Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Linnhoff-Popien

mobile and distributed systems group

Übungsblatt 14
Betriebssysteme im WiSe 21/22

Klausurvorbereitung

Abgabetermin: Dieses Blatt dient zur Klausurvorbereitung und wird nicht abgegeben.

Ankündigungen: Das zentrale Sondertutorium findet am **07.02.22 von 18:00-20:00 Uhr** über Zoom statt. Hier können zur Vorbereitung auf die Klausur nochmals Fragen zum Stoff gestellt werden. Die Zoom-Zugangsdaten finden Sie auf Uni2Work im Abschnitt Termine.

Aufgabe Ü36: Rekursive Prozeduraufrufe

(– Pkt.)

Die Summe der Folge von Zahlen 1 bis N kann durch folgende rekursive Prozedur berechnet werden:

```
PROCEDURE Sum(n: INTEGER) RETURNS INTEGER
BEGIN
IF n <= 0 THEN RETURN 0
ELSE RETURN n + Sum(n-1)
END
```

Gehen Sie davon aus, dass aus dem Hauptprogramm ein Aufruf `Sum(4)` erfolgt.

Erstellen Sie ein Diagramm, aus welchem ausgehend vom Hauptprogramm ersichtlich wird, wann welcher Unterprogrammaufruf erfolgt, welcher Parameterwert übergeben wird, wie der Rückgabewert aus dem Unterprogrammaufruf lautet und was das Endergebnis der Berechnung ist.
Hinweis : Sie können dabei davon ausgehen, dass Ausdrücke in der Prozedur von links nach rechts ausgewertet werden.

Aufgabe Ü37: Preemptives Scheduling

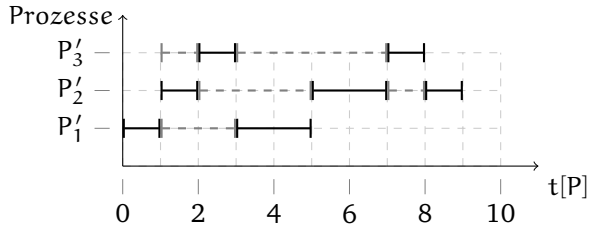
(– Pkt.)

In dieser Aufgabe sollen zwei Scheduling-Strategien untersucht werden: die preemptive Strategie **SRPT (Shortest Remaining Processing Time)** und die preemptive Strategie **RR (Round Robin)**. Dazu seien die u.g. Prozesse (nach dem Beispiel) mit ihren Ankunftszeitpunkten und Bedienzeiten (in beliebigen Zeiteinheiten) gegeben.

Beispiel: Es seien folgende Ankunfts– und Bedienzeiten für die drei Beispielprozesse P'_1, P'_2 und P'_3 gegeben:

Prozess	Ankunftszeitpunkt	Bedienzeit
P'_1	0	3
P'_2	1	4
P'_3	1	2

Das folgende Diagramm veranschaulicht ein beliebiges Scheduling der drei Prozesse P'_1, P'_2 und P'_3:



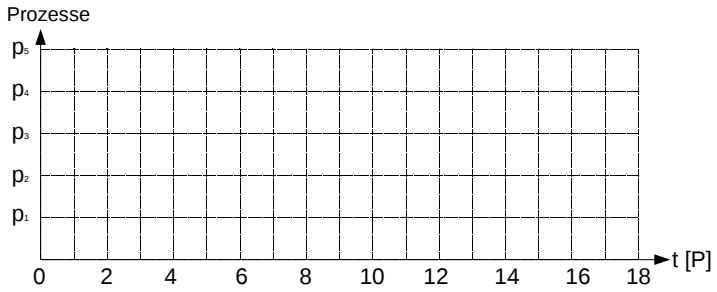
- Trifft ein Prozess zum Zeitpunkt t ein, so wird er direkt zum Zeitpunkt t berücksichtigt.
- Wird ein Prozess zum Zeitpunkt t' unterbrochen, so reiht er sich auch zum Zeitpunkt t' wieder in die Warteschlange ein.
- Sind zwei Prozesse absolut identisch bezüglich ihrer relevanten Werte, so werden die Prozesse nach aufsteigender Prozess-ID in der Warteschlange eingereiht (Prozess P_i vor Prozess P_{i+1}, usw.). Diese Annahme gilt sowohl für neu im System eintreffende Prozesse, als auch für den Prozess, dem der Prozessor u.U. gerade entzogen wird!
- Jeder Prozess nutzt sein Zeitquantum stets vollständig aus d.h. kein Prozess gibt den Prozessor freiwillig frei (Ausnahme: bei Prozessende).

