlage zu jedem Zeitpunkt den jeweils aktiven Task. Bezeichnen Sie weiters deutlich die *arrival time* jedes Tasks (mit einem Pfeil \rightarrow) und den Zeitpunkt, wenn ein Task die gesamte benötigte *service time* konsumiert hat (\leftarrow). Eine der Vorlagen dient als Ersatz, streichen Sie gegebenenfalls eine falsch ausgefüllte Vorlage deutlich durch.

Scheduling nach dem **RR**-Verfahren:

P1	\rightarrow										\leftarrow									
P2																				
P3																				
P4																				
P5																				
	0						5					10					15			

Scheduling nach dem **SRT**-Verfahren:

P1	\rightarrow										\leftarrow									
P2																				
P3																				
P4																				
P5																				
	0							5					10				15			

P1	→																		
P2																			
P3																			
P4																			
P5																			
	0				5				10				15						

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	1	6
B	2	8
C	1	9
D	3	11
E	1	13

Scheduling nach dem **RMS**-Verfahren:

A																	
B																	
C																	
D																	
E																	
	0				5					10					15		

Erfolgreich: ☐ Ja ☐ Nein

Erstversuche: Berechnung nach dem LEWIS-Verfahren:																		Erfolgreich: <input type="radio"/> Ja <input type="radio"/> Nein	
A																			
B																			
C																			
D																			
E																			
	0				5					10					15				

2.2 Verständnisfragen (6)

Werden beim Round-Robin IO-intensive oder CPU-intensive Prozesse benachteiligt? Beschreiben Sie eine Variante des Round-Robin Verfahrens, die dieses Problem zu umgehen zu versucht.

Was versteht man unter dem Begriff Starvation? Nennen Sie ein Scheduling Verfahren, bei dem es zu Starvation kommen kann.

Bei welchen der folgenden Scheduling Verfahren muss die Service Time der Tasks bekannt sein (beziehungsweise geschätzt werden)? Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!

- ☐ Ja ☐ Nein Virtual Round Robin
- ☐ Ja ☐ Nein Earliest Deadline First
- ☐ Ja ☐ Nein Highest Response Ratio Next
- ☐ Ja ☐ Nein Rate Monotonic Scheduling
- ☐ Ja ☐ Nein Shortest Process Next
- ☐ Ja ☐ Nein Feedback Scheduling

3 Deadlock (23)

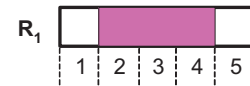
Gegeben sind zwei Prozesse, P_1 und P_2 , die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *begin* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

2: `get(R1)`
3: ...
4: `free(R1)`



Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht möglich ist.
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.
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.
5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

... mit 'A', von welchem aus der Punkt *end* und welcher vom Punkt *begin* erreichbar ist.

... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist.

... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern ein solcher Punkt vorhanden ist.

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

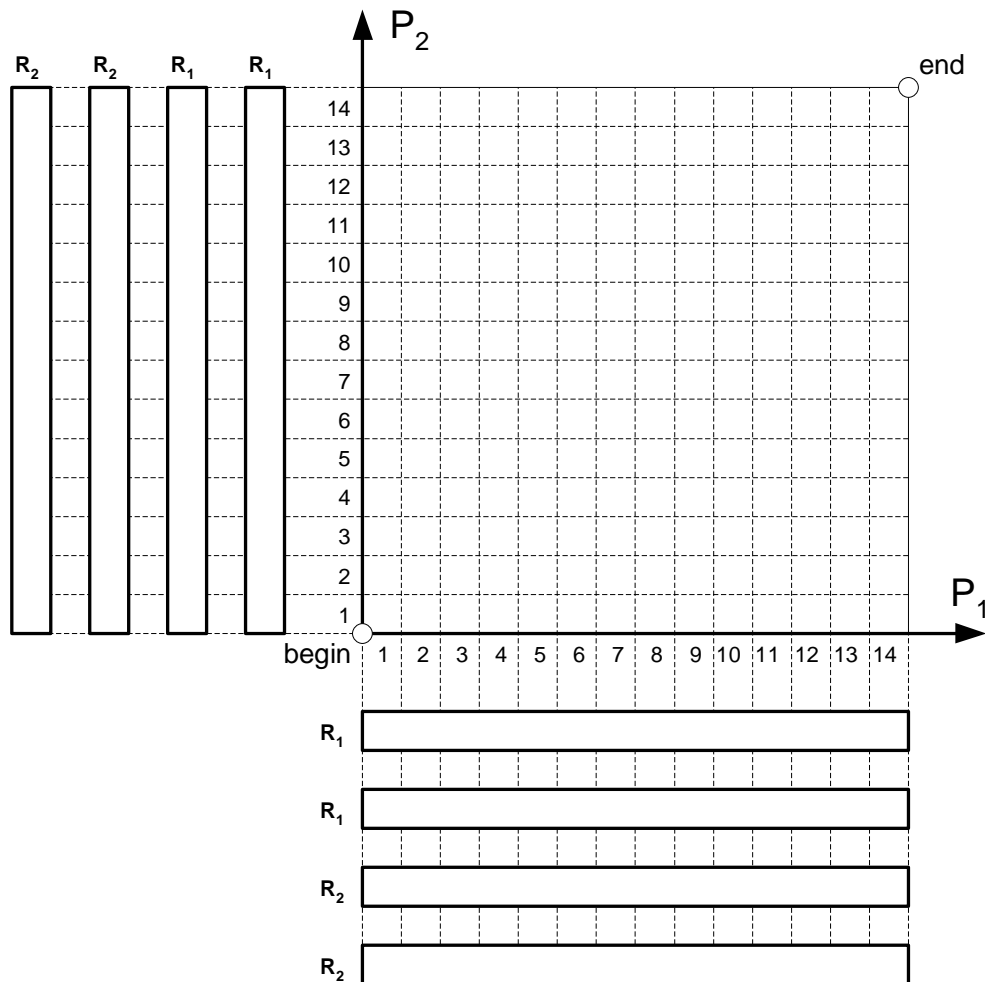
```

Program P_2 :

