## Aufgabe 1: Einfachauswahlaufgabe (12 Pkt.)

1) Welche logische Hierarchieebene liegt nicht unterhalb des Betriebssystems?

| (i) Physikalische Geräte | (ii) Mikroprogrammierung | (iii) Anwendungsprogramme | (iv) Maschinensprache |
|---|---|---|---|
| X | | | |

2) Welche Art von Unterprogrammen gibt es?

| (i) halbgeschlossene | (ii) geschlossene | (iii) halboffene | (iv) unendliche |
|---|---|---|---|
| | X | | |

3) Wie bezeichnet man die Schnittstelle zwischen Anwendungsprogrammen und Betriebssystem, durch welche diese z.B. auf Ressource zugreifen können, auf welche sie keinen direkten Zugriff haben?

| (i) Systemaufrufe | (ii) Nutzeraufrufe | (iii) Nachrufe | (iv) Programmaufrufe |
|---|---|---|---|
| X | | | |

4) Wie ist die mittlere Antwortzeit bei Prozessen mit folgender Ressourcennutzung unter Anwendung von Multiprogramming?

| Job | durchschnittliche CPU-Auslastung | Dauer | benötigter Speicher | Platte | Terminal | Drucker |
|---|---|---|---|---|---|---|
| 1 | 20% | 10 min. | 50 KBytes | - | - | - |
| 2 | 45% | 20 min. | 100 KBytes | - | ja | - |
| 3 | 25% | 30 min. | 80 KBytes | ja | - | ja |

| (i) 3,33 min. | (ii) 6,66 min. | (iii) 10 min. | (iv) 20 min. |
|---|---|---|---|
| | | | X |

5) Was ist kein erlaubter direkter Zustandsübergang im 5-Zustands-Prozessmodell?

| (i) new → exit | (ii) ready → running | (iii) running → ready | (iv) blocked → ready |
|---|---|---|---|
| X | | | |

6) Welcher Zustand kommt beim 7-Zustands-Prozessmodell verglichen zum 5-Zustands-Prozessmodell hinzu?

| (i) running | (ii) new, suspend | (iii) ready, suspend | (iv) exit |
|---|---|---|---|
| | | X | |

7) Welche Aussage bezüglich Threads ist korrekt?

(i) Zur Generierung eines neuen Threads in einem existierenden Prozess ist wesentlich mehr Zeit notwendig, als zur Generierung eines neuen Prozesses.
(ii) Kontextwechsel unter Threads innerhalb eines Prozesses erfordern mehr Zeit als Kontextwechsel von Prozessen.
(iii) Durch den getrennten Adressraum der Threads eines Prozesses sind die Daten einzelner Threads bezüglich anderer Threads sicher.
(iv) Im Falle von Kernel-Level-Threads (KLT) kann das Betriebssystem mehrere Threads eines Prozesses auf verschiedenen Prozessoren einer Multiprozessorumgebung ausführen.

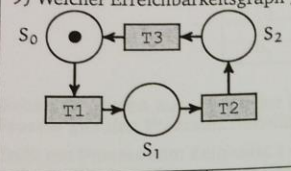
| (i) | (ii) | (iii) | (iv) X |
|---|---|---|---|

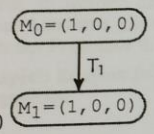
8) Ein Computer habe vier Bandlaufwerke und n Prozesse, von denen jeder zwei Bandlaufwerke gleichzeitig für seine Ausführung benötigt. Bei einer Anfrage bekommt ein Prozess ein beliebiges freies Bandlaufwerk zugewiesen. Für einen Prozess ist es irrelevant welche Bandlaufwerke er verwendet, solange es zwei sind. Nachdem er die zwei Bandlaufwerke erhalten hat, terminiert er nach endlicher Zeit. Für welchen Wert von n besteht die Möglichkeit eines Deadlocks?

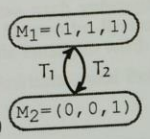
| (i) 4 | (ii) 3 | (iii) 2 | (iv) 1 |
|---|---|---|---|
| X | | | |

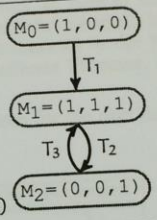
1

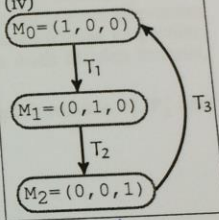
9) Welcher Erreichbarkeitsgraph gehört zu folgendem Petrinetz?




(i)



(ii)



(iii)



(iv)



X

1

10) Für eine korrekte Lösung des wechselseitigen Ausschlusses müssen drei Bedingungen erfüllt sein. Was ist keine davon?

| (i) Mutual Exclusion | (ii) Correlated Blocking | (iii) Progress | (iv) Bounded Waiting |
|---|---|---|---|
| | X | | |

1

11) Wie bezeichnet man die Einheiten fester Länge, in die der logische Adressraum (virtuelle Speicher) unterteilt wird?

| (i) Seitenrahmen | (ii) Bücher | (iii) Seiten | (iv) Bilder |
|---|---|---|---|
| | | X | |

1

12) Der verfügbare Arbeitspeicher eines Systems umfasst 64 MByte und soll komplett als physischer Speicher für das Paging-System verwendet werden. Ein Seitenrahmen habe eine Größe von 2 KByte. Die kleinste adressierbare Einheit (Wort) sei 1 Byte. Wie viele Bits benötigen Sie zur Adressierung eines Wortes innerhalb einer Seite?