Geben sei nun folgende Prozesstabelle:

Prozess	Ankunftszeitpunkt	Bedienzeit
P_1	0	10
P_2	2	1
P_3	1	2
P_4	3	1
P_5	5	2

Bearbeiten Sie unter den gegebenen Voraussetzungen nun die folgenden Aufgaben:

a. Verwenden Sie nun die **preemptive Strategie SRPT** und erstellen Sie entsprechend dem vorherigen Beispiel ein Diagramm, 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. Erstellen Sie Ihre Lösung basierend auf folgender Vorlage:



b. Geben Sie für die **preemptive Strategie RR** an, wann welchem Prozess Rechenzeit zugeteilt wird und wann die Prozesse jeweils terminieren, indem Sie Ihre Lösung wie in der vorherigen Teilaufgabe a) darstellen. 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 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.

c. Berechnen Sie als Dezimalzahl mit zwei Nachkommastellen die mittlere Verweil- und Wartezeit für die zwei Verfahren SRPT und RR.

Aufgabe Ü38: Prozessfortschrittsdiagramm

(– Pkt.)

Bearbeiten Sie die folgenden Aufgaben.

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

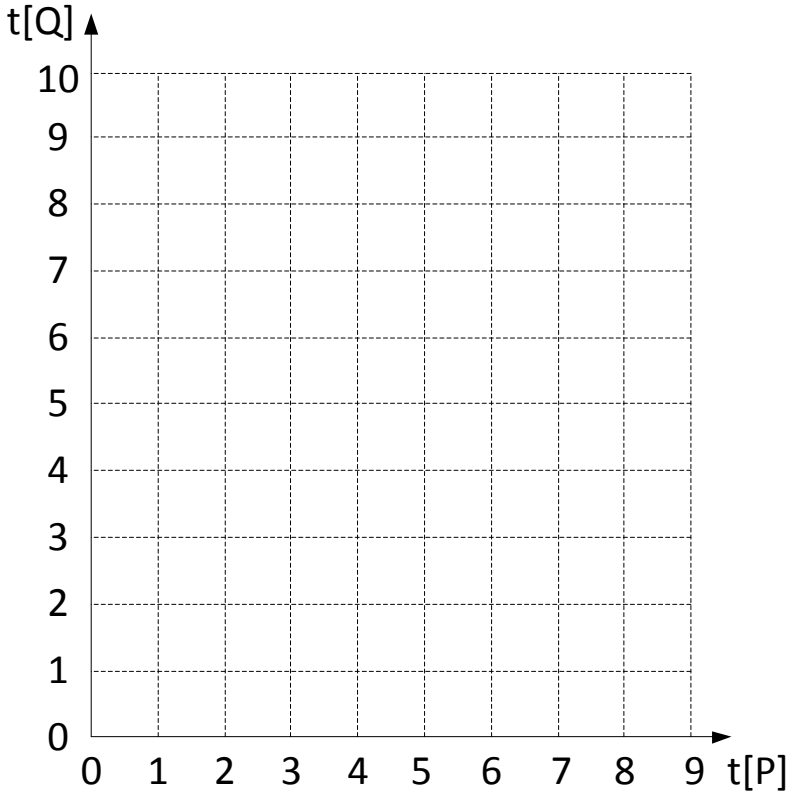
Q benötigt:

- 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 $]7;8[$.

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:



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

c. Zeichnen Sie alle *prinzipiell* unterschiedlichen Ausführungspfade (d.h., dass Prozess P und Q anders ge-scheduled werden und dementsprechend die Betriebsmittel in unterschiedlicher Reihenfolge nutzen) 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 (d.h., dass Prozess P und Q anders ge-scheduled werden und dementsprechend die Betriebsmittel in unterschiedlicher Reihenfolge nutzen), 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.

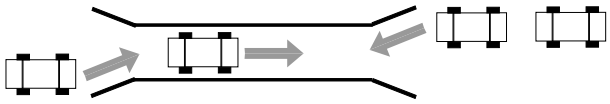
Aufgabe Ü39: Modellierung eines Petri-Netzes

(– Pkt.)

