LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN INSTITUT FUR INFORMATIK LEHRSTUHL FUR MOBILE UND VERTEILTE SYSTEME PROF. DR. CLAUDIA LINNHOFF-POPIEN	Wintersemester 2012/2013 Klausur 07.02.2013

Klausur zur Vorlesung Betriebssysteme

Aufgabe 1: Multiple Choice

(25 Pkt.)
Sind die folgenden Aussagen zum Thema Betriebssysteme wahr oder falsch?

a. In der logischen Hierarchie der Rechnerfunktionen ist das Betriebssystem in die Schicht der Anwendungsprogramme einzuordnen.

b. Das Betriebssystem kann als eine erweiterte Maschine aufgefasst werden, die aus Nutzersicht eine Komplexitätsreduktion bewirkt.

c. Anwendungsprogramme können auf die Hardware nur indirekt zugreifen, indem sie Systemaufrufe beim Betriebssystem tätigen.

d. Ein Betriebssystem verwaltet Speicher, E/A-Geräte, Dateien und Prozesse und muss zu diesem Zweck geeignete Kontrollstrukturen bereitstellen.

e. Ein Betriebssystem ist unabhängig von den unterstützten Maschinenbefehlen der physikalischen Geräte, auf denen es ausgeführt wird.

Sind die folgenden Aussagen zum Thema Prozesse und Threads wahr oder falsch?

a. Bei User-Level-Threads ist das Thread-Management Aufgabe der Anwendung.

b. Ist ein Kernel-Level-Thread eines Prozesses blockiert, so sind stets auch alle anderen Kernel-Level-Threads dieses Prozesses blockiert.

c. Threads eines Prozesses verwalten ihre Daten innerhalb des identischen Adressraums.

d. Es ist möglich, dass ein Prozess terminiert, aber seine Threads noch weiterlaufen.

e. Beim Thread-Wechsel zwischen verschiedenen Kernel-Level-Threads ist kein Moduswechsel erforderlich.

Sind die folgenden Aussagen zum Thema Prozesszustandsmodelle wahr oder falsch?

a. Die Verwaltung von preemtiven Modi erzeugt aus Sicht des Betriebssystems insgesamt weniger Overhead durch Verwaltungsaufgaben.

b. Dispatching ist die Auswahl eines rechenbereiten Prozesses, der als nächstes dem Prozessor zugewiesen werden soll.

c. Das 5Zustands-Proessmodell eignet sich besonders für auf Stapelverarbeitung basierende Systeme.

d. Im 7-Zustands-Pmzasmodefl erlaubt es neue Prozesse in das System aufzunehmeu, obwohl der Hauptspeicher aktuell schon komplett belegt ist.

e. lm 7-Zustands-Prozessmodell sollte immer dann ein Prozess von "Blocked, Suspended" nach "Blocked" überführt werden, wenn das von ihm erwartete Ereignis vermutlich bald eintrifft und genügend Kapazität im Hauptspeicher frei ist.

Sind die folgenden Aussagen zum Thema Speicher wahr oder falsch?

a. Bei der Abbildung von virtuellen auf reale Adresse muss nur der Offset ersetzt werden.

b. Ist in Systemen mit virtueller Speicherverwaltung der logische Adressraum größer als der physische Adressraum, so können insgesamt mehr Prozesse ausgeführt werden, als dies ohne virtuelle Speicherverwaltung der Fall wäre.

c. Die Größe der Seitenrahmen sollte in Systemen mit virtueller Speicherverwaltung immer der Größe der Seiten entsprechen.

d. Je kleiner die Seitengrößen festgelegt werden, um so wahrscheinlicher ist es, dass interne Fragmentierung auftritt.

e. Die Anzahl der Bits, die für den Offset reserviert sind, sowie die Wortlänge erlauben zusammen eine Aussage über die Größe der verwendeten Seitenrahmen.

Sind die folgenden Aussagen zum Thema Prozesskoordination wahr oder falsch?

a. Lässt man zu, dass zwei Prozesse den kritischen Bereich gleichzeitig betreten, so kann ein Deadlock entstehen.

b. Beim Warten auf die Erlaubnis zum Betreten des kritischen Bereichs verschwenden Prozesse im Algorithmus von Peterson keine CPU-Zeit.

c. Die kritischen Bereiche eines Prozesses stehen erst zur Laufzeit fest.

d. Der Algorithmus von Peterson wird verwendet, um das Betreten des kritischen Bereichs von Prozessen zu regeln.

e. In größeren Programmen ist die Verwendung des Algorithmus von Peterson übersichtlicher als die Verwendung von Semaphoren.

