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

mobile and distributed systems group

Tutoriumsblatt 8
Betriebssysteme im WiSe 21/22

Zum Modul I

Besprechung: Die Tutoriumsaufgaben werden im Tutoriumsvideo 8 besprochen. Alle Tutorenvideos sind auf LMUCast zum Abruf verfügbar.

Aufgabe T18.1: Betriebsmittelzugriffe

(– Pkt.)

Eine wesentliche Aufgaben des Betriebssystems ist die Verwaltung von Betriebsmitteln bzw. der Zugriffe auf diese.

a. Welche Arten von Prozessabläufen gibt es und welche davon erfordern eine Synchronisation der Prozesse?

b. Was versteht man unter einem kritischen Bereich und welche Anforderungen werden an einen Lösung zum wechselseitigen Ausschluss gestellt?

c. Nennen Sie ein Beispiel für ein unteilbares Betriebsmittel. Gibt es Fälle in dem solche Betriebsmittel trotzdem parallel genutzt werden können?

Aufgabe T18.2: Wechselseitiger Ausschluss

(– Pkt.)

Von einer **Race Condition** spricht man, wenn das Ergebnis nebenläufiger Prozesse von der Reihenfolge der Prozessaktivierung abhängt.

Im Folgenden seien die beiden Prozesse *Lagerist* und *Lageristin* gegeben. Die beiden Kollegen arbeiten im Lager der gleichen Firma. Die Lageristin will eine neu eingetroffene Lieferung von 60 Kisten im Lager verbuchen. Zum gleichen Zeitpunkt will der Lagerist 20 Kisten an einen Geschäftspartner verkaufen. Der digitale Bestand `k` sei eine globale Variable, die von beiden Prozessen verändert werden kann. Zu Beginn sei der Bestand `k` bei 40 Kisten. Die Variable `b` stellt eine lokale Variable dar, von der jeder Prozess seine eigene Instanz besitzt.

Prozess **Lagerist**

```
1 ...
2 b = k;
3 b = b - 20;
4 k = b;
5 ...
```

Prozess **Lageristin**

```
1 ...
2 b = k;
3 b = b + 60;
4 k = b;
5 ...
```

a. Geben Sie eine zeitliche Abfolge der Prozesse Lageristin und Lagerist an, so dass eine Race Condition eintritt und der Bestand `k` am Ende beider Verbuchungen inkonsistent wird. Verwenden Sie dazu folgende Darstellung:

aktiver Prozess	ausgeführte Codezeile	Inhalt von `k`	Inhalt von `b` (Lagerist)	Inhalt von `b` (Lageristin)	Kommentar
...	...	...	...	...	...

b. Nennen Sie die drei Bedingungen, die erfüllt sein müssen, um Race Conditions zu verhindern.

c. Betrachten Sie nun die folgenden Vorschläge, das Problem zu lösen:

(i) Der Prozess Lagerist setzt die globale boolesche Variable `x` auf `true`, bevor er den kritischen Bereich betritt. Vor dem Setzen von `x` wird geprüft, ob die Variable nicht bereits vom Prozess Lageristin auf `true` gesetzt wurde. Ist dies der Fall, wird kein Eintritt in den kritischen Bereich gewährt. Der Prozess Lageristin verwendet die globale boolesche Variable `y` und geht analog vor.

Prozess **Lagerist**

```
1 ...
2 x = false;
3 while(y) { /* do nothing */ }
4 x = true;
5 b = k;
6 b = b - 20;
7 k = b;
8 x = false;
9 ...
```

Prozess **Lageristin**

```
1 ...
2 y = false;
3 while(x) { /* do nothing */ }
4 y = true;
5 b = k;
6 b = b + 60;
7 k = b;
8 y = false;
9 ...
```

Zeigen Sie an einem Beispielablauf, dass dieses Vorgehen das Auftreten einer Race Condition nicht verhindern kann. Warum funktioniert dieser Ansatz nicht? Welche der drei Bedingungen wird verletzt?

(ii) Um das im ersten Ansatz auftretende Problem zu beheben, verwenden wir nun zusätzlich eine Integervariable:

Prozess **Lagerist**

```
1 ...
2 x = true;
3 z = 0;
4 while(z == 1 || y) {
5     /* do nothing */
6 }
7 b = k;
8 b = b - 20;
9 k = b;
10 x = false;
11 ...
```

Prozess **Lageristin**

```
1 ...
2 y = true;
3 z = 1;
4 while(z == 0 || x) {
5     /* do nothing */
6 }
7 b = k;
8 b = b + 60;
9 k = b;
10 y = false;
11 ...
```

Zeigen Sie wieder an einem Beispielablauf, dass auch dieses Vorgehen das Problem nicht behebt. Welche der drei Bedingungen wird hier verletzt?