**Achtung:** Gehen Sie von folgender Konversion aus:

1024 Byte = $2^{10}$ Byte = 1 KByte

1024 KByte = $2^{10}$ KByte = 1 MByte

| (i) 9 | (ii) 10 | (iii) 11 | (iv) 20 |
|---|---|---|---|
| | | X | |

1

## Aufgabe 2: Scheduling (19 Pkt.)

In dieser Aufgabe sollen drei Scheduling-Strategien untersucht werden: die nicht-präemptive Strategie **FCFS (First Come First Served)**, die präemptive Strategie **SRPT (Shortest Remaining Processing Time)** und die präemptive Strategie **RR (Round Robin)**. Dazu seien die folgenden Prozesse mit ihren Ankunftszeitpunkten und Rechenzeiten (in beliebigen Zeiteinheiten) gegeben.

| Prozess | Ankunftszeitpunkt | Rechenzeit |
|---|---|---|
| $P_1$ | 0 | 6 |
| $P_2$ | 3 | 2 |
| $P_3$ | 2 | 5 |
| $P_4$ | 3 | 3 |
| $P_5$ | 5 | 2 |

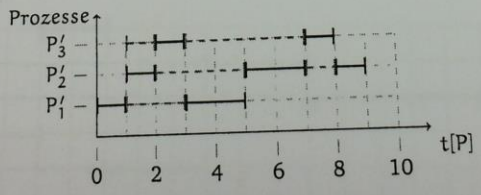
Gehen Sie davon aus, dass jeder Prozess sein Zeitquantum stets vollständig ausnutzt, d.h. kein Prozess gibt den Prozessor freiwillig frei (Ausnahme: bei Prozessende).

Trifft ein Prozess zum Zeitpunkt t ein, so wird er direkt zum Zeitpunkt t beim Scheduling berücksichtigt. Wird ein Prozess zum Zeitpunkt t′ unterbrochen, so reiht er sich auch zum Zeitpunkt t′ wieder in die Warteschlange ein. Sind zwei Prozesse absolut identisch bezüglich ihrer relevanten Werte, so werden die Prozesse nach aufsteigender Prozess–ID in die Warteschlange eingereiht. Diese Annahme gilt sowohl für neu im System eintreffende Prozesse, als auch für den Prozess, dem der Prozessor u.U. gerade entzogen wird!

**Beispiel:** Es seien folgende Ankunfts– und Rechenzeiten für die drei Beispielprozesse $P_1'$, $P_2'$ und $P_3'$ gegeben:

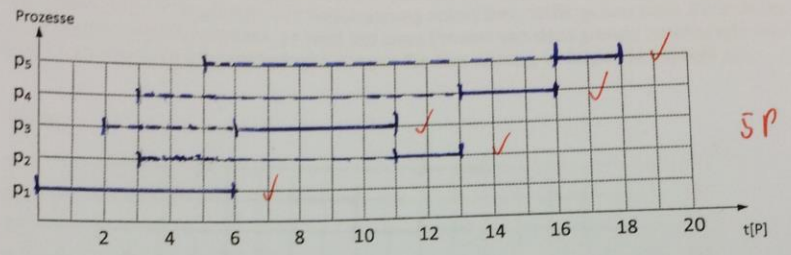
| Prozess | Ankunftszeitpunkt | Rechenzeit |
|---|---|---|
| $P_1'$ | 0 | 3 |
| $P_2'$ | 1 | 4 |
| $P_3'$ | 1 | 2 |

Das folgende Diagramm zeigt ein zufälliges Scheduling der drei Prozesse $P_1'$, $P_2'$ und $P_3'$ und soll die Art der Darstellung veranschaulichen:




Bearbeiten Sie unter den gegebenen Voraussetzungen nun die folgenden Aufgaben:

a. Erstellen Sie entsprechend des Beispiels ein Diagramm für die **nicht-präemptive Strategie FCFS**, das für die Prozesse $P_1$–$P_5$ angibt, wann welchem Prozess Rechenzeit zugeteilt wird und wann die Prozesse jeweils terminieren. Kennzeichnen Sie zudem für jeden Prozess seine Ankunftszeit. Tragen Sie Ihre Lösung in folgende Vorlage ein:



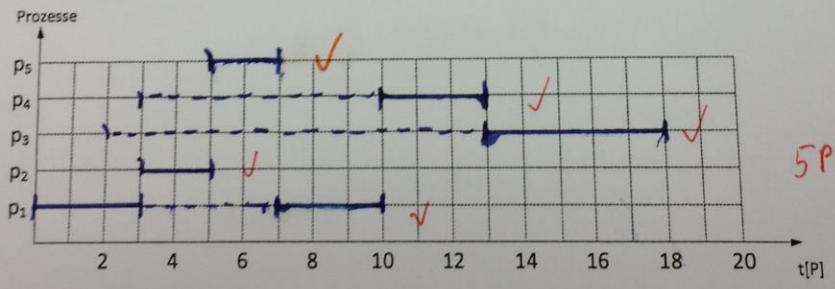

5P

b. Berechnen Sie als Dezimalzahl mit einer Nachkommastelle die mittlere Verweil- und Wartezeit für das Verfahren FCFS und tragen Sie Ihre Ergebnisse in die folgende Tabelle ein.

| Strategie | Mittlere Verweilzeit | Mittlere Wartezeit |
|---|---|---|
| FCFS | 10,2 ✓✓ | 6,6 ✓✓ |

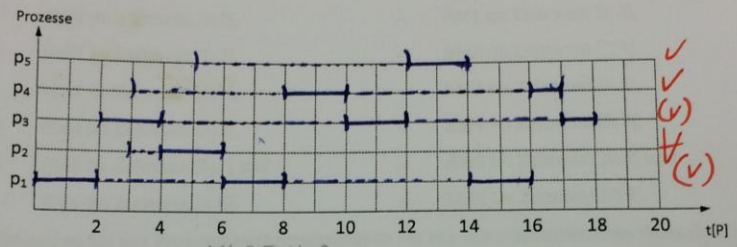
4P

c. Erstellen Sie entsprechend des Beispiels ein Diagramm für die **präemptive Strategie SRPT**, das für die Prozesse $P_1$–$P_5$ angibt, wann welchem Prozess Rechenzeit zugeteilt wird und wann die Prozesse jeweils terminieren. Kennzeichnen Sie zudem für jeden Prozess seine Ankunftszeit. Tragen Sie Ihre Lösung in folgende Vorlage ein:




5P

d. Erstellen Sie entsprechend des Beispiels ein Diagramm für die **präemptive Strategie RR**, das für die Prozesse $P_1$–$P_5$ angibt, wann welchem Prozess Rechenzeit zugeteilt wird und wann die Prozesse jeweils terminieren. Die Dauer einer Zeitscheibe betrage 2 Zeiteinheiten. Gehen Sie davon aus, dass jeder Prozess die Dauer seiner Zeitscheibe stets vollständig ausnutzt, sofern er nicht terminiert. Terminiert ein Prozess aber vor Ablauf seiner Zeitscheibe, gibt er den Prozessor zum Zeitpunkt der Terminierung sofort frei. Trifft genau nach Ende einer Zeitscheibe ein neuer Prozess ein, so wird der neue Prozess **vor** dem gerade unterbrochenen Prozess in die Warteschlange eingereiht. Tragen Sie Ihre Lösung in folgende Vorlage ein:




✓ ✓ (✓) ✓ (✓)

~~18~~ 8 14 3

4 P.

18/19

## Aufgabe 3: Deadlocks (16 Pkt.)

Bearbeiten Sie die folgenden Aufgaben.

a. Gegeben seien zwei Prozesse P und Q, die auf einem Uniprozessorsystem ausgeführt werden sollen. Der Prozess Q benötigt 9 und der Prozess P 10 Zeiteinheiten für seine Ausführung. Es stehen die Betriebsmittel BM 1–6 zur Verfügung, die von den Prozessen während ihrer Ausführung benötigt werden.

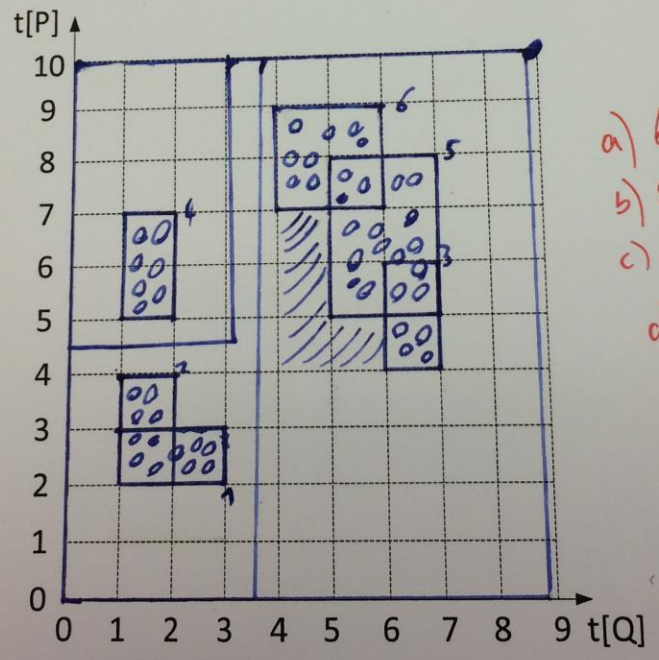
Q benötigt:

- BM1 im Zeitraum ]1;3[,
- BM2 im Zeitraum ]1;2[,
- BM3 im Zeitraum ]6;7[,
- BM4 im Zeitraum ]1;2[,
- BM5 im Zeitraum ]5;7[ und
- BM6 im Zeitraum ]4;6[.