Aufgabe 2: Multilevel Feedback Queuing
(20 Pkt.)
In dieser Aufgabe sollen Sie ein modifiziertes MLFQ-Verfahren anwenden, welches auf unterschiedlichen Leveln verschieden lange Zeitscheiben verwendet. Die drei folgenden Prioritätsklassen stehen zur Verfügung:

Priorität	Verfahren
0	FIFO mit Quantum 1
1	FIFO mit Quantum 1
2	FIFO mit Quantum 1
3	Round Robin mit Quantum 2

Das Quantum gibt die Dauer der Zeitscheibe auf dem entsprechenden Level wieder.
Gehen sie für die folgenden Teilaufgaben von folgenden Annahmen aus:

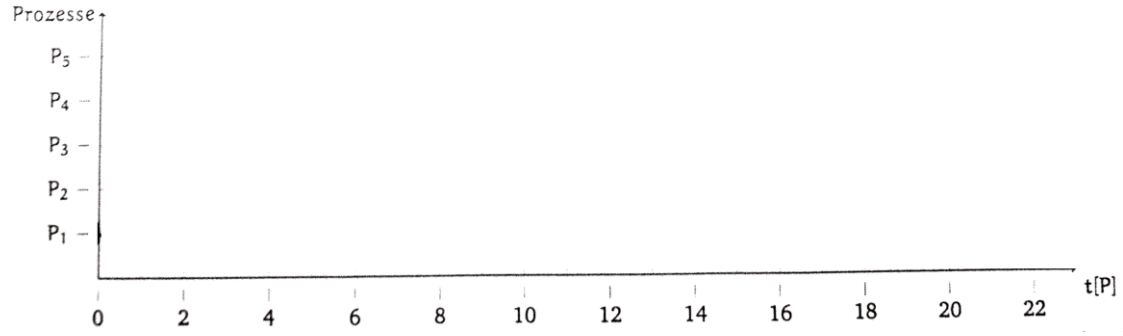
- Jeder Prozess nutzt sein Zeitquantum stets vollständig aus d.h. kein Prozess gibt den Prozessor freiwillig frei (Ausnahme: bei Prozessende)
- Trifft ein Prozess zum Zeitpunkt t ein, so wird er direkt zum Zeitpunkt t berücksichtigt.
- Wird ein Prozess zum Zeitpunkt t' unterbrochen, so reiht er sich 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 Pi vor Prozess Pi+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!

Bearbeiten Sie unter den gegebenen Voraussetzungen nun die folgenden Aufgaben:

a. Gegeben seien die Prozesse P1, . . . , P5 mit den folgenden Ankunfts- und Rechenzeiten:

Prozess	Ankunftszeitpunkt	Rechenzeit
P1	0	10
P2	1	5
P3	6	3
P4	10	2
P5	14	2

Veranschaulichen Sie die Zuteilung der Rechenzeit an die Prozesse für die oben beschriebene Variante des MLFQ-Verfahrens im nachfolgenden Diagramm. Kennzeichnen Sie zudem für jeden Prozess seine Ankunftszeit.