```

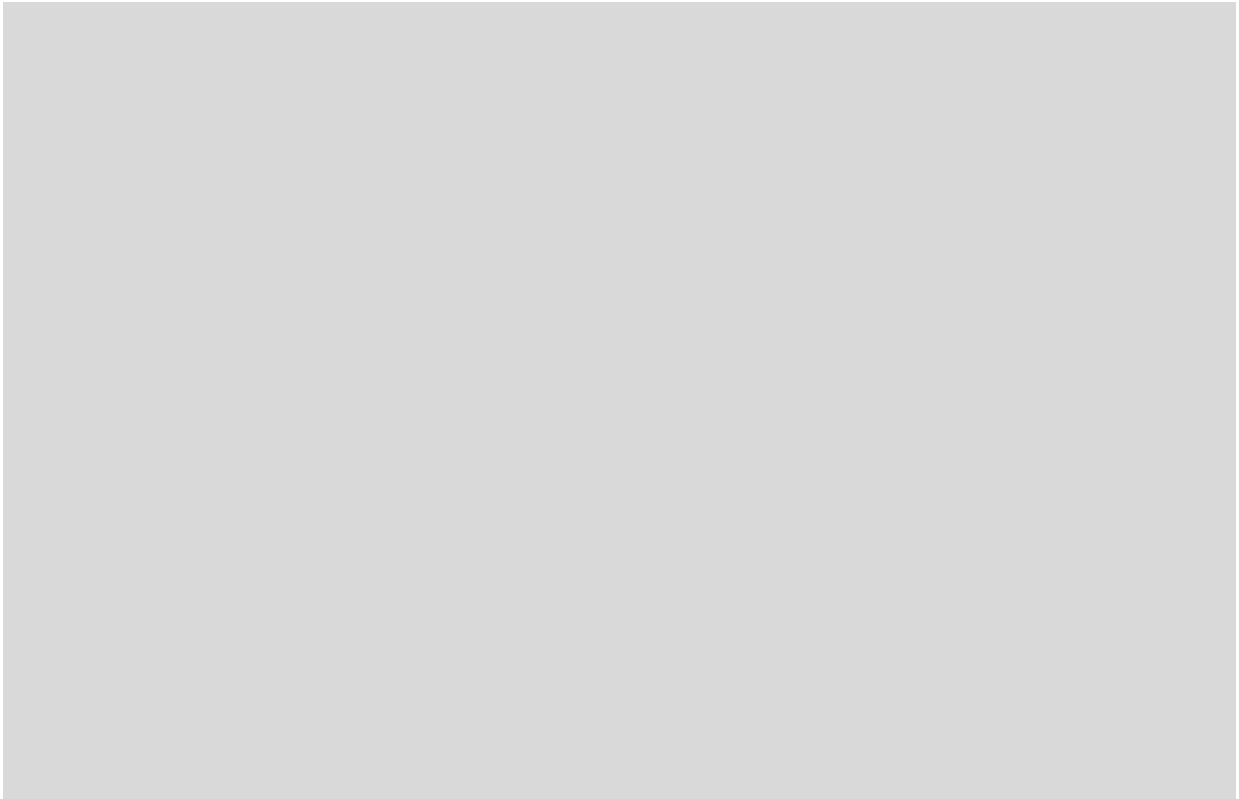
1: a=54;
2: get(R1);
3: get(R2);
4: get(R2);
5: get(R1);
6: c=a+b;
7: free(R2);
8: a=2*b*c;
9: free(R1);
10: free(R2);
11: a=b*5;
12: free(R1);
13: b=5+d;
14: return;

```

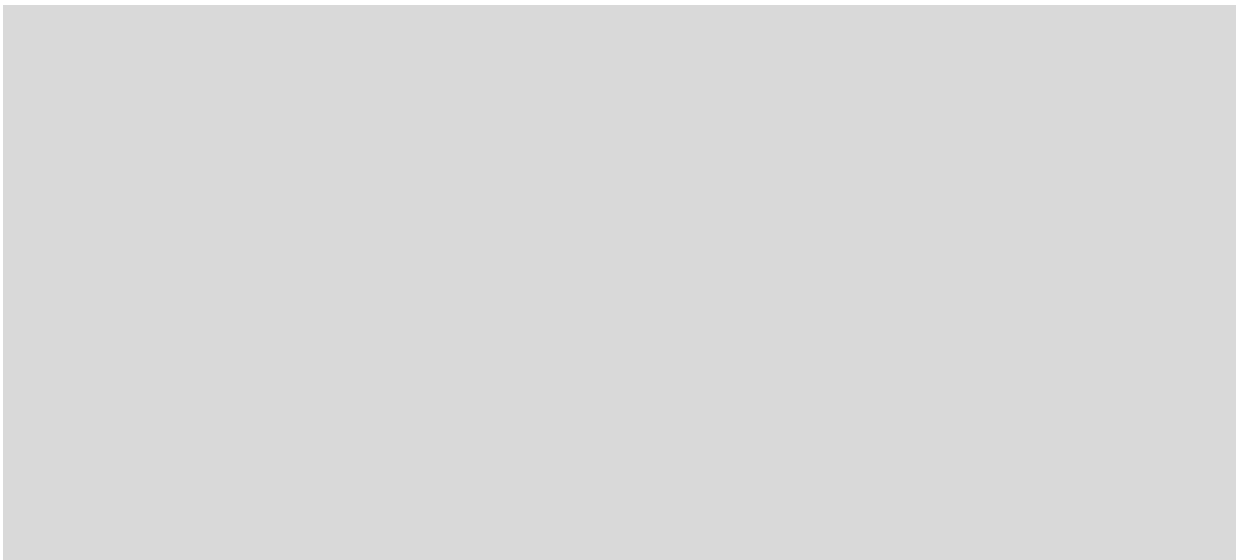


4 Betriebssysteme, Prozesse und Threads (23)

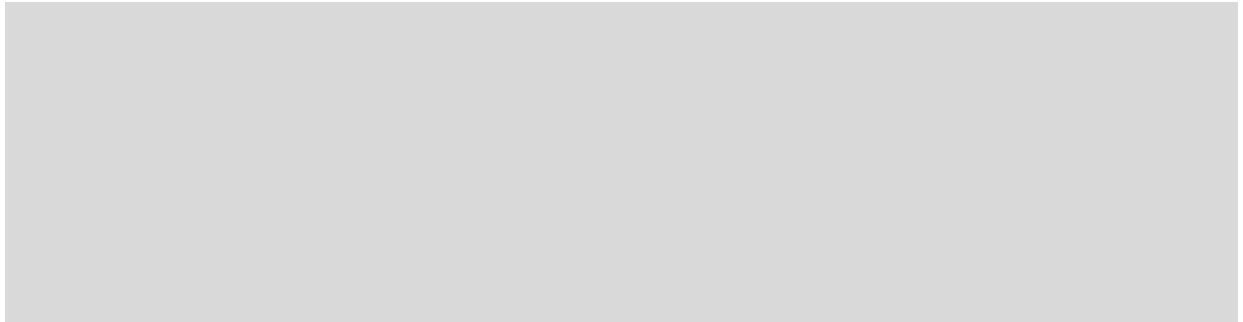
Was versteht man unter einem Microkernel-Betriebssystem? Welche Services stellt ein Microkernel zur Verfügung? Welche Eigenschaften zeichnen ein solches Betriebssystem aus?



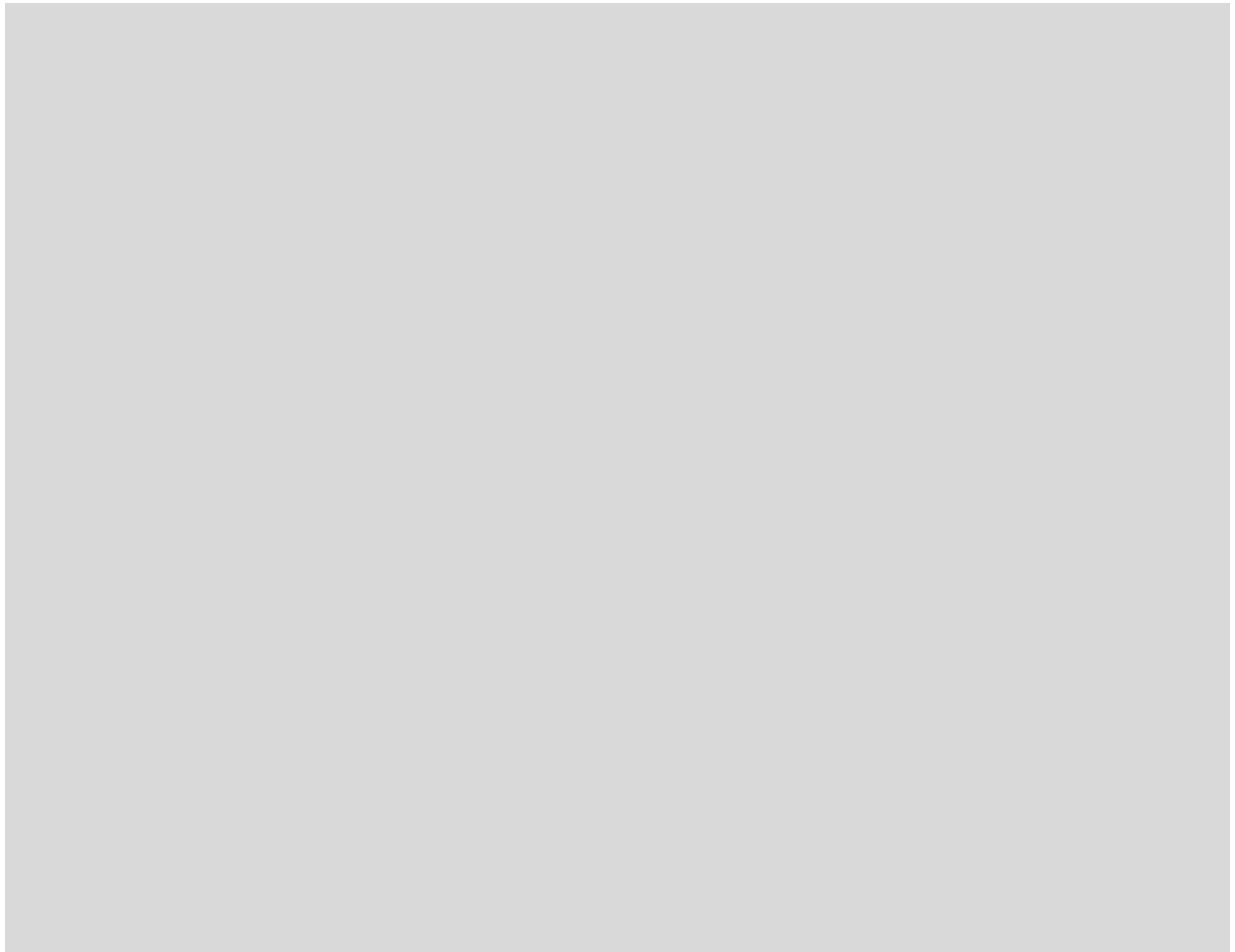
Was ist ein *Mode Switch*? Was ist ein *Process Switch*? Erklären Sie die beiden Begriffe. Geben Sie an, wozu diese Switches benötigt werden und erläutern Sie, welcher Zusammenhang zwischen Mode Switch und Process Switch besteht.



Nennen Sie die drei Kategorien von Ereignissen, mit deren Hilfe das Betriebssystem die Kontrolle über das Computersystem übernimmt. Geben Sie für jede der Kategorien ein Beispiel an.



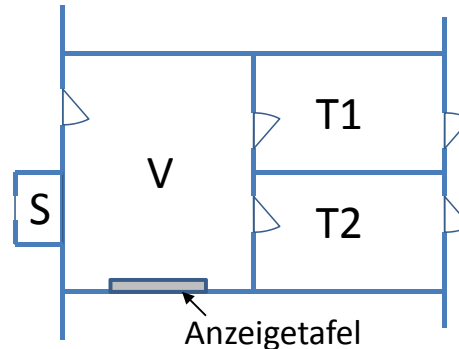