P benötigt:

- BM1 im Zeitraum ]2;3[,
- BM2 im Zeitraum ]2;4[,
- BM3 im Zeitraum ]4;6[,
- BM4 im Zeitraum ]5;7[,
- BM5 im Zeitraum ]5;8[ und
- BM6 im Zeitraum ]7;9[.

Skizzieren Sie das Prozessfortschrittsdiagramm für die oben beschriebenen Anforderungen, indem Sie die benötigten Betriebsmittel entsprechend ihrer zeitlichen Verwendung durch die beiden Prozesse P und Q korrekt einzeichnen. Gehen Sie davon aus, dass der Scheduler die Prozesse P und Q zu beliebigen Zeitpunkten aktivieren bzw. suspendieren kann. Gehen Sie zudem davon aus, dass ein Kontextwechsel zwischen P und Q keinerlei Zeit in Anspruch nimmt. Tragen Sie Ihre Lösung in die folgende Vorlage ein:




a) 6/6
b) 2/2
c) 4/5
d) 0/3

///: unsicher/kritisch
000: unmöglich

b. Kennzeichnen Sie **deutlich** alle unmöglichen und unsicheren Bereiche im Diagramm aus Teilaufgabe a).

c. Zeichnen Sie alle *prinzipiell* unterschiedlichen Ausführungspfade in das Diagramm aus Teilaufgabe a) ein, so dass die Prozesse P und Q terminieren.

d. Bezogen auf Teilaufgabe a): Wieviele *prinzipiell* unterschiedliche Ausführungspfade gibt es, die in einem Deadlock enden?
Geben Sie für jeden solchen Ablauf ein Beispiel an und beschreiben sie dabei, wann und wie lange Prozess P bzw. Q aktiviert bzw. suspendiert werden muss, um in eine Deadlock-Situation zu gelangen.

Es gibt 7 prinzipiell unterschiedliche Ausführungspfade.

Ein Beispiel in der Zeichnung auf der Seite davor wäre ein Eintreffen des Ausführungspfades bei 3 und Q1. In diesem Fall wäre P für 3 und Q für 1 Zeiteinheit aktiv, um in eine Deadlock-Situation zu gelangen.

## Aufgabe 4: Petrinetze

10 (17 Pkt.)

Bearbeiten Sie folgende Aufgaben:

a. Gegeben sei ein Prozess, der wiederholt zwei Kernel Level Threads (KLT) startet und wartet, bis beide KLTs beendet wurden. Folgender Pseudocode verdeutlicht den Ablauf des Prozesses:

```
1  while(true) {
2          <Berechnung vorbereiten>;
3          ...
4          <Starte Threads>; // Reihenfolge beliebig
5          ...
6          <Warte auf Ende beider Threads>;
7  }
```

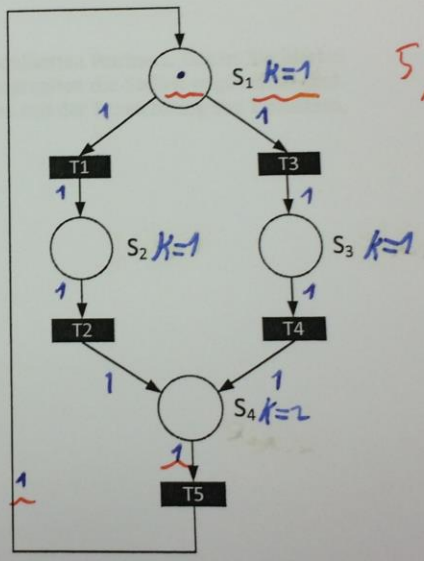
Der Ablauf dieses Prozesses soll im Folgenden als Petrinetz modelliert werden. Dazu ist folgender Rahmen vorgegeben:

Für die Semantik der Stellen:

- $S_1$: Berechnung vorbereitet
- $S_2$: Thread 1 arbeitet
- $S_3$: Thread 2 arbeitet
- $S_4$: Threads beendet

Für die Semantik der Transitionen gilt:

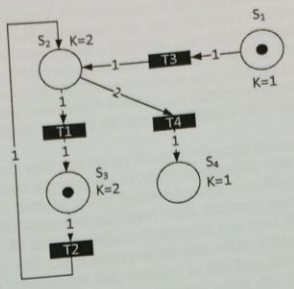
- T1: Starte Thread 1
- T2: Beende Thread 2
- T3: Starte Thread 2
- T4: Beende Thread 2
- T5: Berechnung vorbereiten




5,5

Ergänzen Sie in dem oben dargestelltem Rahmen die Kapazität von Stellen, das Kantengewicht für Kanten und die Anfangsmarkierung, so dass der beschriebene Prozess durch das Petrinetz modelliert wird. Zu Beginn soll die Berechnung bereits vorbereitet sein.

b. Gegeben sei folgendes Petrinetz:




Erstellen Sie den Erreichbarkeitsgraphen zu dem modellierten Petrinetz. Geben Sie hierbei auch an, wie in den Markierungen des Erreichbarkeitsgraphen die Stellen angeordnet sind. Beschriften Sie alle Übergänge zwischen Markierungen mit der Bezeichnung der Transition, die hierfür feuern muss.