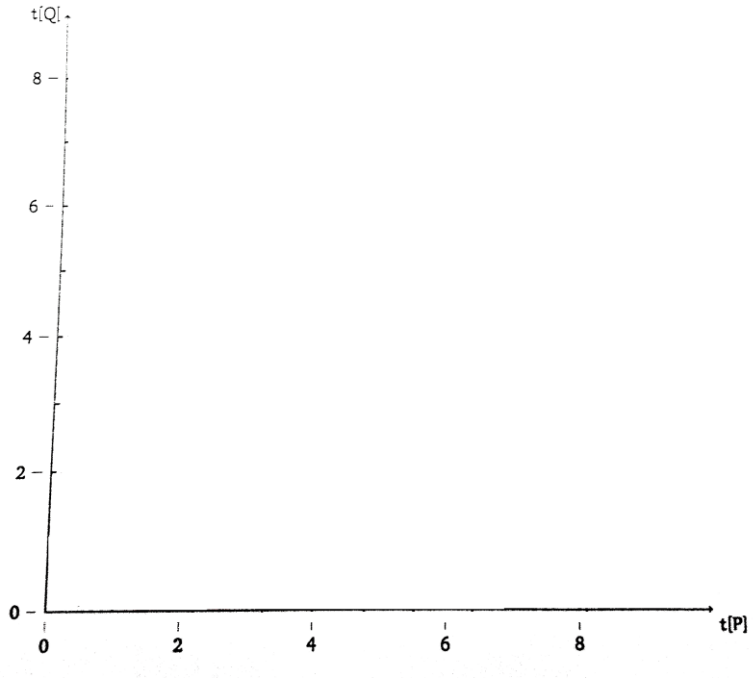


b. Geben Sie für die Prozesse P1 und P2 jeweils die Verweilzeit (Antwortzeit) und die Wartezeit an.

c. Mit welcher Strategie ist Round Robin mit einem Quantum Q = ∞ gleichzusetzen?

Aufgabe 3: Prozesskoordination
(15 Pkt.)
Beantworten Sie die folgenden Fragen zum Thema Prozesskoordination.
Gegeben seien zwei Prozesse P und Q, die auf einem Uniprozessorsystem ausgeführt werden sollen. Prozess P benötigt zu seiner Ausführung neun Zeiteinheiten, Prozess Q acht Zeiteinheiten. Es stehen die Betriebsmittel BM 1-6 zur Verfügung, die von den Prozessen während ihrer Ausführung benötigt werden. P benötigt BM1 im Zeitraum 1 bis 3, BM2 im Zeitraum 2 bis 3, BM3 im Zeitraum 1 bis 3, BM4 im Zeitraum 4 bis 6, BM5 im Zeitraum 5 bis 6 und BM6 im Zeitraum 7 bis 8. Q benötigt BM1 im Zeitraum 1 bis 2, BM2 im Zeitraum 4 bis 6, BM3 im Zeitraum 6 bis 7, BM4 im Zeitraum 4 bis 6, BM5 im Zeitraum 3 bis 4 und BM6 im Zeitraum 5 bis 7.

Skizzieren Sie das Prozessfortschrittsdiagramm für die oben beschriebenen Anforderungen. Gehen Sie davon aus, dass der Scheduler die Prozess 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.



b. Kennzeichnen Sie alle unmöglichen und unsichere Bereiche im Diagramm aus Aufgabe a).

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

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

e. Nennen Sie zwei Möglichkeiten, wie eine Deadlocksituation auf einem Uniprozessorsystem vermieden werden kann.

f. Bezogen auf Aufgabe a): Gehen Sie nun davon aus, dass die beiden Prozesse P und Q auf einem Multiprozessorsystem ausgeführt werden. Geben sie die minimale Anzahl an Zeiteinheiten an, die zur vollständigen Abarbeitung der Prozesse P und Q (unter der Annahme eines optimalen Schedulings) benötigt werden.

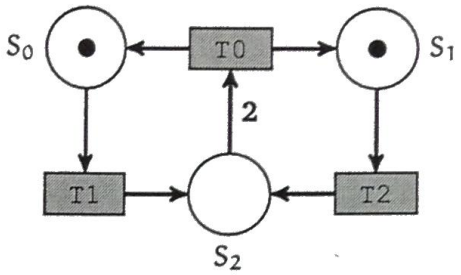
Aufgabe 4: Petrinetze
(20 Pkt.)
a. Modellierung einer Druckerwarteschlange
Gehen Sie für die folgenden beiden Teilaufgaben von folgender Situation aus:
Ein Computer ist mit einem Drucker verbunden. Um zu vermeiden, dass der Computer einen Druckauftrag an den Drucker sendet, während er gerade noch mit einem anderen Druckauftrag beschäftigt ist, schickt der Computer seine Druckaufträge an eine Warteschlange. Wenn die Warteschlange nicht leer ist, entfernt der Drucker einen Druckauftrag aus der Warteschlange und druckt diesen.
(i) Modellieren Sie den oben beschriebenen Sachverhalt mit einem Petrinetz. Gehen Sie zudem von folgenden Bedingungen aus:

- Der Computer kann immer nur einen Druckauftrag in die Warteschlange stellen.
- Er kann diesen Schritt aber beliebig oft wiederholen.
- Der Drucker hat zwei Zustände: Entweder druckt er oder er wartet auf den nächsten Druckauftrag.
- Die Warteschlange besitzt eine unendliche Kapazität, d.h. es können beliebig viele
- Druckaufträge in die Warteschlange gestellt werden.
- Es spielt keine Rolle in welcher Reihenfolge der Drucker die Druckaufträge aus
- der Warteschlange entfernt.