Beschreiben Sie *User-Level Threads* und *Kernel-Level Threads*. Wodurch unterscheiden sich diese beiden Arten der Thread-Implementierung?



Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semaphoren (30)

In einem Fitnesscenter kann man seine Fitness testen lassen. Dazu gibt es ein Testzentrum mit folgenden Einrichtungen (siehe Abbildung):



- einen Schalter (S), bei dem man sich ein Anmeldeformular abholt bzw. das ausgefüllte Formulare abgibt und damit auch eine Identifikationsnummer (ID) für den Test bekommt. Beim Schalter darf sich immer nur maximal eine Person aufhalten.
- einen Vorbereitungsraum (V), in dem sich maximal 6 Personen aufhalten dürfen, um sich auf den Test vorzubereiten.
- zwei Testräume (T1 und T2), in denen sich jeweils maximal eine Person aufhalten darf, um den Fitnesstest zu absolvieren.
- Weiters gibt es eine Anzeigetafel, die die IDs der beiden Testteilnehmer anzeigt, die sich als nächstes für das Eintreten in einen der beiden Testräume bereithalten sollen. Diese Reihenfolge kann von der Reihenfolge der Anmeldung beim Schalter S abweichen. Die Aktualisierung der Anzeigetafel erfolgt mit folgendem Codestück:

```
P(Anzeige)
update_Anzeige()
V(Anzeige)
```

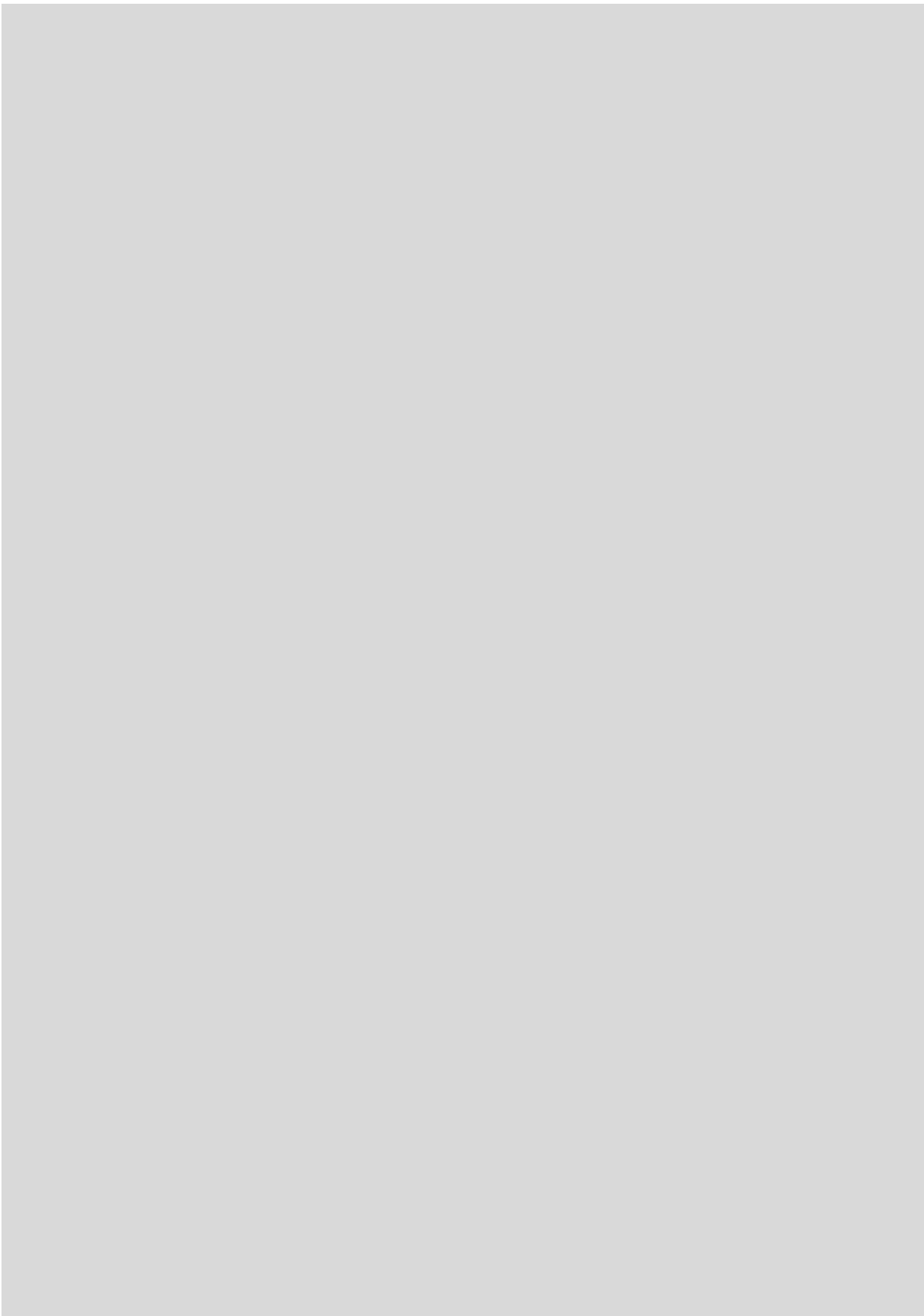
Schreiben Sie einen Prozess für einen Sportler Sp , der unter Einhaltung der obigen Einschränkungen einen Fitnesstest durchführt. Verwenden Sie *Semaphore* zur Synchronisation und vermeiden Sie unnötige Einschränkungen der Parallelität mehrerer Sportler. Sp führt folgende Schritte durch:

- *Sp* holt sich mit der Funktion `get_form()` ein Anmeldeformular vom Schalter, füllt dieses aus (`fill_form()`), gibt das Formular ab und bekommt eine ID (Der Aufruf der Funktion `submit_form()` dient zum Abgeben des Formulars und retourniert die ID).
- *Sp* tritt in den Vorbereitungsraum ein (`enter_V()`) und ruft abwechselnd `prepare()` und `check_board()` auf, die die Vorbereitung auf den Test bzw. das Ablesen der Anzeigetafel realisieren. An `check_board()` übergibt der Prozess die ID von *Sp*. Der Returnwert der Funktion gibt an, ob und für welchen Raum die Nummer von *Sp* auf der Anzeigetafel steht: (Wert 0: ID steht nicht auf der Anzeigetafel, Wert 1: *Sp* soll sich für das Eintreten in T1 bereithalten, Wert 2: *Sp* soll sich für das Eintreten in T2 bereithalten).
Achten Sie darauf, dass Ihre Lösung das gleichzeitige Ablesen der Tafel durch mehrere Sportler erlaubt.
- *Sp* wartet bis der ihm zugewiesene Testraum frei wird, betritt dann den Testraum (Aufruf von `enter_T()` mit T1 oder T2 als Parameter), führt den Test durch (Aufruf von `fitness_test()`), und verlässt den Testraum durch die Ausgangstür, am Plan rechts (Aufruf von `exit_T()`).

Lösen Sie die Synchronisationsaufgabe mit *Semaphoren* und geben Sie Initialisierungen für die Semaphore an. Achten Sie bei der Implementierung von *Sp* darauf, dass die Lösung die Ausführung und Synchronisation einer “beliebigen” Anzahl von Sportlern erlaubt.

(a) Initialisierungen

(b) Code für *Sp*



2 Memory Management (20)

Was ist eine Inverted Page Table?

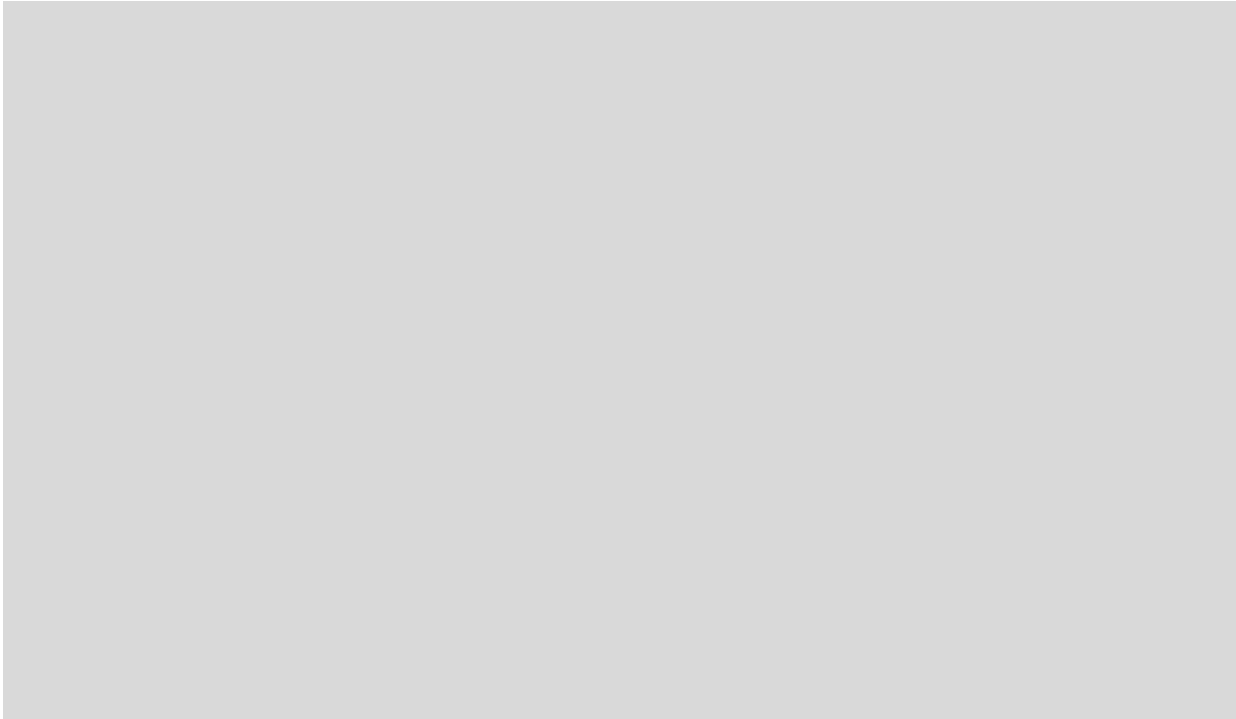
Die Page Table eines kleinen Computersystems ist als Inverted Page Table (IPT) mit acht Einträgen realisiert. Beim Auftreten von Page Faults wird in diesem Computersystem der Clock Algorithmus als globale Page Replacement Strategie eingesetzt.

Im zu lösenden Beispiel ist eine Ausgangsbelegung der Page Table (links), sowie eine Folge von Zugriffen auf den virtuellen Speicher, charakterisiert durch Prozessnummer (PID), adressierte Seite (page) und Offset, gegeben. Simulieren Sie die Ausführung der Zugriffsfolge von links nach rechts und tragen Sie für jeden Zugriff auf den virtuellen Speicher (a) die entsprechende physikalische Adresse, (b) den Zustand der IPT und (c) den Wert des Replacement Pointers (rep. pointer) für den Clock Algorithmus nach dem Speicherzugriff an (Sie brauchen bei jedem Schritt nur die Werte in der IPT anzugeben, die sich beim aktuellen Zugriff geändert haben).