$M_0 := (S1, S2, S3, S4)$ ✓

$M_0 (1,0,1,0)$ ✓ $\xrightarrow{T_2}$ $M_2 (1,1,0,0)$ –

$\downarrow T_3$ –

$M_1 (0,1,1,0)$ $\xrightarrow{T_4}$ $M_3 (0,0,0,2)$

3

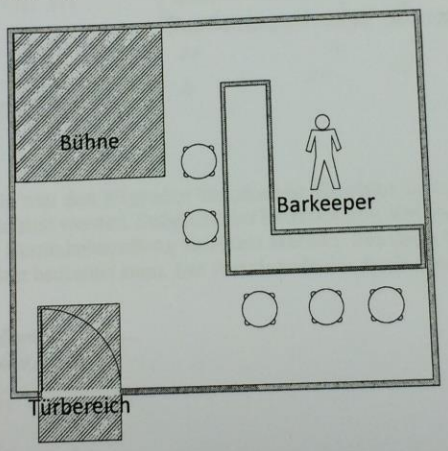
c. Gibt es eine Markierung des Erreichbarkeitsgraphen aus Teilaufgabe b, in der eine teilweise Verklemmung (partieller Deadlock), aber keine echte Verklemmung vorliegt? Wenn ja, benennen Sie diese Markierung. Begründen Sie (für ja oder nein) in jedem Fall Ihre Antwort.

1,5 ✓

Ja, für $S_1$ und $S_4$. Diese beiden Stellen sind vom Kreislauf abgeschnitten, da keine Transitionen ein- bzw. ausgehen. S1 kann von nirgendwo erreicht werden, und von S4 kann keine weitere Stelle mehr erreicht werden → kein Kreislauf

## Aufgabe 5: Semaphore (16 Pkt.)

In dieser Aufgabe sollen Sie das Konzept der Semaphore am Beispiel einer kleinen Studentenbar umsetzen. Dazu soll das Betreten und Verlassen der Bar simuliert werden, welche 5 Barhocker bereitstellt, die von den Gästen genutzt werden können. Die Gäste (welche hier als Prozesse angesehen werden können) betreten bzw. verlassen die Bar über eine Tür. Damit es nicht zu Auseinandersetzungen kommt, darf sich zu jedem Zeitpunkt nur eine Person im Türbereich aufhalten. Auch dieser Umstand soll von Ihnen modelliert werden. Es dürfen sich stets nur maximal so viele Personen in der Bar befinden (inklusive einer Person im Türbereich), wie Barhocker vorhanden sind.
Nachdem ein Gast die Bar betreten und auf einen Hocker Platz genommen hat, versucht er drei Getränke zu sich zu nehmen. Dazu signalisiert er dem Barkeeper, der sich bereits hinter dem Tresen befindet, für jedes Getränk einzeln einen Bestellwunsch. Der Gast wartet, bis sein Getränk zubereitet wurde, was ihm vom Barkeeper signalisiert wird. Solange keine Bestellung aufgegeben wurde, ist der Barkeeper untätig. Er kann zu jedem Zeitpunkt nur eine Bestellung bedienen. Nach der Bearbeitung einer Bestellung soll der Barkeeper wieder allen Gästen zur Verfügung stehen. Nachdem ein Gast drei Getränke konsumiert hat, verlässt er die Bar durch den Türbereich. Da alle Gäste den Barkeeper gut kennen und implizit anschreiben, kann der Bezahlvorgang außer Acht gelassen werden.
Die nachfolgende Abbildung zeigt den schematischen Aufbau.




Beantworten Sie nun auf Grundlage dieses Szenarios folgende Aufgaben:

a. Was sind die kritischen Bereiche bei diesem Problem?