Verwenden Sie zur Modellierung des Petrinetzes eine minimale Anzahl an Stellen, Marken und Transitionen.

(ii) Damit die Warteschlange nicht überläuft, soll deren Kapazität nun begrenzt werden
Erweitern sie Ihr Petrinetz nun so, dass die Warteschlange nur noch maximal drei Aufträge akzeptiert.
Verwenden Sie zur Modellierung wieder minimale Anzahl an Stellen, Marken und Transitionen.
Führen Sie dabei keine Begrenzung der Kapazität der Stellen ein.

b. Erstellung eines Erreichbarkeitsgraphen:
Bearbeiten Sie die folgenden beiden Teilaufgaben in Bezug auf das unten abgebildete Petrinetz.

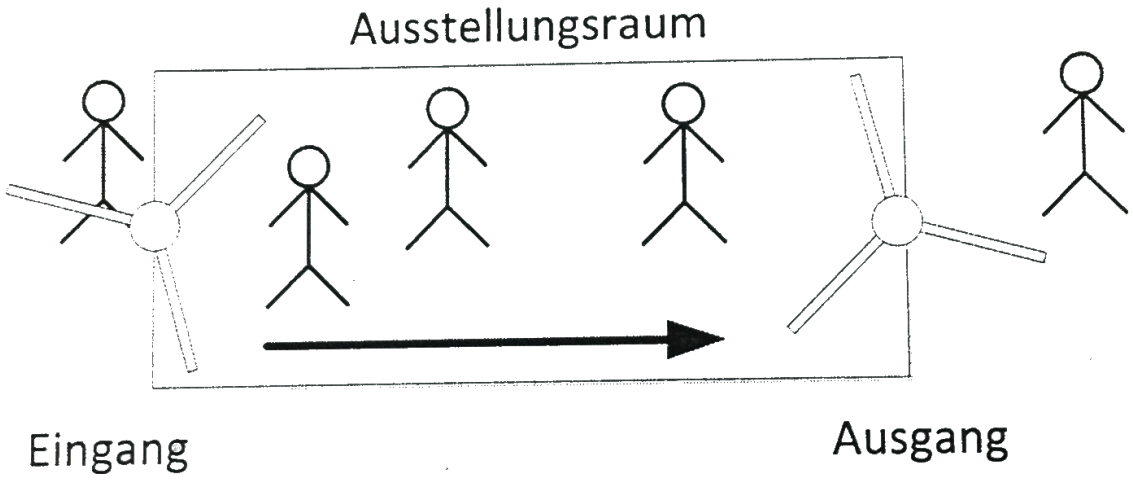


(i)
Geben Sie den Erreichbarkeitsgraphen für das dargestellte Petrinetz an. Verwenden Sie für eine Markierung Mj folgende Anordnung der Stellen Si(i ϵ {0, . . . , 2}):

Mj = (S0, S1, S2)
Der oben abgebildete Zustand entspricht also z.B. folgender Markierung:
M0 = (1,1,0)
Kennzeichnen Sie jede Kante zwischen zwei Markierungen mit der entsprechenden Transition, die für den Übergang von einer Markierung zur anderen verantwortlich ist.

(ii) Kann im dargestellten Petrinetz ein Deadlock entstehen? Begründen Sie Ihre Antwort.