In dieser Aufgabe sollen Sie mit Hilfe eines Petri-Netzes eine Einbahnbrücke und vier Fahrzeuge modellieren. Dabei sollen folgende Bedingungen gelten.

- Über die Einbahnbrücke kann zu einem Zeitpunkt immer nur genau ein Fahrzeug fahren.
- Ein Fahrzeug darf auf der Einbahnbrücke nicht die Fahrtrichtung wechseln.
- Ein Fahrzeug, das die Brücke überquert hat, reiht sich unter den Fahrzeuge ein, die die Brücke in der anderen Richtung überqueren wollen.
- Unter den Fahrzeugen, die die Brücke überqueren wollen existiert keine Ordnung, d.h. es wird stets zufällig eine Fahrzeug ausgewählt, das als nächstes die Brücke überquert.

Die folgende Abbildung veranschaulicht eine Situation, in der sich gerade ein Fahrzeug auf der Einbahnbrücke befindet, ein Fahrzeug darauf wartet, die Brücke von links nach rechts zu überqueren und zwei Fahrzeuge darauf warten, die Brücke von rechts nach links zu überqueren.



Bearbeiten Sie unter den oben geschilderten Voraussetzungen nun folgende Aufgaben.

a. Modellieren Sie die Einbahnbrücke und die vier Fahrzeuge als ein Petri-Netz. Gehen Sie dabei davon aus, dass sich zu Beginn je zwei Fahrzeuge auf jeder Seite der Brücke befinden. Auf der Brücke selbst befindet sich zu Beginn kein Fahrzeug. Beschriften Sie alle Stellen und Transitionen ihres Petri-Netzes! **Hinweis:** Modellieren Sie die Situation in der sich ein Fahrzeug von links nach rechts bzw. von rechts nach links bewegt jeweils als eine eigene Stelle.

b. Skizzieren Sie den Erreichbarkeitsgraphen für das in Aufgabe a) modellierte Petri-Netz. Notieren Sie an jeder Kante des Erreichbarkeitsgraphen die Transition, die feuern muss, um den Übergang zwischen zwei Zuständen Ihres Petri-Netzes zu erwirken.

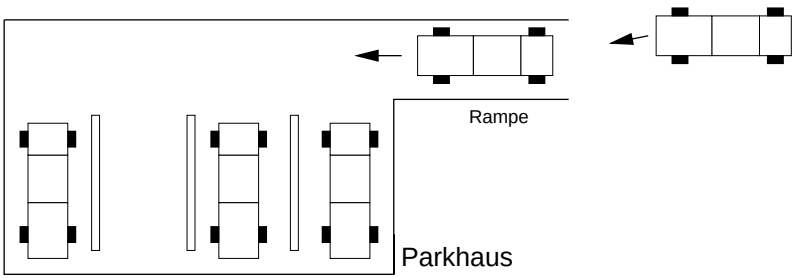
c. Handelt es sich bei Ihrer Modellierung aus Aufgabe a) um ein faires Petri-Netz? Begründen Sie Ihre Antwort!

Aufgabe Ü40: Parkhauskontrolle mit Semaphoren

(– Pkt.)

In dieser Aufgabe sollen Sie das Konzept der Semaphore am Beispiel eines Parkhauses umsetzen. Dazu soll die Ein- und Ausfahrt von Autos in ein Parkhaus mit `4` Stellplätzen simuliert werden. Autos (welche hier als Prozesse angesehen werden können) können ein- und ausfahren. Dabei dürfen sich stets maximal soviele Autos im Parkhaus befinden (inklusive eines Autos auf der Rampe), wie Stellplätze vorhanden sind. Zu einem bestimmten Zeitpunkt kann sich immer nur ein Auto auf der Rampe befinden.