2/2

Türbereich ✓ und die Anzahl der Barhocker. (-1)

b. Tragen Sie in folgende Tabelle die aus Ihrer Sicht benötigten Semaphoren ein, um den beschrieben Sachverhalt zu synchronisieren. Geben Sie zu jedem von Ihnen angedachten Semaphor einen Bezeichner, den Typ und eine kurze Beschreibung, wofür er verwendet werden soll, an. Die beiden Semaphoren `bestellwunsch` und `getränk_fertig` sind bereits gegeben. 3/3

| Bezeichner | Typ | Zweck |
|---|---|---|
| `bestellwunsch` | Zählsemaphor | Signalisiert dem Barkeeper einen Bestellwunsch |
| `getränk_fertig` | Zählsemaphor | Signalisiert einem Gast die Fertigstellung seines Getränks |
| tür ✓ | Binärsem. | Gibt an, ob Tür gerade benutzt ist/ob einer rein oder raus geht |
| barhocker ✓ | Zählsem. | Gibt an, wie viele Hocker besetzt sind. |
| barkeeper ✓ | Binärsem. | Gibt an, ob Barkeeper gerade tätig ist oder nicht. |

c. Geben Sie in Pseudocode an, wie die benötigten Semaphore initialisiert werden müssen. Verwenden Sie dabei die Notation, die bei den gegebenen Semaphoren verwendet wurde: 1/3

| Pseudocode | Bedeutung |
|---|---|
| `init(bestellwunsch, 0);` | Initialisiert den Semaphor `bestellwunsch` mit dem Wert 0. |
| `init(getränk_fertig, 0);` | Initialisiert den Semaphor `getränk_fertig` mit dem Wert 0. |
| init(tür, 0); | Initialisiert den Semaphor tür mit dem Wert 0. |
| init(barhocker, 5); ✓ | " " " barhocker mit dem Wert 5. |
| init(barkeeper, 0); | " " " barkeeper mit 0. |

d. Vervollständigen Sie nun den folgenden Pseudocode für einen Gast, so dass auch mehrere Gäste stets synchronisiert werden. Dabei soll das Betreten bzw. Verlassen des Lokals durch die Gäste sowie deren Getränkebestellung simuliert werden. Beachten Sie, dass der Barkeeper immer nur einen Gast bedienen kann. Der Pseudocode des Barkeepers ist bereits gegeben. 5,5/8

```
1  Gast() {
2      while(true) {
3      wait(tür);
4
5          <die Bar betreten>;
6
7      signal(tür);
8          <auf Hocker Platz nehmen>;
9      wait(barhocker);
10
11
12         for(Getränke = 0; Getränke < 3; Getränke++) {
13     signal(bestellwunsch);
14
15
16             //Bestellung aufgeben
17     wait(bestellwunsch);
18     wait(barkeeper);
19
20             //Zubereitung abwarten
21
22
23
```

```
24
25                signal(getränk_fertig);
26
27          <Getränk entgegennehmen>;
28          signal(barkeeper);
29
30
31        }
32        wait(tür);
33        signal(barbrocken);
34          <die Bar verlassen>;
35        signal(tür);
36
37
38
39  }
```

```
1  Barkeeper() {
2        while(true) {
3              //auf Bestellung warten
4              wait(bestellwunsch);
5              //Fertigstellung signalisieren
6              signal(getränk_fertig);
7              <Getränk an den Gast geben>;
8        }
9  }
```

Verwenden Sie für den Zugriff auf Ihre Semaphore folgende Notation:

| Pseudocode | Beispiel | Bedeutung |
|---|---|---|
| `wait(<semaphor>);` | `wait(mutex);` | Erniedrigt den Wert des Semaphor `mutex` um eins |
| `signal(<semaphor>);` | `signal(mutex);` | Erhöhe den Wert des Semaphor `mutex` um eins |
| `<Aktion>` | `<Getränk entgegennehmen>` | Führe die gelistete Aktion aus |
| `//Kommentar` | `//Zubereitung abwarten` | Kommentarzeile |

## Aufgabe 6: Seitenersetzung (20 Pkt.)

Die Menge der Seiten sei gegeben durch $N = \{1,2,3,4,5\}$ und die Menge der Seitenrahmen, die für die Speicherung der Seiten im Arbeitsspeicher zur Verfügung steht, sei gegeben durch $Frames = \{FR_1, FR_2, FR_3\}$. Auf die fünf Seiten der Menge N werde in folgender Reihenfolge zugegriffen (Reference String):

w = 1 3 2 2 5 3 4 5 4 1 2 1

Ein Seitenfehler liegt immer dann vor, wenn sich eine referenzierte Seite nicht im Arbeitsspeicher befindet. Dieser ist zu Beginn leer.

a. Dokumentieren Sie den Vorgang der Seitenersetzung nach der Seitenersetzungsstrategie **LRU (Least Recently Used)**, indem Sie alle Veränderungen im Speicher in der folgenden Tabelle dokumentieren. Markieren Sie dabei alle zu ersetzenden Seiten mit einem Stern. Befüllen Sie die zu Beginn leeren Seitenrahmen initial aufsteigend nach ihrem Index.