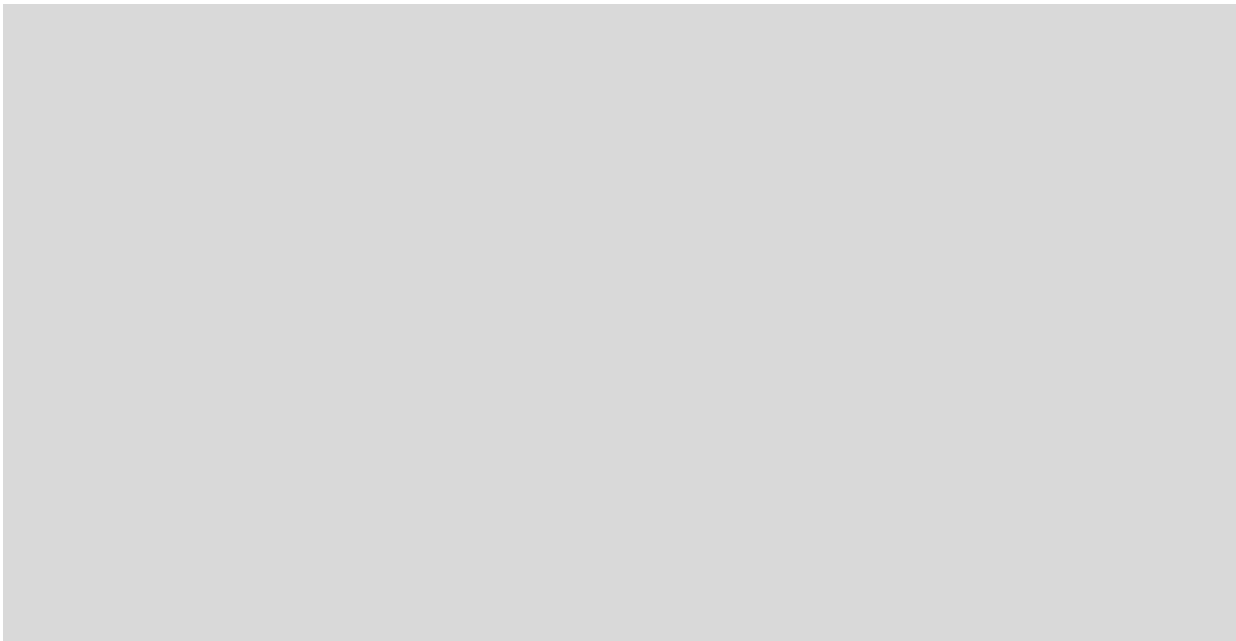
PID page offset				PID page offset				PID page offset				PID page offset				PID page offset			
1 0 00a0				1 0 00a2				1 4 0002				2 3 fffe				2 4 0000			
phys. address				phys. address				phys. address				phys. address				phys. address			
i	PID	page	use	i	PID	page	use	i	PID	page	use	i	PID	page	use	i	PID	page	use
0	1	5	1	0			0	0			0	0			0	0			
1	3	1	0	1	1	0	1	1			1	1			1	1			
2	2	3	0	2			0	2	1	0	1	2			1	2			
3	2	1	0	3			0	3			0	3	1	4	1	3			
4	1	2	1	4			1	4			1	4			0	4			
5	1	3	1	5			0	5			0	5	2	3	1	5			
6	1	4	1	6			0	6			0	6			0	6	2	4	1
7	3	3	1	7			0	7			0	7			0	7			0
rep. pointer 5				rep. pointer 2				rep. pointer 3				rep. pointer 4				rep. pointer 6			

3 Fragen zu Betriebssystemen (50)

Zeichnen Sie das Zustandsdiagramm eines Prozesses, beginnend mit der Entstehung bis zur Beendigung des Prozesses. Geben Sie die Namen der Zustände an und beschreiben Sie kurz jeden Zustand und Zustandsübergang. (5)



Welche Strategien zur Vorbeugung gegen Deadlocks bzw. zur Vermeidung von Deadlocks gibt es? Beschreiben Sie diese kurz. (5)



Bei der Deadlock-Vermeidung spricht man von einem *Safe State* bzw. einem *Unsafe State*. Erklären Sie die Bedeutung dieser Begriffe. (4)

Erklären Sie den Begriff des *Multithreading*. Geben Sie Probleme bei der Verwendung von Threads an. (3)

Nennen Sie die Arten von Optimierungszielen, die ein Scheduler beim Prozess-Scheduling verfolgen kann und geben Sie jeweils zwei Beispiele an. (4)

Was versteht man unter einem *Monitor* zur Prozesssynchronisation? Nennen Sie die wichtigsten Komponenten und Eigenschaften eines Monitors. (4)

Was versteht man unter dem Begriff *Relocation*? (2)

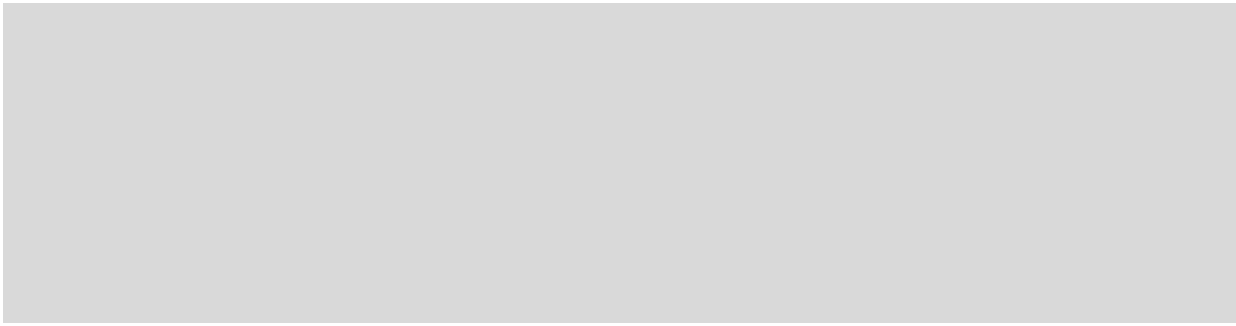
Beim Paging ist ein optimaler Seitenersetzungsalgorithmus bekannt. Beschreiben Sie diesen und geben Sie an, warum er in der Praxis nicht verwendet wird. (3)

Welche Möglichkeiten kennen Sie, mit denen in Paging-Systemen Speicherschutz realisiert werden kann? (4)

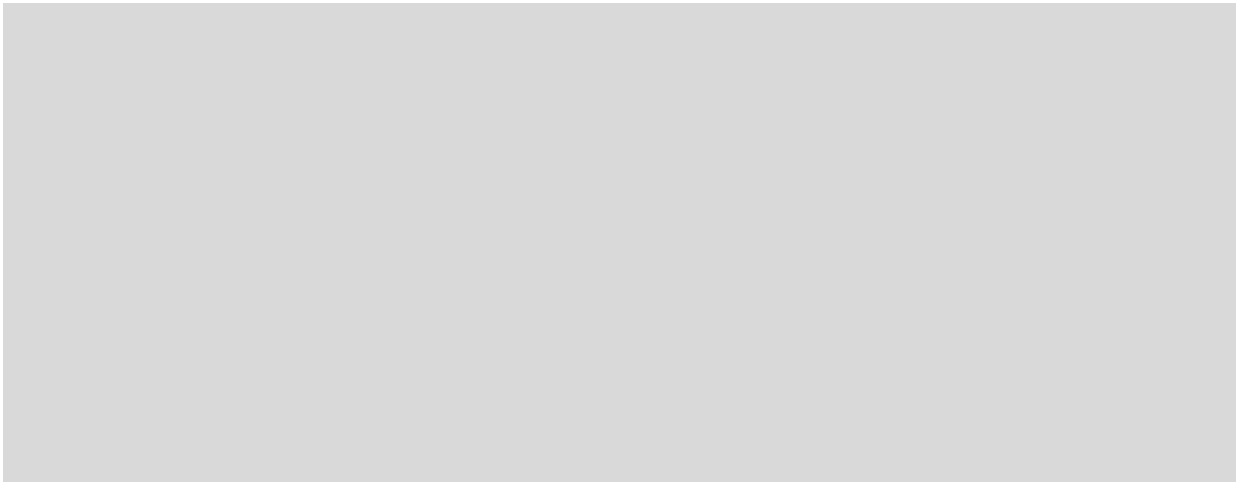
Was versteht man unter der Working-Set Strategie? Beschreiben Sie deren Funktionsweise und wie diese Strategie zur Optimierung eines Paging-Systems eingesetzt werden kann. (5)

