ALP4 SoSe 2013, Di. 16-18

Lösung Übungsblatt 8

Christoph van Heteren-Frese (Matr.-Nr.: 4465677)
Sven Wildermann (Matr.-Nr.: 4567553)
Tutor: Alexander Steen, eingereicht am 14. Juni 2013

Aufgabe 1

a)

Das Paket `rwmutex.go` implementiert ein spezielles Schloss, das zwar viele Prozesse lesen, aber nur einen schreiben lässt. Es werden dafür vier Zugriffsfunktionen definiert: `RLock()`, `RUnlock()`, `Lock()` und `Unlock()`. Will ein Prozess den Mutex für den Schreibzugriff nutzen obwohl gerade andere Prozesse lesen, wird dieser blockiert. Grundlage der Erläuterung ist folgendes kleines Beispiel:

```
1  package main
2
3  import (
4    "fmt"
5    "sync"
6    "time"
7  )
8
9  var (
10   rwm     sync.RWMutex
11   balance int
12 )
13
14 // "reads" the current balance and prints it
15 // to the sdtout
16 func get(balance *int) {
17   rwm.RLock()
18   fmt.Println(*balance)
19   rwm.RUnlock()
20 }
21
22 // "writes" the current balance: increases the
23 // balance by the given amount
24 func put(balance *int, amount int) {
25   rwm.Lock()
26   *balance += amount
27   rwm.Unlock()
28 }
29
30 func main() {
31   balance = 0
32   go get(&balance)
33   go put(&balance, 100)
34   go get(&balance)
35   go put(&balance, 100)
36   go get(&balance)
37   time.Sleep(1000)
38 }
```

b)

c)

Aufgabe 2

Aufgabe 3

Algorithmus in Pseudocode.

```
1   // Variabeldeklarationen
2   type objects struct {
3       Enum groesse = {gross, mittel, klein}
4       bool richtung
5       time timestamp_incoming}
6   mainqueue (queue)objects
7   leftqueue (queue)objects
8   rightqueue (queue)objects
9   islocked int
10
11  // Algorithmus
12  func add_object(new objects){
13       // 0 = von links nach rechts
14       // 1 = von rechts nach links
15      if new.richtung=0{
16       leftqueue.add(new)
17       }else{
18       rightqueue.add(new)
19       }
20       if mainqueue.isEmpty{
21        // wenn es das erste Element ist,
22        // wird die Aktualisierung angestossen
23              mainqueue.add(new)
24              objekte_aktualisieren()
25       }else{
26       //sonst nicht
27              mainqueue.add(new)
28       }
29    }
30  func objekte_aktualisieren(){
31       current objects;
32       // sortiert die Elemente nach Einf gezeit
33       mainqueue.sort(timestamp)
34       leftqueue.sort(timestamp)
35       rightqueue.sort(timestamp)
36       //  berpruefen   ob der Weg frei ist und ob ein
37       // Objekt den Weg passieren will
38       if islocked == 0 && (not mainqueue.isEmpty){
39       current = mainqueue.get
40       if current.richtung==0{
41       // pr fe nach der Reihe ob es in lefftqueue
42       // weitere Elemente gibt, die mit current
43       // auf den Weg gehen k nnen.
44       // Sende diese inkl. current los und l sche Sie aus
45       // den queues, erh he jeweils islocked um die Anzahl
46       // der losgeschickten Elemente
47       }else if current.richtung==1{
48       // pr fe nach der Reihe ob es in rightqueue
49       // weitere Elemente gibt, die mit current
50       // auf den Weg gehen k nnen.
```

```
51        // Sende diese inkl. current los und l sche Sie aus
52        // den queues, erh he jeweils islocked um die Anzahl
53        // der losgeschickten Elemente
54             }
55
56
57        }
58
59   }
60
61   func objekt_ist_angekommen(){
62        // wird vom Objekt aufgerufen
63        // sobald ein Objekt angekommen ist
64        islocked.lock();
65        islocked--;
66        islocked.unlock();
67        objekte_aktualisieren();
68   }
```

Dadurch dass die Queues nach der Einfügezeit der Objekte sortiert werden führt auch ein nebenläufiges einfügen zur korrekten Reihenfolge der Elemente in der Queue. Sobald entweder ein Objekt angekommen ist oder ein Objekt in die leere haupt-queue eingefügt wird, wird "objekte-aktualisieren" ausgeführt. Diese Funktion überprüft erst, welches das am längsten wartende Objekt ist und dann welche Elemente aus der Queue der selben Richtung mit diesem Element möglicherweise zusammen losgeschickt werden könnten. Diese werden dann alle zusammen losgelassen (Barriere wird geöffnet). Um einen Überblick über die Anzahl der zur Zeit reisenden Objekte zu erhalten, wird der Counter islocked um den entsprechenden Wert erhöht. Sobald die einzelnen Elemente am Ziel angekommen sind, führen sie objekt-ist-angekommen() aus, der den Counter schrittweise wieder verringert. So wird in der Funktion objekte-aktualisieren sicher gestellt, dass keine neuen Objekte frei gelassen werden wenn noch nicht alle Objekte wieder angekommen sind.

Aufgabe 4