Die nachfolgende Abbildung zeigt einen Beispielzustand des Parkhauses, wo drei von den vier Stellplätzen bereits belegt sind. Ein Auto befindet sich gerade auf der Einfahrt in das Parkhaus und ein Auto wartet auf das Freiwerden der Einfahrt.



a. Was sind die kritischen Bereiche bei diesem Problem?

b. Wieviele Semaphore benötigt man um ein- und ausfahrende Autos zu synchronisieren? Um welche Art von Semaphor handelt es sich jeweils?

c. Geben Sie in Pseudocode an, wie die benötigten Semaphore initialisiert werden müssen. Wählen Sie sinnvolle Bezeichner für Ihre Semaphore. Verwenden Sie folgende Notation für die Initialisierung:

Pseudocode	Beispiel	Bedeutung
`init(<semaphor>,<value>);`	`init(mutex,1);`	Initialisiert den Semaphor `mutex` mit dem Wert 1.

d. Vervollständigen Sie nun den folgenden Pseudocode, so dass dieser das Ein- und Ausfahren eines Autos simuliert und mehrere Autos stets synchronisiert werden.

```
1   auto() {
2       while(true) {
3           ...
4           <in das Parkhaus einfahren>;
5           ...
6           <parken>;
7           ...
8           <aus dem Parkhaus herausfahren>;
9           ...
10      }
11  }
```

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

e. Das Parkhaus soll nun auch LKWs die Möglichkeit zum Parken geben. Dazu werden zwei der vier vorhandenen Stellplätze vergrößert. Es gibt fortan also vier Parkplätze wovon zwei

als LKW Parkplatz genutzt werden können. LKWs können nur auf LKW Parkplätzen parken, Autos dürfen aber weiterhin auf allen Parkplätzen parken. Unabhängig von der Art kann auf einem Parkplatz immer nur ein Fahrzeug parken. Gehen Sie davon aus, dass ein Prozess stets über die boolesche Variable `is_lkw_parkplatz` testen kann, ob es sich bei dem freien Parkplatz um einen Parkplatz für LKWs handelt.

(i) Damit das Parken von LKWs und Autos weiterhin synchronisiert erfolgen kann, benötigt man weitere Semaphore. Wählen Sie geeignete Bezeichner für die neuen Semaphore und initialisieren Sie diese in Analogie zu Aufgabe c).

(ii) Geben Sie in Analogie zum Pseudocode für Autos aus Aufgabe d) den Pseudocode für LKWs an, die in dem Parkhaus parken wollen. Nennen Sie die entsprechende Funktion `lkw()`.

(iii) Verändern Sie nun den Pseudocode aus der Funktion `auto()` aus Aufgabe d) so, dass Autos bzw. LKWs (d.h. die ausführenden Prozesse der Funktionen `auto()` und `lkw()`) synchronisiert werden. Verwenden Sie dazu geeignete `if`-Konstrukte.

Aufgabe Ü41: Buddy-Systeme

(– Pkt.)

Gegeben sei ein mobiles System mit einem Hauptspeicher von 1GB = 1024 MB, der byteweise adressiert wird. Zur Speicherverwaltung werden Buddy-Systeme eingesetzt. Dabei wird immer die am weitesten links stehende Speicherzelle geteilt, wenn ein neuer Prozess eingefügt wird. Die minimale Buddy-Größe soll 64MB betragen. Bearbeiten Sie nun folgende Aufgaben:

a. Es werden 4 Prozesse der Reihe nach in das Buddy-System eingefügt:

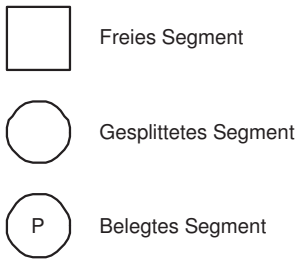
   (i) p_1: 17 MB

   (ii) p_2: 70 MB

   (iii) p_3: 80 MB

   (iv) p_4: 32 MB

   Zeichnen Sie insgesamt 4 Buddy-Bäume, die jeweils den Zustand der Speicherbelegung darstellen, nachdem ein weiterer der 4 Prozesse eingefügt wurde. Es muss genau ersichtlich sein, ob es sich um ein freies, ein gesplittetes bzw. um ein belegtes Segmente des Hauptspeichers handelt. Kennzeichnen Sie die entsprechenden Segmente eines Buddy-Baums mit den folgenden Symbolen:



b. Wieviele Bits werden zur Adressierung eines Bytes im gegebenen Speicher benötigt? Ergänzen Sie im letzten Buddy-Baum der Teilaufgabe a (also dem Buddy-Baum nach dem Einfügen von p_4) für alle freien, belegten und gesplitteten Segmente die ersten 5 Bits ihrer Speicheradresse.