Nennen Sie Möglichkeiten, wie die Blockbelegung von Dateien auf einer Festplatte repräsentiert werden kann. Geben Sie Vor- und Nachteile der angegebenen Organisationsformen an. (4)

Erklären Sie die Begriffe *absoluter Pfadname* und *relativer Pfadname* und geben Sie jeweils ein Beispiel an. (4)

A large, empty gray rectangular box intended for the student's answer to the first question.

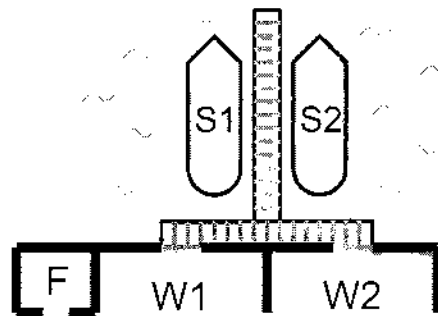
Die Implementierung einer Zugriffsmatrix kann in der Form von *Access Control Lists* oder *Capability Lists* erfolgen. Erklären Sie diese Begriffe. (4)

A large, empty gray rectangular box intended for the student's answer to the second question.

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semaphoren (30)

Sie sollen einen Softwareentwickler beim Schreiben einer Simulationssoftware für eine Schiffsanlegestelle (siehe Abbildung) unterstützen. Diese Software simuliert die Aktionen von Schiffen und Passagieren durch Prozesse, die mit Semaphoren synchronisiert werden.



An der Schiffsanlegestelle gibt es Anlegeplätze für zwei Schiffe, S1 und S2, zwei Wartezonen, W1 und W2, die jeweils einem Schiff zugeordnet sind, und einen Fahrkartenschalter F.

Personen, die mit einem der Schiffe, S1 oder S2, verreisen, lösen am Schalter F eine Fahrkarte für ihr Schiff und gehen dann in die entsprechende Wartezone. Dort warten sie, bis sie an die Reihe kommen und gehen dann über den Steg auf ihr Schiff, um ihre Schiffsreise anzutreten.

Schiffe kommen jeweils leer am Steg an, warten dann, bis die erlaubte Anzahl von Passagieren an Bord ist, und verlassen dann den Steg wieder.

Ergänzen Sie in den Codestücken die fehlenden Semaphoreoperationen, um die Prozesse unter folgenden Rahmenbedingungen zu synchronisieren.

- Ein Schiff kann N Passagiere aufnehmen. Schiffe fahren leer zur Anlegestelle, nehmen dort N Passagiere auf und fahren dann wieder ab.
- Für die Durchgänge von den Warteräumen zum Steg gilt, dass jeder Durchgang von maximal zwei Personen gleichzeitig passiert werden kann.
- Die Einstiegstelle der Schiffe ist sehr schmal. Passagiere können die Schiffe daher nur einzeln besteigen.

- Aus Sicherheitsgründen dürfen sich zu jedem Zeitpunkt maximal K Personen auf dem Steg befinden.
- Beim Anlegen bzw. Ablegen eines Schiffes dürfen sich keine Passagiere am Steg befinden.
- Schiffe und Passagiere sollen unter Einhaltung der angegebenen Regeln möglichst unabhängig und möglichst ohne Einschränkungen der Parallelität agieren können.

(a) Initialisierungen (vor Start der Prozesse)

