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

mobile and distributed systems group

Betriebssysteme im Wintersemester 2015/2016
Übungsblatt 11

Abgabetermin: 18.01.2016, 16:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 11. – 15. Januar
Besprechung der H-Aufgaben in den Tutorien vom 18. – 22. Januar

Aufgabe 47: (T) Algorithmus von Peterson

(– Pkt.)

Druckaufträge werden vom Betriebssystem in einer (FIFO-)Warteschlange `W` verwaltet. Die Warteschlange verwaltet selbst lediglich eine Liste von Zeigern, die auf den Speicherbereich verweisen, an dem die zu druckenden Daten liegen. Die Variable `next` enthält den Index der nächsten freien Position in der Warteschlange.
Gegeben seien zwei Prozesse P_1 und P_2, die jeweils eine Datei drucken möchten. Die folgende Tabelle illustriert die Programmausschnitte, die die Prozesse P_1 bzw. P_2 dazu jeweils ausführen.

Prozess $\mathbf{P_1}$

```
1 ...
2 W[next] = ptr_file1;
3 next = next + 1;
4 ...
```

Prozess $\mathbf{P_2}$

```
1 ...
2 W[next] = ptr_file2;
3 next = next + 1;
4 ...
```

a. Welches Problem kann auftreten, wenn P_1 und P_2 im Mehrprogrammbetrieb parallel ausgeführt werden? Modellieren Sie einen Ablauf, der dieses Problem illustriert. Verwenden Sie dazu folgende Darstellung:

aktiver Prozess	ausgeführte Codezeile	Inhalt von `W`	Wert von `next`	Kommentar
...	...	...	...	...

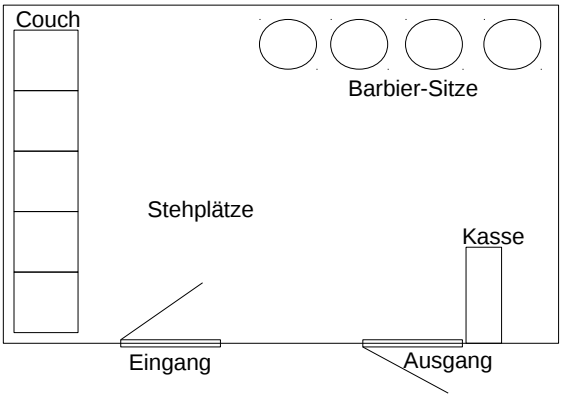
Stellen Sie den Inhalt von `W` als Liste der Form `[ptr_1, ptr_2, ...]` dar.

b. Synchronisieren Sie die Prozesse mit dem Algorithmus von Peterson. Geben Sie dazu (in Analogie zu den Code-Ausschnitten der Angabe) die Codeausschnitte der Prozesse P_1 und P_2 an.

c. Welchen erheblichen Nachteil hat der Peterson-Ansatz?

Aufgabe 48: (T) Barbierladen

(– Pkt.)



Ein Beispiel für die Nutzung von Semaphoren zur Synchronisation ist das **Problem des schlafenden Friseurs**. Unser Barbierladen habe vier Sitze, vier Barbiere und einen Warteraum. Im Warteraum steht eine Couch, die fünf Kunden Platz zum Sitzen bietet. Außerdem ist auch noch Raum zum Stehen vorhanden. Brandschutzvorschriften beschränken die Gesamtanzahl von Kunden im Laden auf 18 Personen. Es gibt genau eine Kasse, an der alle Kunden bezahlen können. Die Barbiere verbringen ihre Zeit mit Haareschneiden, Kassieren und Warten auf einen Kunden. Folgende Verhaltensregeln erlauben den reibungslosen Betrieb:

- Ein Kunde darf den Laden nicht betreten, wenn die maximal erlaubte Zahl an Kunden bereits erreicht ist.
- Hat ein Kunde einmal den Laden betreten, setzt er sich auf die Couch. Ist die Couch schon voll bleibt er im Warteraum stehen.
- Wenn ein Barbier frei ist, wird als nächstes einer der auf der Couch sitzenden Kunden bedient. Wenn es stehende Kunden gibt, wird der frei gewordene Platz auf der Couch von einem der stehenden Kunden eingenommen.
- Wenn ein Kunde bedient wurde, kann er bei jedem Barbier seine Rechnung bezahlen. Da es nur eine Kasse gibt, kann jedoch nur ein Kunde gleichzeitig bezahlen.