c. Geben Sie für jeden der vier Prozesse an, wie viel Speicher diesem entsprechend der Zuordnung nach dem Buddy-Verfahren tatsächlich zur Verfügung stünde.

d. Die eingefügten Prozesse p_1 bis p_4 benötigen insgesamt 199 MB. Durch die Verwendung des Buddy-Baums unterscheidet sich der tatsächlich zugeordnete Hauptspeicher jedoch von diesem Wert. Wieviel von den ursprünglichen 1024 MB stehen für weitere Prozesse somit nur noch zur Verfügung und wie viel Speicher wird letztendlich verschwendet? Wie nennt man den für diese Verschwendung verantwortlichen Effekt?

e. Ein neuer Prozess p_5 mit einem Speicherbedarf von 520 MB soll gestartet werden. Ist es möglich diesen Prozess einem Buddy zuzuweisen? Falls ja, zeichnen sie den aktualisierten Buddy-Baum. Falls nein, erläutern Sie den Grund.

f. Zunächst terminieren Prozess p_1 und dann Prozess p_2. Zeichen Sie den aktualisierten Buddy-Baum nach jeder der beiden Prozessterminierungen. Kennzeichnen Sie dabei freie, gesplitteten bzw. belegten Segmente wieder mit den zuvor verwendeten Symbolen.

Aufgabe Ü42: Koordination von Threads

(– Pkt.)

In dieser Aufgabe soll das bekannte Philosophenproblem (Dining Philosophers) mit Hilfe der Programmiersprache Java implementiert werden.
Beim Philosophenproblem sitzen fünf Philosophen an einem runden Tisch beim Essen. Vor jedem Philosophen befindet sich ein Teller voller Reis und zwischen je zwei Tellern befindet sich ein Stäbchen. Ein Philosoph, der hungrig ist, benötigt sowohl das linke als auch das rechte Stäbchen, um essen zu können. Hat ein Philosoph zwei Stäbchen, so isst dieser, bis er satt ist, erst dann legt er die Stäbchen an die ursprünglichen Plätze zurück, so dass seine Nachbarn davon Gebrauch machen können.

a. Beschreiben Sie, ab wann man sich beim Philosophenproblem im kritischen Bereiche befindet und ab wann man diesen wieder verlässt.

b. Wenn man beim Philosophenproblem keine Einschränkung beim Zugriff auf die Stäbchen formuliert, kann es zu einem Deadlock kommen.

   (i) Beschreiben Sie informell einen Ablauf für das Philosophenproblem, der zu einem Deadlock führt.

   (ii) Geben Sie zwei der *effizientesten* (!) Möglichkeit an, wie sich die Deadlocksituation im speziellen Fall des Fünf-Philosophenproblems vermeiden lässt.

   (iii) Das Auftreten welcher der für einen Deadlock notwendigen Bedingungen wird durch ihre Lösungen ausgeschlossen?

c. Im Folgenden sollen Sie das Philosophenproblem in Form der drei Java Klassen `Philosoph`, `Staebchen` und `Dinner` implementieren. Der Quelltext der Klasse `Dinner` ist bereits vorgegeben und sieht folgendermaßen aus:

```
1  public class Dinner {
2     public static void main(String[] args) {
3        Thread[] philosoph = new Thread[5];
4        Staebchen staebchen[] = new Staebchen[5];
5
6        for (int i = 0; i < 5; ++i) {
7           staebchen[i] = new Staebchen();
8        }
9        for (int i = 0; i < 5; ++i) {
10          philosoph[i] =
11             new Philosoph(i, staebchen[(i+4)%5], staebchen[i]);
12          philosoph[i].start();
13       }
14    }
15 }
```

   (i) Implementieren Sie zunächst die Klasse `Staebchen`. Diese soll die beiden Methoden `get()` (Stäbchen vom Tisch nehmen) und `put()` (Stäbchen an den ursprünglichen Platz zurücklegen) zur Verfügung stellen. Verwenden Sie dazu den nachfolgenden Code-Rahmen. Achten Sie darauf, dass der Zugriff auf ein Stäbchen im wechselseitigen Ausschluss erfolgt und keine Race Conditions auftreten können! Verwenden Sie für Ihre Implementierung das vorgegebene Attribut `wirdBenutzt`.

