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

mobile and distributed systems group

Tutoriumsblatt 9
Betriebssysteme im WiSe 2020/2021

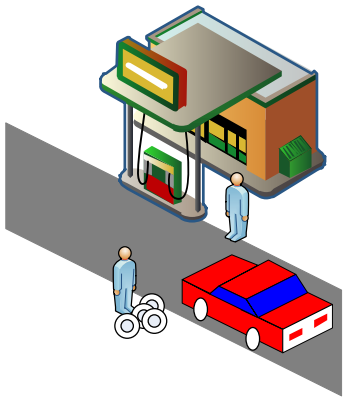
Zum Modul K

Tutorium: Die Aufgaben werden in einem Tutorien-Video vorgestellt, das am 13. Januar 2020 (17 Uhr) veröffentlicht wird.

Aufgabe T21: Semaphore

(– Pkt.)

In dieser Aufgabe sollen Sie das Konzept der Semaphore am Beispiel eines Boxenstopps während eines Autorennens umsetzen. Dabei fährt ein Rennwagen in die Boxengasse und hält an der Box seines Teams an. Dort befinden sich ein Tankwart und ein Mechaniker, der die Reifen des Wagens wechselt. Bevor der Wagen an der Box ankommt sind der Tankwart und der Mechaniker untätig (blockiert). Sobald der Wagen eintrifft erhalten sie Signale und werden tätig. Dann tankt der Tankwart den Wagen voll und der Mechaniker wechselt alle vier Reifen des Wagens. Während Tankwart und Mechaniker tätig sind, ist wiederum der Wagen untätig (blockiert). Er darf erst wieder abfahren, wenn Tankwart und Mechaniker jeweils ihre Tätigkeiten verrichtet haben, was sie dem Wagen signalisieren. Modellieren Sie den Signalisierungsprozess zwischen dem Rennwagen und dem Tankwart bzw. dem Mechaniker mit Hilfe von Semaphoren. In dieser Aufgabe wird nur die Box eines Teams modelliert. Allerdings kann dieses Team mehrere Wagen auf der Strecke haben, die um den Platz an der Box konkurrieren. Stellen Sie dementsprechend sicher, dass sich zu jedem Zeitpunkt nur ein Rennwagen an der Box befindet.
Die nachfolgende Abbildung zeigt den schematischen Aufbau.



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 `tanken` und `abfahrt` sind bereits gegeben.

Bezeichner	Zweck
`tanken`	Signalisiert dem Tankwart, dass der Wagen eingefahren ist und betankt werden muss
`abfahren`	Signalisiert, dass ein Arbeiter mit seiner Tätigkeit fertig ist
hspace*20px:	

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(tanken, 0);`	Initialisiert den Semaphor `tanken` mit dem Wert 0.
`init(abfahrt, 0);`	Initialisiert den Semaphor `abfahrt` mit dem Wert 0.
hspace*20px:	

d. Vervollständigen Sie nun den folgenden Pseudocode für einen Rennwagen, so dass auch mehrere Rennwagen, die um den Platz an der Box konkurrieren, stets synchronisiert werden. Vervollständigen Sie zudem den Pseudocode für den Mechaniker, der erst tätig werden soll, wenn ein Rennwagen eingefahren ist, dessen vier Reifen wechselt und dies dem Rennwagen signalisiert. Der Pseudocode des Tankwarts ist bereits gegeben. Der Rennwagen darf erst wieder abfahren, wenn beide Arbeiter ihre Tätigkeiten verrichtet haben.

```
1  Tankwart() {
2        while(true) {
3              //auf Tankaufforderung warten
4              wait(tanken);
5              <tanken>
6              //Fertigstellung signalisieren
7              signal(abfahrt);
8        }
9  }
```

```
1  Mechaniker() {
2        while(true) {
3
4
5              //auf Signal zum Wechseln der Reifen warten
6
7
8
9
10             <Reifen wechseln>
11
12
13
14
15
16            //Fertigstellung signalisieren
17
18
19
20       }
21 }
```

```
Rennwagen() {
     while(true) {


             <in die Boxengasse bzw. an die Box fahren>;


             //tanken anfordern




             //Reifenwechsel anfordern




             for(int Arbeiter = 0; Arbeiter < 2; Arbeiter++) {


                  //auf Signal des jeweiligen Arbeiters warten


             }


             <die Box verlassen>;


       }
}
```

Verwenden Sie für den Zugriff auf Ihre Semaphore folgende Notation:

Pseudocode	Beispiel	Bedeutung
`wait(<semaphor>);`	`wait(mutex);`	Verringert den Wert des Semaphor `mutex` um eins
`signal(<semaphor>);`	`signal(mutex);`	Erhöht den Wert des Semaphor `mutex` um eins
`<Aktion>`	`<die Box verlassen>`	Führe die gelistete Aktion aus
`//Kommentar`	`//Fertigstellung signalisieren`	Kommentarzeile

Aufgabe T22: 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