| Reference String | 1 | 3 | 2 | 2 | 5 | 3 | 4 | 5 | 4 | 1 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $FR_3$ | | | 2 | 2 | 2 | 2* | 2 | 2 | 2 | 1 | 1 | 1 |
| $FR_2$ | | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 |
| $FR_1$ | 1 | 1 | 1* | 1 | 5 | 5 | 5 | 5 | 5 | 5 | 2 | 2 |

b. Ermitteln Sie mit Hilfe der vorangegangen Tabelle die Anzahl der Seitenfehler. Unterscheiden Sie dabei zwischen Seitenfehler durch Erstbelegung und Seitenfehler nach der Erstbelegung. Berechnen Sie zudem die Gesamtzahl der Seitenfehler und tragen Sie die Werte in die folgende Tabelle ein:

| | |
|---|---|
| Seitenfehler durch Erstbelegung: | 3 |
| Seitenfehler nach der Erstbelegung: | 4 |
| Gesamtanzahl der Seitenfehler: | 7 |

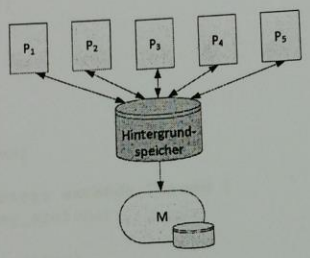
c. Kann die LRU-Strategie bessere Ergebnisse bezüglich der aufgetretenen Seitenfehler liefern als die OPT-Strategie? Begründen Sie Ihre Antwort.

d. Nennen Sie zwei weitere Seitenersetzungsstrategien (additiv zur LRU-Strategie und zur OPT-Strategie).

LFU (Least Frequently Used)
Clock / Second Chance
Climb
FIFO

## Aufgabe 7: Threads in Java (20 Pkt.)

In dieser Aufgabe soll der Betrieb eines Terminals in Java simuliert werden, auf dem insgesamt 5 Prozesse aktiv sind. Es wird angenommen, dass die Prozesse $P_1$ bis $P_5$ spezielle Rechenprozesse sind, die regelmäßig große Datenmengen in speziellen *Zugriffsphasen* auf den gemeinsamen Hintergrundspeicher schreiben. Aus Performanzgründen dürfen zu einem Zeitpunkt jedoch **maximal 2 Rechenprozesse gleichzeitig** den Hintergrundspeicher beschreiben. Ein Rechenprozess muss deshalb gegebenenfalls mit dem Beginn des Schreibzugriffs warten, damit diese Bedingung nicht verletzt wird. Vor der Wartephase meldet er einen Schreibewunsch im System an. In den *Zwischenphasen* greifen die Rechenprozesse nicht auf den Hintergrundspeicher zu und führen spezielle Berechnungen durch.
In regelmäßigen Abständen soll der aktuelle Stand des Hintergrundspeichers über ein Bandlaufwerk gesichert werden, wobei exakt der Zustand zu einem dedizierten Zeitpunkt gespeichert werden soll. Diese Datensicherung führt ein weiterer Datensicherungsprozess M durch. Der Datensicherungsprozess befindet sich im Ruhezustand, wenn aktuell keine Sicherung nötig ist. Er geht in den Wartezustand über, wenn eine Sicherung nötig ist und wartet solange, bis kein Rechenprozess auf den Hintergrundspreicher zugreift und kein Rechenprozess einen Schreibewunsch besitzt.
Das beschriebene Szenario ist in folgender Abbildung nochmals dargestellt:




Im folgenden soll eine Klasse `Speicher` zur Simulation des Hintergrundspeichers implementiert werden. Die Beispielimplementierungen der Klassen `Rechenprozess`, `Datensicherungsprozess` und `Terminal` soll Ihnen verdeutlichen, wie die Klasse `Speicher` verwendet werden kann: *(Bitte wenden!)*

Die Klasse `Terminal`:

```
1  public class Terminal {
2      private Speicher my_speicher;
3      private Rechenprozess[] rechenprozesse;
4      private Datensicherungsprozess my_datensicherungsprozess;
5
6      public Terminal() {
7          my_speicher = new Speicher(2);
8          rechenprozesse = new Rechenprozess[5];
9          for(int i = 0; i < rechenprozesse.length; i++) {
10             rechenprozesse[i] = new Rechenprozess(my_speicher, i);
11             rechenprozesse[i].start();
12         }
13         my_datensicherungsprozess = new Datensicherungsprozess(my_speicher);
14         my_datensicherungsprozess.start();
15     }
16
17
18     public static void main(String[] args) {
19         new Terminal();
20     }
21 }
```

Die Klasse `Rechenprozess`:

```
1  import java.util.Random;
2
3  public class Rechenprozess extends Thread {
4      private Speicher my_speicher;
5      private int id;
6      private Random generator;
7
8      public Rechenprozess(Speicher speicher, int id) {
9          this.my_speicher = speicher;
10         this.id = id;
11         generator = new Random();
12     }
13
14     public void run() {
15         try {
16             Thread.sleep(generator.nextInt(2000) + 500);
17             my_speicher.schreibzugriffBeginnen(id);
18             Thread.sleep(generator.nextInt(2000) + 500);
19             my_speicher.schreibzugriffBeenden(id);
20         } catch (InterruptedException ie) {
21         }
22     }
23 }
```