```

init(SP, 0);      init(S1, 2);      init(P1, 0);      init(S2, 0);
init(S1, 0);      init(ES1, 1);      init(P2, 0);
init(S2, 0);      init(ES2, 1);

```

(b) Prozesse für Schiffe

Bedeutung der verwendeten Funktionen:

anlegen(S) Schiff S legt am Steg an

abfahren(S) Schiff S legt vom Steg ab

/** Schiff S1 **/

```

P(SP); // Schiff für Passagiere
do K times: P(St);

anlegen(S1) // K=2
do N times: V(S1); // Passagiere
for (i=0; i<N; i++) P(Pi); // Passagiere

do K times: P(St);

abfahren(S1)
do K times: V(St);
V(SP);

```

/** Schiff S2 **/

```

P(S2);

anlegen(S2)
do N times: V(S2); // Passagiere
for (i=0; i<N; i++) P(Pi); // Passagiere

abfahren(S2)
do K times: V(St);
V(S2);

```


(c) Prozesse für Passagiere

Bedeutung der verwendeten Funktionen:

fahrkarte_kaufen() der Passagier kauft eine Fahrkarte
warteraum_betreten(W) der Passagier betritt den Warteraum W
steg_betreten() der Passagier betritt und geht auf dem Steg
schiff_betreten(S) der Passagier betritt das Schiff S

/** Passagier fuer S1 **/

/** Passagier fuer S2 **/

fahrkarte_kaufen()

fahrkarte_kaufen()

warteraum_betreten(W1)

warteraum_betreten(W2)

steg_betreten()

steg_betreten()

schiff_betreten(S1)

schiff_betreten(S2)

2 Page Replacement (25)

Gegeben ist ein Arbeitsspeicher mit vier Frames, dessen Seiten mit unterschiedlichen Ersetzungsstrategien ersetzt werden sollen: mit OPT, LRU, bzw. mit dem Clock Algorithmus. Geben Sie in den Tabellen für jeden Algorithmus die Speicherinhalte für jeden Frame nach jedem Zugriff der angegebenen Seitenzugriffsfolge an. Die Seitenzugriffsfolge ist für alle Algorithmen gleich. Sie ist jeweils in der Kopfzeile der Tabelle gegeben. Geben Sie in den Spalten die Speicherbelegung nach dem entsprechenden Seitenzugriff an und kennzeichnen Sie in der letzten, mit *PF* markierten Zeile das Auftreten von Page Faults. Der Arbeitsspeicher ist am Beginn leer.

OPT-Strategie:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A	A	A	A	E	E			E	E	E	E	E	E	E
1		B	B	B	B	F	F	F	F	F	F	G	G	G	G
2			C	C	C	C	C	C	C	C	C	C	H	H	H
3				D	D	D	D	D	D	D	D	D	D	D	D
PF		PF	PF	PF	PF	PF						PF	PF		

LRU-Strategie:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A	A	A	A	E	E			E			E	H	E	E
1		B	B	B	B	F	F	F				F	F	F	F
2			C	C	C	C	C	C	C	C	C	C	C	E	E
3				D	D	D	D	D	D	D	D	C	G	G	G
PF	PF	PF	PF	PF	PF	PF						PF	PF	PF	PF

Clock-Algorithmus:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A		A	A	E	F	F	E	E	E	E	E	E	E	E
1		B	B	B	B	F	F	F	F	F	F	F	F	F	F
2			C	C	C	C	C	C	C	C	C	G	G	G	G
3				D	D	D	D	D	D	D	D	D	H	H	H
PF	PF	PF	PF	PF	PF	PF						PF	PF		PF

Vergleichen und diskutieren Sie die Anzahl der bei den Seitenersetzungsstrategien beobachteten Page Faults.

OPT	6
LRU	12
Clock	12

3 Fragen zu Betriebssystemen (45)

Erklären Sie die drei Begriffe *Deadlock*, *Livelock* und *Starvation*. (5)

Handwritten answer:

Deadlock: Zustand, in dem zwei oder mehr Prozesse auf Ressourcen warten, die von anderen Prozessen gehalten werden, die ebenfalls auf Ressourcen warten. Livelock: Zustand, in dem zwei oder mehr Prozesse auf Ressourcen warten, die von anderen Prozessen gehalten werden, die ebenfalls auf Ressourcen warten. Starvation: Zustand, in dem ein Prozess für eine unendliche Zeit auf Ressourcen warten muss, weil andere Prozesse die Ressourcen monopolisieren.

Worin liegt der grundlegende Unterschied zwischen *Prozessen* und *Threads*? Welcher Vorteil ergibt sich aus der Einführung von Threads für den Benutzer und worauf muss der Benutzer achten? (4)

Handwritten answer:

