

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 stdout
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

a)

Modellierung siehe Aufgabenteil b). Begründung der Korrektheit: Für jedes Gleis an einem Bahnhof gibt es als Semaphore so genannte “Wächter”. Diese geben initial die Gleise frei. Sobald nun ein Zug das Gleis befahren möchte, wird ein Wächter für diesen Vorgang reserviert und das Gleis von diesem Wächter im Anschluss gesperrt. Nachdem der Zug dann durchgefahren ist, wird erst das Gleis durch den Wächter wieder freigegeben und anschließend der Wächter wieder für andere Vorgänge verfügbar. Die Zugriffe auf die Variablen sind durch mutex geschützt.

b)

```
1  /* bahnhof.go */
2  package main
3
4  import (
5      "sync"
6      "fmt"
7      "time"
8  )
9
10 type Imp struct {
11     free []int
12     val int
13     cs, mutex Mutex
14 }
15
16 type Bahnhof struct {
17     guard *Imp
18     gleis []Mutex
19 }
20
21 func NewImp(n int) *Imp {
22     x := new(Imp)
23     x.val = n
24     x.free = make([]int, n)
25     for i := 0; i < n; i++ {
26         x.free[i] = i
27     }
28     if n == 0 {
29         x.cs.Lock()
30     }
31     return x
32 }
33
34 func (x *Imp) P() (i int) {
35     x.cs.Lock()
36     x.mutex.Lock()
37     x.val--
38     if x.val > 0 {
```

```

39     x.cs.Unlock()
40 }
41 i = x.free[i]
42 x.free = x.free[1:]
43 x.mutex.Unlock()
44 return i
45 }
46
47 func (x *Imp) V(i int) {
48     x.mutex.Lock()
49     x.val++
50     if x.val == 1 {
51         x.cs.Unlock()
52     }
53     x.free = append(x.free, i)
54     x.mutex.Unlock()
55 }
56
57 func NewBahnhof(n int) *Bahnhof {
58     b := new(Bahnhof)
59     b.guard = NewImp(n)
60     b.gleis = make([]Mutex, n)
61     return b
62 }
63
64 func (b *Bahnhof) durchfahren(e chan bool) {
65     i := b.guard.P()
66     b.gleis[i].Lock()
67     fmt.Printf("ich fahre hier auf Gleis %d!\n", i)
68     fmt.Printf("Ich lasse mir jetzt 10 Sekunden Zeit..\n")
69     x := time.Duration(10)
70     time.Sleep(x * time.Second)
71     b.gleis[i].Unlock()
72     b.guard.V(i)
73     e <- true
74 }
75
76 func main() {
77     done1 := make(chan bool, 1)
78     done2 := make(chan bool, 1)
79     done3 := make(chan bool, 1)
80     done4 := make(chan bool, 1)
81     b := NewBahnhof(2)
82     go b.durchfahren(done1)
83     go b.durchfahren(done2)
84     go b.durchfahren(done3)
85     go b.durchfahren(done4)
86     <- done1
87     <- done2
88     <- done3
89     <- done4
90 }

```

c)

Ja, bei der Implementierung kann es zu Kollisionen zweier Züge an zusammenführenden Weichen kommen, da keine Weichen sondern nur vollständig getrennte Gleise gesichert werden. Durch weitere Anpassungen (zusätzliche Wächter für Weichen) könnte dies jedoch verhindert werden.

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 leftqueue
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

Die Implementierung in GO compiliert leider nicht erfolgreich. In Anbetracht einer pünktlichen Abgabe habe ich es nicht mehr geschafft, dies noch zu fixen.

```

1  package main
2  import "fmt"
3
4  /* Leider habe ich es nicht puenktlich
5  geschafft den Code erfolgreich zum compilieren zu bringen*/
6
7
8
9  func mult(matrix1 [][]int, matrix2 [][]int, e chan bool){
10
11      newMa := [][]int{{0,0,0},{0,0,0},{0,0,0}}
12      for i := 0 ; i < len(matrix1) ;i++){
13          w := make (chan bool)
14              for j := 0 ; j < len(matrix2[0]) ;j++ {
15                  go rowCol(i,j,matrix1,matrix2,newMa,w)
16              }
17          }
18
19      }
20      fmt.Println(newMa)
21      e<-true
22  }
23
24
25  func rowCol( r int , c int, matrix1 [][]int, matrix2 [][]int, newMa [][]int, e chan bool){

```

```

26     var sum int = 0
27     for i := 0 ; i < len(matrix1) ; i++ {
28         sum += matrix1 [r][i] * matrix1 [i][c]
29     }
30     newMa[r][c] = sum
31     e<-true
32
33 }
34
35 func main(){
36     matrix1 := [][]int { {1,2,3} ,
37                           {4,5,6} ,
38                           {7,8,9} }
39     matrix2 := [][]int {{0,0,1} ,
40                         {0,1,0} ,
41                         {1,0,0} }
42
43     q := make (chan bool)
44
45     mult(matrix1 , matrix2, q)
46
47     <-q
48
49
50 }

```