Die Klasse `Datensicherungsprozess`:

```
1  import java.util.Random;
2
3  public class Datensicherungsprozess extends Thread {
4      private Speicher my_speicher;
```

```
5       private Random generator;
6
7       public Datensicherungsprozess(Speicher speicher) {
8           this.my_speicher = speicher;
9           generator = new Random();
10      }
11
12      public void run() {
13          while (true) {
14              try {
15                  Thread.sleep(generator.nextInt(2000) + 500);
16                  my_speicher.sicherungBeginnen();
17                  System.out.println("Daten werden gesichert");
18                  Thread.sleep(generator.nextInt(2000) + 500);
19                  my_speicher.sicherungBeenden();
20                  Thread.sleep(1000);
21              } catch (InterruptedException ie) {
22              }
23          }
24      }
25  }
```

Bearbeiten Sie nun die folgenden Aufgaben:

a. Was versteht man allgemein unter einem kritischen Bereich?

Der Bereich, in dem Prozesse gleichzeitig rechnen (vor dem Deadlock).

0/2

b. Implementieren Sie den Konstruktor der Klasse `Speicher`. Verwenden Sie dabei den Code-Rahmen am Ende der Aufgabe und *kommentieren Sie Ihre Lösung*! Die Klassenattribute sind dorts bereits deklariert und müssen durch den Konstruktur initialisiert werden.

c. Implementieren Sie die Methode `schreibzugriffBeginnen(int id)`, welche den Beginn des Zugriffs eines Rechenprozesses auf den Hintergrundspeicher modelliert, sowie die die Methode `schreibzugriffBeenden(int id)`, welche im Gegenzug das Beenden des Zugriffs eines Rechenprozesses auf den Hintergrundspeicher modelliert. Beachten Sie dazu die folgenden Randbedingungen:

- Alle oben genannten Anforderungen müssen beachtet werden.
- Maximal 2 Rechenprozesse dürfen den Hintergrundspeicher gleichzeitig beschreiben.
- Solange ein Rechenprozess einen Schreibewunsch hat oder sich gerade in der Zugriffsphase befindet, darf der Datensicherungsprozess nicht aktiv werden.

Ergänzen Sie dazu den Code-Rahmen am Ende der Aufgabe und *kommentieren Sie Ihre Lösung!*
*Hinweis:* Sie können davon ausgehen, dass die Methoden `schreibzugriffBeginnen(int id)` bzw. `schreibzugriffBeenden(int id)` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Rechenprozess`).

d. Implementieren Sie nun die Methoden für den Datensicherungsprozess. Vervollständigen Sie dazu den Code-Rahmen für die Methoden `sicherungBeginnen()` und `sicherungBeenden()` in dem Code-Rahmen am Ende der Aufgabe und *kommentieren Sie Ihre Lösung*.
*Hinweis:* Sie können davon ausgehen, dass die Methoden `sicherungBeginnen()` und `sicherungBeenden()` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Datensicherungsprozess`).

Folgender Code-Rahmen steht Ihnen zur Bearbeitung der Aufgaben b), c) und d) zur Verfügung:

```
1  public class Speicher {
2      private static int maxRechenprozesse;
3      private int anzahlRechenprozesse;
4      private boolean sicherungAktiv;
5      private int anzahlSchreibewuensche;
6
7
8
9      public Speicher(                                    ) {
10
11
12
13
14
15
16
17
18
19     }
20
21     public synchronized void schreibzugriffBeginnen(int id) {
22         // Wird durch einen Rechenprozess aufgerufen.
23
24
25
26         try {
27             while (
28
29
30
31
32
33
34         } catch (InterruptedException ie) {
35         }
36
37
38
39
40
41
42
43
44         System.out.println("Rechenprozess " + id + " ==> Speicher");
45
46
47
48
49     }
50
51     public synchronized void schreibzugriffBeenden(int id) {
52         // Wird durch einen Rechenprozess aufgerufen.
53
54
55
56
57
```

```
58
59          System.out.println("Speicher ==> Rechenprozess " + id);
60
61
62
63
64
65
66      }
67
68      public synchronized void sicherungBeginnen() {
69          // Wird durch den Datensicherungsprozess aufgerufen.
70          try {
71              while (
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86      }
87
88      public synchronized void sicherungBeenden() {
89          // Wird durch den Datensicherungsprozess aufgerufen.
90          this.sicherungAktiv = false;
91
92
93
94
95
96
97
98          System.out.println("Speicher ==> Datensicherungsprozess");
99
100
101
102
103
104
105
106
107     }
108
109 }
```

anzahlRechenprozesse--; // Prozess hat Lesezugriff beendet und wird entfernt
notifyAll(); // andere Prozesse können jetzt nachrücken

anzahlSchreibbereitschaft > 0 ) {
anzahlRechenprozesse++; // Prozess kann rechnen, nachdem Schreibbereitschaft gesetzt wurde

1/4

}
} catch (InterruptedException ie) {
}
this.sicherungAktiv = true;
System.out.println("Datensicherungsprozess ==> Speicher");

wait();

notifyAll();

2/2

3