Prozesse sind unabhängige Ausführungsinstanzen, die ihren eigenen Adressraum und Ressourcen haben. Threads sind Ausführungsinstanzen innerhalb eines Prozesses, die den Adressraum des Prozesses teilen. Threads sind leichter zu erstellen und zu verwalten als Prozesse. Threads ermöglichen eine bessere Auslastung der CPU und eine schnellere Reaktion auf Benutzeranfragen. Der Benutzer muss jedoch auf Synchronisation achten, um Dateninkonsistenzen zu vermeiden.

Was versteht man unter *Virtual Memory Management*? Welche Vorteile bietet es? (5)

Handwritten answer:

Virtual Memory Management ist die Verwaltung von virtuellem Hauptspeicher. Es ermöglicht die Ausführung von Programmen, die mehr Speicher benötigen, als physisch vorhanden ist. Vorteile: Erhöhter Speicherbedarf, bessere Auslastung des Hauptspeichers, Schutz vor Speicherüberschreitung.

Beschreiben Sie Aufgabe und Funktion eines *Translation Lookaside Buffers*? Worauf hat man bei der Betriebssystemimplementierung bei einem Process Switch zu achten, wenn man einen Translation Lookaside Buffer verwendet? (4)

Ein TLB (Translation Lookaside Buffer) ist eine kleine, schnelle Cache-Struktur, die die aktuellen Übersetzungen von virtuellen zu physischen Adressen speichert. Seine Aufgabe ist es, die Performance des Systems zu verbessern, indem es die Zeit, die für die Suche nach der richtigen physischen Adresse benötigt wird, verkürzt. Bei einem Prozesswechsel (Process Switch) muss der TLB geleert werden, da die gespeicherten Übersetzungen für den alten Prozess gültig sind, aber für den neuen Prozess nicht. Wenn der TLB nicht geleert wird, kann es zu falschen Übersetzungen und damit zu Datenkorruption kommen. Daher ist es wichtig, dass der TLB bei einem Prozesswechsel korrekt geleert wird.

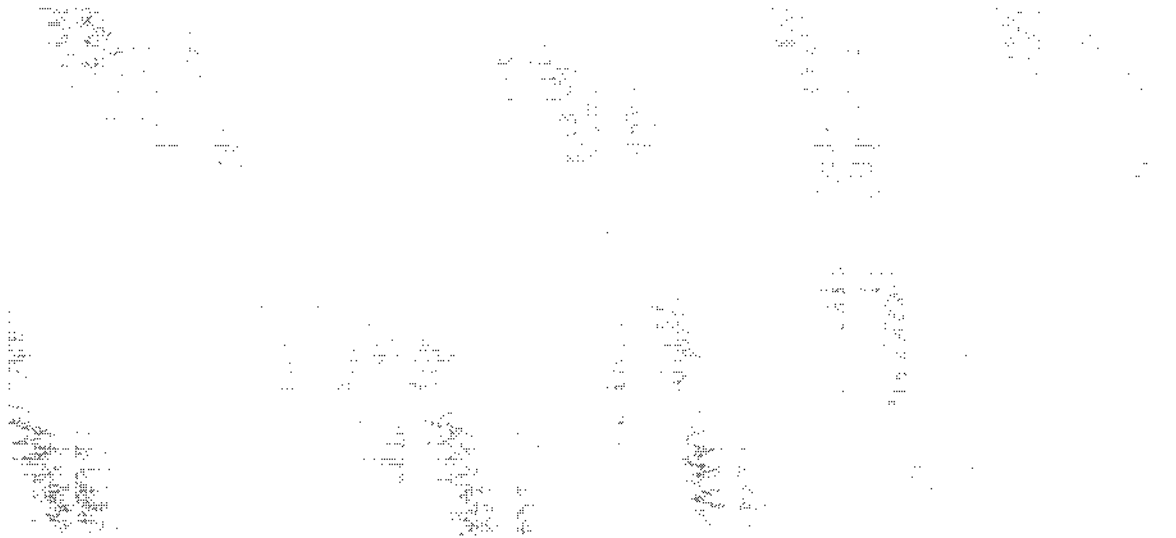
Was versteht man unter *Thrashing*? (2)

Thrashing ist ein Zustand, in dem ein System so viel Zeit mit dem Verschieben von Daten zwischen dem Hauptspeicher und der Festplatte verbringt, dass keine weiteren Berechnungen durchgeführt werden können. Dies tritt auf, wenn die Arbeitslast des Systems größer ist als die Kapazität des Hauptspeichers. In diesem Zustand wird die Leistung des Systems stark beeinträchtigt, da die CPU fast ausschließlich mit dem Verschieben von Daten beschäftigt ist und keine Zeit für die Ausführung von Berechnungen hat.

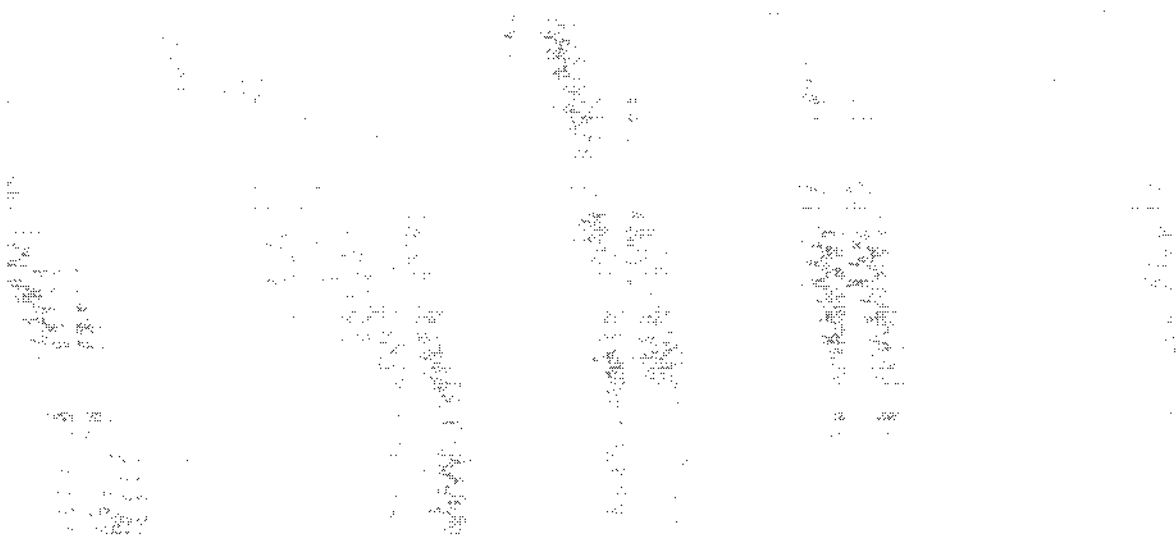
Wie funktioniert *Round Robin* Scheduling? Welchen wichtigen Parameter gibt es bei diesem Verfahren? Wie wird man diesen Parameter günstiger Weise wählen? (4)

Round Robin Scheduling ist ein Zeit-slices-basiertes Scheduling-Verfahren. Es teilt die CPU-Zeit in kleine, gleich große Zeit-slices auf, die nacheinander den verschiedenen Prozessen zugeteilt werden. Der wichtigste Parameter bei diesem Verfahren ist die Zeit-slice-Dauer. Eine zu kurze Zeit-slice-Dauer führt zu einem hohen Overhead, da die CPU häufig zwischen den Prozessen wechseln muss. Eine zu lange Zeit-slice-Dauer führt zu einer schlechten Reaktionszeit, da die Prozesse zu lange warten müssen, bevor sie die CPU erhalten. Die Zeit-slice-Dauer sollte daher so gewählt werden, dass sie einen guten Kompromiss zwischen Overhead und Reaktionszeit darstellt.

Was versteht man unter *Buffering*? Welche Vorteile bietet es, wo liegen seine Grenzen und worauf hat man bei der Verwendung von Puffern bei der Betriebssystemimplementierung zu achten? (5)



Wie ist der Inhalt einer (mechanischen) Festplatte typischer Weise organisiert? (4)



Wie ist ein *i-node* aufgebaut? Welche Informationen enthält er? (4)



Was beschreibt das Modell von Bell und LaPadula? Geben Sie die vom Modell geforderten Eigenschaften an. (4)

Nennen Sie Design Prinzipien für die Konstruktion von sicheren Systemen. Geben Sie für jede Regel ein Beispiel an. (4)