Im Folgenden sollen Sie eine Lösung für dieses Synchronisationsproblem erarbeiten, das nur auf Semaphoren basiert.
Dazu sollen Sie die drei Prozeduren so `customer()`, `barbier` und `cashier()` in Pseudocode implementieren, so dass der Wechselseitige Ausschluß für alle Ressourcen gewährleistet ist.
Verwenden Sie folgende Notation für den Zugriff auf Ihre Semaphoren:

Pseudo Code	Beispiel	Bedeutung
`init(<semahor>, <value>)`	`init(max_capacity, 18)`	Initialisiert die Semaphor `max_capacity` mit dem Wert 18
`wait(<semaphor>)`	`wait(barber_chair)`	Erniedrigt den Wert der Semaphor `barber_chair` um eins
`<<action>>`	`<sit in barber chair>`	Führe die gelistete Aktion aus

a. Um das Problem zu lösen benötigen Sie insgesamt 9 Semaphore. Zählen Sie diese 9 Semaphore auf, geben Sie jeweils den Funktionsaufruf zur Initialisierung der entsprechenden Semaphore an und beschreiben Sie den Zweck der Semaphore. Geben Sie zudem an, um welche Art von Semaphor es sich jeweils handelt.

b. Implementieren Sie nun den Pseudocode der Prozeduren `customer()`, `barber()` und `cashier()` mit Hilfe der oben gelisteten Zugriffsmöglichkeiten für Semaphore, so dass Ihr Code das Barbier-Problem löst. Dabei kann die Reihenfolge in der die Kunden eintreffen vernachlässigt werden.

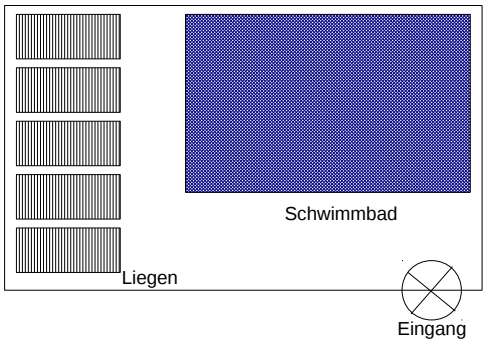
Aufgabe 49: (H) Schwimmbad

(9 Pkt.)

In dieser Aufgabe sollen Sie das Konzept der Semaphore am Beispiel eines Schwimmbads umsetzen. Dazu soll das Betreten und Verlassen eines Schwimmbads simuliert werden, welches 5

Liegen bereitstellt, die von den Badegästen genutzt werden können. Die Badegäste (welche hier als Prozesse angesehen werden können) können das Schwimmbad über ein Drehkreuz betreten bzw. verlassen, das zu jedem Zeitpunkt nur Platz für eine Person bietet. Daher kann zu einem bestimmten Zeitpunkt immer nur eine Person durch das Drehkreuz gehen. Es dürfen sich stets auch nur maximal soviele Personen im Schwimmbad befinden (inklusive einer Person im Drehkreuz), wie Liegen vorhanden sind.

Die nachfolgende Abbildung zeigt den schematischen Aufbau.