Aufgabe 5: Koordination von Threads
(30 Pkt.)
In dieser Aufgabe soll ein Ausstellungsraum in einem Museum simuliert werden, in dem aufgrund des Platzmangels die Anzahl der aktuellen Besucher zu jedem Zeitpunkt maximal maxBesucher betragen darf. Die einzelnen Besucher (die als Java Threads realisiert werden sollen) besitzen Tickets mit einer eindeutigen Ticketnummer. Sie können den Ausstellungsraum durch ein Drehkreuz am Eingang betreten, indem sie ihr Ticket an einen dort angebrachten Ticketleser halten. Am anderen Ende des Raums können die Besucher den Ausstellungsraum über den Ausgang wieder verlassen, indem Sie ihr Ticket an den Ticketleser des dort installierten Drehkreuzes halten, welches anschließend das Passieren erlaubt. Der Raum ist somit nur in eine Richtung passierbar. Die Drehkreuze sind so konstruiert, dass immer nur genau eine Person passieren kann, nachdem sie ihr Ticket an den jeweiligen Ticketleser gehalten hat. Im Ausstellungsraum dürfen die Besucher für eine beliebige Zeitspanne verweilen - somit können Besucher den Ausstellungsraum in einer anderen Reihenfolge verlassen, als sie ihn betreten haben. In folgender Abbildung ist ein solcher Ausstellungsraum mit einer maximalen Anzahl an gleichzeitigen Besuchern von maxBesucher = 4 dargestellt, wobei sich gegenwärtig nur anzahlBesucher = 3 Besucher in dem Ausstellungsraum befinden. Ein Besucher ist gerade dabei das Drehkreuz am Eingang zu passieren.



Im folgenden soll eine Klasse Ausstellungsraum implementiert werden Die Beispielimplementierungen der Klassen Besucher und Museum soll Ihnen verdeutlichen, wie die Klasse Besucher verwendet werden kann:

Die Klasse Museum:

```
public class Museum {
        private Ausstellungsraum my_ausstellungsraum;
        private Besucher[] besucher;

        public Museum() {
                my_ausstellungsraum = new Ausstellungsraum(4);
                besucher = new Besucher[10];
                for(int i = O; i < besucher.length; i++) {
                        besucher[i] = new Besucher(my_ausstellunqsraum, i);
                        besucher[i].start();
                }
        }

        public static void main(String[] args) {
                new Museum();
```

}

Die Klasse Besucher:

```
import java.util.Random;
publıc class Besucher extends Thread (
        private Ausstellunqsraum my_ausstellungsraum;
        private int ticket_nummer;
        private Random generator;

        public Besucher(Ausstellungsraum ausstellungsraum, int ticket_nummer) {
        this.my_ausstellungsraum = ausstellungsraum;
        this.ticket_nummer = ticket_nummer;
        generator = new Random();
        }

        public void run() {
                try {
                        Thread.sleep(generator.nextInt(2000) + 500);
                        my_ausstellungsraum.betreten(ticket_nummer);
                        Thread.sleep(generator.nextInt(2000) + 500);
                        my_ausstellungsraum.verlassen(ticket_nummer);
                }
                catch(InterruptedException ie) {}
        }
}
```

a. Was sind die kritischen Bereiche bei diesem Problem?

b. Überlegen Sie sich die notwendigen Attribute der Klasse Ausstellungsraum und implementieren Sie deren Konstruktor. Verwenden Sie dabei den Coderahmen am Ende der Aufgabe und kommentieren Sie Ihre Lösung ausführlich!

c. Implementieren Sie die Methode betreten(int ticket_nummer) , welche das Passieren des Drehkreuzes am Eingang des Ausstellungsraums modelliert, sowie die die Methode verlassen(int ticket_nummer), welche im Gegenzug das Verlassen des Ausstellungsraums über das Drehkreuz am Ausgang modelliert. Beachten Sie bei der Implementierung, dass alle oben genannten Bedingungen eingehalten werden. Schreiben Sie die Methode betreten (int ticket_nummer) so, dass sie die eindeutige Ticketnummer des eintretenden Besuchers sowie die Anzahl der sich gerade im Ausstellungsraum befindlichen Besucher (inklusive des eintretenden Besuchers) auf der Konsole ausgibt. Schreiben sie analog die Methode verlassen(int ticket_nummer) so, dass sie die Ticketnummer des Besuchers, der gerade das Drehkreuz am Ausgang passiert sowie die Anzahl der sich gerade im Ausstellungsraum befindlichen Besucher (exklusive des Besuchers der den Ausstellungsraum gerade verlässt) auf der Konsole ausgibt. Sie können davon ausgehen, dass die Methoden betreten(int ticket_nummer) bzw. verlassen(int ticket_nummer) immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse Besucher). Verwenden Sie dabei den Coderahmen am Ende der Aufgabe und kommentieren Sie Ihre Lösung ausführlich!

d. Erläutern Sie kurz (2 Sätze), warum Ihre Lösung des wechselseitigen Ausschlusses die Bedingung der Mutual Exclusion erfüllt.

