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 2019/2020
Übungsblatt 9

Abgabetermin: 23.12.2019, 18:00 Uhr

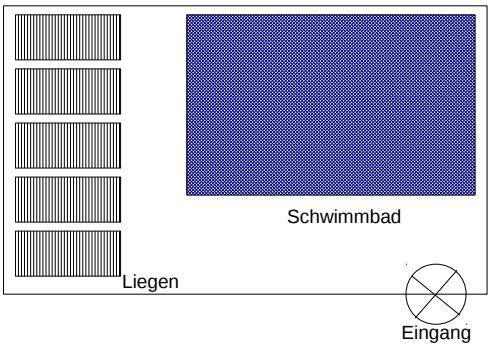
Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 16. – 20. Dezember 2019
Besprechung der H-Aufgaben in den Tutorien am 23. Dezember 2019 und vom 07. – 10. Januar 2020

Aufgabe 43: (T) 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 c).

(ii) Geben Sie in Analogie zum Pseudocode für Badegäste aus Aufgabe d) 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 d) so anpassen, dass erwachsene Badegäste bzw. Kinder (d.h. die ausführenden Prozesse der Funktionen `badegast()` und `kind()`) synchronisiert werden.

Aufgabe 44: (T) Semaphore und Prozessabläufe

(– Pkt.)

Wir betrachten im Folgenden die drei Prozesse P_1, P_2 und P_3, die ihren Zugriff auf gemeinsam genutze Ressourcen über die drei Zählsemaphore a, b und c synchronisieren. Für jeden der drei Prozesse zeigen die folgenden Programmsequenzen den Zugriff der Prozesse auf die entsprechenden Semaphore.

Prozess P_1

```
1  p1() {
2      ...
3      wait(a);
4      wait(a);
5      wait(a);
6      ...
7      signal(b);
8      signal(b);
9      ...
10 }
```

Prozess P_2

```
1  p2() {
2      ...
3      wait(c);
4      wait(c);
5      wait(c);
6      ...
7      signal(b);
8      ...
9  }
```

Prozess P_3

```
1  p3() {
2      ...
3      wait(b);
4      ...
5      signal(a);
6      signal(c);
7      ...
8  }
```

Ermitteln Sie für die vier folgenden Initialisierungen der Semaphore a, b und c, in welcher Reihenfolge die Prozesse P_1, P_2 und P_3 geschedult werden müssen, um möglichst viele dieser Prozesse vollständig abzuarbeiten und geben Sie die Bezeichner der vollständig abgearbeiteten Prozesse

an. Gehen Sie dabei von einem preemptiven Scheduling Algorithmus aus. Verwenden Sie folgende Darstellung zur Veranschaulichung der Abläufe, die in einem Deadlock bzw. einem blockierten Zustand enden oder zu einem erfolgreichen Durchlauf führen. Markieren Sie mögliche Deadlocks bzw. Blockierungen:

Prozess	Ausgeführte Codezeile
...	...

Hinweis: Eine Codezeile soll erst dann als ausgeführt angesehen werden, wenn der entsprechende Funktionsaufruf zurückgekehrt ist.

Initialisierung	i)	ii)	iii)	iv)
Semaphor a	2	2	1	3
Semaphor b	0	0	1	0
Semaphor c	2	3	2	0

Aufgabe 45: (H) Semaphore

(17 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 Gäste 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 konzentriert abarbeiten. Deshalb muss in Ihrer Modellierung sichergestellt werden, dass zu jeder Zeit nur ein Gast mit dem Barkeeper interagiert. 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.
Bearbeiten Sie nun auf Grundlage dieses Szenarios folgende Aufgaben:

a. Was sind die kritischen Bereiche bei diesem Problem?

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

Bezeichner	Zweck
`bestellwunsch`	Signalisiert dem Barkeeper einen Bestellwunsch
`getränk_fertig`	Signalisiert einem Gast die Fertigstellung seines Getränks

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:

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.

d. Vervollständigen Sie nun den Pseudocode für einen Gast (den Code-Rahmen dafür können Sie von der Website zur Vorlesung herunterladen), 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, was von Ihnen modelliert werden soll. Der Pseudocode des Barkeepers ist bereits gegeben (ebenfalls auf der Webseite zur Vorlesung). Verwenden Sie für den Zugriff auf Ihre Semaphore die gleiche Notation wie in Aufgabe 43).

Aufgabe 46: (H) Einfachauswahlaufgabe: Prozesskoordination

(5 Pkt.)

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

a) Wie bezeichnet man den Modus, bei dem Anwendungsprogramme ausgeführt werden und damit keine privilegierten Op- erationen abgearbeitet werden können?

(i) Private Mode	(ii) Public Mode	(iii) User Mode	(iv) Kernel Mode

b) Was entspricht der Zeitspanne, die ein Prozess im System verbringt, ohne dass er ausgeführt wird (nicht im Zustand „running“)?

(i) Wartezeit	(ii) Bedienzeit	(iii) Antwortzeit	(iv) Verweildauer

c) Angenommen das Erzeuger/Verbraucher-Problem wurde mit Hilfe von Semaphoren gelöst. Dabei regelt ein binärer Semaphor `s` den wechselseitigen Zugriff auf den gemeinsamen Speicher, ein Zählsemaphor `b` repräsentiert die Anzahl der belegten Plätze und ein weiterer Zählsemaphor `p` gibt die Anzahl der freien Speicherplätze (wovon `MAX > 0` zur Verfügung stehen) an. Die Semaphoren werden wie folgt initialisiert: `init(s, 1); init(b, 0); init(p, MAX).`

Welche Aussage bezüglich des folgenden Quellcodes für den Erzeuger und den Verbraucher, welche parallel ausgeführt werden, ist korrekt?

```
(* Erzeuger: *)
REPEAT
  <erzeuge Element>;
  wait(s);
  wait(p);
  <Element in Speicher legen>;
  signal(s);
  signal(b);
UNTIL FALSE;
```

```
(* Verbraucher: *)
REPEAT
  wait(b);
  wait(s);
  <El. aus Speicher nehmen>;
  signal(s);
  signal(p);
  <verbrauche Element>;
UNTIL FALSE;
```

(i) Die Lösung ist korrekt.	(ii) Der Verbraucher kann niemals ein Element aus dem Speicher nehmen.	(iii) Der Erzeuger kann niemals ein Element in den Speicher legen.	(iv) Es kann ein Deadlock entstehen.

d) 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

e) Welche Aussage bezüglich des klassischen Erzeuger/Verbraucher- Problems ist falsch? Dabei handelt es sich zum Beispiel um zwei Prozesse, die über eine gemeinsam genutzte Datenstruktur (gemeinsam genutzten Speicherbereich) Informationen austauschen (analog zu Aufgabenteil c)).

(i) Die Summe aus der Anzahl der verfügbaren Speicherplätze `p` und der Anzahl der belegten Speicherplätze (Bestand) `b` ergibt bei einer korrekten Implementierung des Erzeugers und Verbrauchers zu jedem Ausführungszeitpunkt den Wert `MAX`; (`p + b = MAX`)

(ii) Der Verbraucher kann nur so lange Objekte aus der Datenstruktur entnehmen, bis diese leer ist.

(iii) Erzeuger und Verbraucher dürfen nicht gleichzeitig auf die gemeinsam genutzt Datenstruktur (den gemeinsam genutzten Speicherbereich) zugreifen.

(iv) Der Erzeuger kann nur so viele Objekte in die Datenstruktur einfügen, wie der Speicherbereich Kapazität bietet.