Beantworten Sie nun auf Grundlage dieses Szenarios folgende Aufgaben:

a. Welches klassische Problem aus der Informatik wird hier beschrieben?

b. Was sind die kritischen Bereiche bei diesem Problem?

c. Wieviele Semaphoren benötigt man um die Badegäste, die durch das Drehkreuz gehen wollen zu synchronisieren? Um welche Art von Semaphor handelt es sich jeweils?

d. 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.

e. Vervollständigen Sie nun den folgenden Pseudocode, so dass dieser das Betreten bzw. Verlassen eines Badegastes simuliert und mehrere Gäste stets synchronisiert werden.

```
1  badegast() {
2      while(true) {
3          ...
4          <das Schwimmbad betreten>;
5          ...
6          <Liege belegen>;
7          ...
8          <das Schwimmbad verlassen>;
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

f. Das Schwimmbad will sein Image verbessern und familienfreundlicher werden. Deshalb bietet es nun auch spezielle Kinderliegen an, die auch nur von Kindern benutzt werden dürfen. Dazu werden zwei der fünf vorhandenen Liegen zu Kinderliegen verkleinert. Es gibt fortan also fünf Liegen wovon zwei nur von Kindern genutzt werden können. Kinder können natürlich auf allen Liegen PLatz nehmen. Erwachsene Badegäste hingegen sind für die Kinderliegen zu groß und dürfen nur auf großen Liegen Platz nehmen. Unabhängig davon kann aber immer nur eine Person eine Liege besetzen. Gehen Sie davon aus, dass eine Person stets über die boolesche Variable `is_Kinderliege` testen kann, ob es sich bei einer freien Liege um eine Kinderliege oder um eine große Liege handelt.

(i) Damit das Belegen von Liegen im Schwimmbad weiterhin synchronisiert erfolgen kann, benötigt man weitere Semaphore. Wählen Sie geeignete Bezeichner für die neuen Semaphore und initialisieren Sie diese in Analogie zu Aufgabe d).

(ii) Geben Sie in Analogie zum Pseudocode für Badegäste aus Aufgabe e) den Pseudocode für Kinder an, die ins Schwimmbad wollen. Nennen Sie die entsprechende Funktion `kind()`.

(iii) Da erwachsene Badegäste nun auf die Kinder Rücksicht nehmen müssen, ist es erforderlich, dass Sie ihren Pseudocode `badegast()` aus Aufgabe e) so anpassen, dass erwachsene Badegäste bzw. Kinder (d.h. die ausführenden Prozesse der Funktionen `badegast()` und `kind()`) synchronisiert werden.

Aufgabe 50: (H) Einfachauswahlaufgabe: Prozesskoordination

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Welche Art von Prozessen macht die Synchronisation der Prozesse untereinander erforderlich?			
(i) unabhängige Prozesse	(ii) sequenzielle Prozesse	(iii) nebenläufige Prozesse	(iv) parallele aber unabhängige Prozesse
b) Wie bezeichnet man einen Speicher, der in Form eines eindimensionalen Arrays unter Verwendung der Modulo-Funktion realisiert wird?			
(i) Ringpuffer	(ii) Linearpuffer	(iii) Wechselpuffer	(iv) Sparpuffer
c) Wie bezeichnet man die Phase, in der sich zu einem Zeitpunkt nur ein Prozess befinden darf, da sich sonst z.B. inkonsistente Zustände bei gemeinsam genutzten Datenstrukturen ergeben?			
(i) einfacher Bereich	(ii) schwieriger Bereich	(iii) unkritischer Bereich	(iv) kritischer Bereich
d) Für eine korrekte Lösung des wechselseitigen Ausschlusses müssen die drei Bedingungen Mutual Exclusion, Progress, und Bounded Waiting erfüllt sein. Welche Bedingung(en) erfüllt der Algorithmus von Decker (erster Ansatz) nicht?			
(i) Mutual Exclusion	(ii) Progress	(iii) Bounded Waiting	(iv) alle drei
e) Für eine korrekte Lösung des wechselseitigen Ausschlusses müssen die drei Bedingungen Mutual Exclusion, Progress, und Bounded Waiting erfüllt sein. Was trifft auf den Algorithmus von Peterson zu?			
(i) Er erfüllt alle Bedingungen.	(ii) Er erfüllt keine der Bedingungen.	(iii) Er erfüllt nur die Mutual Exclusion Bedingung.	(iv) Er erfüllt nur die Progress Bedingung.