Folgender Coderahmen steht Ihnen zur Bearbeitung der Aufgabe b) und c) zur Verfügung:

```
public class Ausstellungsraum {
        // Attribute
        // Teilaufgabe b)









        // Konstruktor
        public Ausstellungsraum(int maxBesucher) {









        }

        // Methoden
        public synchronized void betreten(int ticket_nummer) {
        // Teilaufgabe c)









        }
```

```
public synchronized void verlassen(int ticket_nummer) {
// Teilaufgabe c)









}
```

Aufgabe 6: Seitenersetzungsstrategien
(26 Pkt.)

Gegeben seien eine Menge an Seiten N = { 0,1,2,3,4 } und eine Menge der im Arbeitsspeicher zur Verfügung stehenden Seitenrahmen F = { f0,f1,f2 }. Auf die Seiten wird in der folgenden Reihenfolge zugegriffen:

W = 2 0 3 1 4 1 3 4 0 3 4 2 2 3 4 0

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

a. Bei der Paging-Strategie FIFO (First In, First Out) wird bei einem Seitenfehler diejenige Seite im Arbeitsspeicher ersetzt, deren Zeitpunkt der Zuordnung zu einem Seitenrahmen am längsten zurück liegt.

Ermitteln Sie die Anzahl der Seitenfehler für die Seitenersetzungsstrategie FlFO, indem Sie alle Veränderungen im Speicher dokumentieren. Vervollständıgen Sie dazu die folgende Tabelle, indem Sie jede referenzierte Seite dem entsprechenden Seitenrahmen fi (i ϵ {0,…,2}) zuordnen und den Zeitpunkt t der Zuordnung dokumentieren. Geben Sie zudem nach jedem Seitenzugriff die aktuelle Summe an Seitenfehlern an.

Achtung: Bereits in den Hauptspeicher geladene Seiten durfen nicht von einem Seitenrahmen in den anderen verschoben werden!

Zeit	Referenzierte Seite	f0, t	f1, t	f2, t	Summe Seitenfehler
1	2				
2	0				
3	3				
4	1				
5	4				
6	1				
7	3				
8	4				
9	0				
10	3				
11	4				
12	2				
13	2				
14	3				
15	4				
16	0				

b. Ermitteln Sie die Anzahl der Seitenfehler für die Seitenersetzungsstrategie LRU (Least Recently Used), indem Sie alle Veränderungen im Speicher dokumentieren. Vervollständigen Sie dazu die folgende Tabelle, indem Sie jede referenzierte Seite dem entsprechenden Seitenrahmen fi (i ϵ {0, . . . ,2}) zuordnen und den Zeitpunkt t eines Seitenzugriffs dokumentieren. Geben Sie zudem nach jedem Seitenzugriff die aktuelle Summe an Seitenfehlern an.

Achtung: Bereits in den Hauptspeicher geladene Seiten dürfen nicht von einem Seitenrahmen in einen anderen verschoben werden! Stehen mehrere Seitenrahmen zur Auswahl, so wird der mit der kleineren Nummer gewählt.

Zeit	Referenzierte Seite	f0, t	f1, t	f2, t	Summe Seitenfehler
1	2				
2	0				
3	3				
4	1				
5	4				
6	1				
7	3				
8	4				
9	0				
10	3				
11	4				
12	2				
13	2				
14	3				
15	4				
16	0				

c. Nun soll theoretisch untersucht werden, wieviele Seitenfehler für das Beispiel aus Teilaufgabe b entstehen, wenn das System 1,2,3,4 oder 5 Seitenrahmen besäße. Der zugehörige Distance String lautet ∞ ∞ ∞ ∞ ∞ 2 3 3 4 3 3 5 1 3 3 4. Der Rechenweg muss klar nachvollziehbar sein.

d. Welchen Effekt im Zusammenhang mit der Anzahl der Seitenrahmen eines Systems beschreibt die Belady's Anomalie? Kann diese Anomalie mit der Seitenersetzungsstrategie LRU (Least Recently Used) auftreten?