   Kommentieren Sie Ihre Lösung ausführlich!

   Code-Rahmen der Klasse `Staebchen`

```java
public class Staebchen {

   // Attribut
   private boolean wirdBenutzt = false;

   // get()-Methode
   // ...
   // put()-Methode
   // ...
}
```

(ii) Implementieren Sie nun die Klasse `Philosoph` Achten Sie darauf, dass Ihre Implementierung frei von Deadlocks ist. Verwenden Sie dazu den nachfolgenden Code-Rahmen (Fortsetzung auf der nächsten Seite!). Fügen Sie Ihrer Implementierung alle Attribute hinzu, die notwendig sind, damit Ihre Implementierung sich mit der vorgegebenen Klasse `Dinner` fehlerfrei übersetzen lässt.

Kommentieren Sie Ihre Lösung ausführlich!

Hinweis: Die Klasse `Random` dient dazu, Zufallszahlen zu erzeugen. Die Methode `nextInt(int positiveZahl)` liefert eine Zahl zwischen 0 (inklusive) und `positiveZahl` (exklusive). Mit Hilfe dieser Methode wird die Zeit, die ein Philosoph isst bzw. denkt simuliert.

Code-Rahmen der Klasse `Philosoph`

```java
import java.util.Random;

public class Philosoph extends Thread {
   private static Random rand = new Random();

   // Attribute
   // ...
   // Konstruktor
   // ...
   // run()-Methode
   public void run() {
      try {
         while(true) {
            // denken
            System.out.println("Philosoph "+ id + " denkt");
            Thread.sleep(rand.nextInt(500) + 500);
            // hungrig (versuche Staebchen zu erhalten)
            // ...
            // essen
            System.out.println("Philosoph "+ id + " isst");
            Thread.sleep(rand.nextInt(500) + 500);
            // satt (lege Staebchen zurueck)
            // ...
         }
      }
      catch(InterruptedException ie) {
      }
   }
}
```

Aufgabe Ü43: Einfachauswahlaufgabe

(– Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwhlen („1 aus n“). Kreuzen Sie dazu die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)) an. Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Worüber können vom Hauptprogramm schnellstmöglich eine geringe Anzahl von Parametern an ein Unterprogramm übergeben werden?

(i) Stack	(ii) Spezielle CPU-Register	(iii) Speicherbereich auf der Festplatte	(iv) Speicherbereich auf einem Bandlaufwerk

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

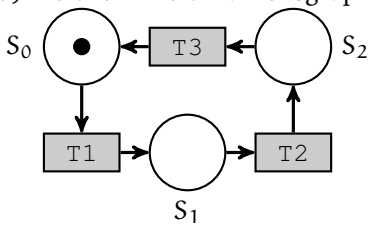
Job	durchschnittliche CPU-Auslastung	Dauer	benötigter Speicher	Platte
1	20%	10 min.	50 KBytes	-
2	45%	20 min.	100 KBytes	-
3	25%	30 min.	80 KBytes	ja

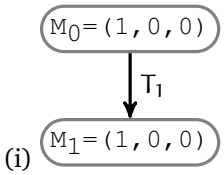
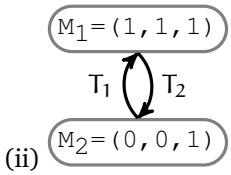
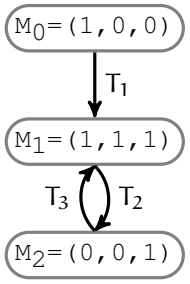
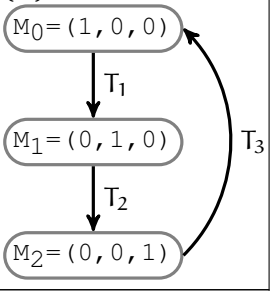
(i) 3,33 min.	(ii) 6,66 min.	(iii) 10 min.	(iv) 20 min.

c) Mit welchem Systemaufruf wird unter Unix/Linux Systemen eine identische Kopie des aufrufenden Prozesses erzeugt?

(i) `close`	(ii) `fork`	(iii) `create`	(iv) `getpid`

d) Welcher Erreichbarkeitsgraph gehört zu folgendem Petrinetz?



(i) 	(ii) 	(iii) 	(iv) 

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

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