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

mobile and distributed systems group

Tutoriumsblatt 10
Betriebssysteme im WiSe 2020/2021

Zu den Modulen L, M

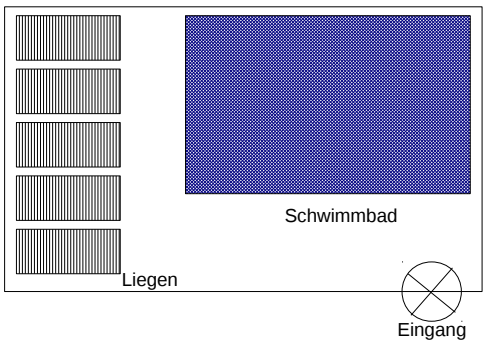
Tutorium: Die Aufgaben werden in einem Tutorien-Video vorgestellt, das am 20. Januar 2021 (17 Uhr) veröffentlicht wird.

Aufgabe T23: Schwimmbad

(– 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 T24: Nachbildung von Zählsemaphoren mittels Monitoren in Java

(– Pkt.)

Schreiben Sie eine Java Klasse `SimpleCountingSemaphore`, die es ermöglicht, Instanzen zu erzeugen, welche in Analogie zu der von Dijkstra vorgeschlagenen Datenstruktur eines Zählsemaphor verwendet werden können. Verwenden Sie dazu den Java `synchronized`-Mechanismus. Verwenden Sie außerdem eine *minimale* Anzahl an `wait()`- und `notify()`-Aufrufen!

Hinweis: Für die Lösung dieser Aufgabe reicht es aus, wenn Sie das auftreten einer etwaigen `InterruptedException` ohne weitere Fehlerbehandlung abfangen.

Des Weiteren sollen Sie sich in dieser Aufgabe die Unterschiede und Gemeinsamkeiten zwischen Java-Synchronisation und Monitoren klar machen.

a. Beschreiben Sie das grundlegende Konzept der Synchronisation in Java. Gehen Sie dabei auf die Verwendung von Objekt-Locks, Threads und Warteschlangen ein.

b. Wie werden `wait()` und `signal()` in Java umgesetzt?

c. Erläutern Sie den grundlegenden Unterschied des Java Synchronisationsmechnismus im Gegensatz zu „echten“ Monitoren (ohne Beachtung der Signalisierungsmechanismen).

d. Beschreiben Sie die Einschränkungen bei der Verwendung von `wait()` und `notify()` gegenüber „echten“ Monitoren.

Hinweis: Überlegen Sie sich dazu, wie im Falle echter Monitore bzw. im Falle des Java Synchronisierungsmechanismus der nächste aktive Thread selektiert wird.

e. Überlegen Sie sich, wie diese Einschränkungen umgangen werden können.

f. Es gibt zwei Modelle, wie die Ausführung in einem Monitor nach dem Aufruf von `signal()` fortfährt (A: signalisierender Prozess, B: aufgeweckter Prozeß):

- Signal-and-wait: A muß nach seiner Signalisierung den Monitor sofort freigeben und warten, bis B den Monitor verlassen hat oder auf eine andere Bedingung wartet.
- Signal-and-continue: B muß warten, bis A den Monitor verlassen hat oder auf eine andere Bedingung wartet.

Welchem Modell folgt Java?