12 Errata

12.1 *Einführung*

S. 13, Zeile 2:
`import ("math/rand"; . "time"; "fmt")`

S. 13, Zeile 4:
`func v() { Sleep(Duration(rand.Int63n(1e5))) }`

S. 14, Zeile 7:
`count` einfach nur ...

S. 23, Zeile 12:
```
    go f()
```

S. 26, Zeile 3 von unten:
Anweisung `x = <-c` die Zuweisung ...

12.2 *Schlösser*

S. 30, Zeile 13 von unten:
`Sichert den Zugang zu einem kritischen Abschnitt.`

S. 33, Zeile 18:
```
    Counter = Accu // "STA Counter"
```

S. 41, Zeile 3:
`func Lock(p uint) { // p < 2`

S. 43, Zeile 3:
i_p für `interested[p] == true`

S. 43, Zeile 8 von unten:
wegen $\neg i_1$ möglich, weil ...

S. 45, Zeile 12:
```
    if favoured == p { goto L }
```

S. 51, Zeile 10 von unten:
`number[p] = max() + 1`

S. 52, Zeile 6:
```
  for a:= uint(0); a < P; a++ {
```

12.3 *Semaphore*

S. 61, Zeile 12:
`notEmpty.V()`

S. 61, Zeile 12 von unten:
Für `in != out` ist das ...

S. 64, Zeile 14 von unten:
`// Die Funktionen Insert und Get sind atomar, ...`

S. 63 bis 65:
Die Programmtexte der Spezifikation im sequentiellen Fall (S. 65, Zeile 3-24) und der Implementierung im nebenläufigen Fall (S. 63, Zeile 17, bis S. 64, Zeile 9) sind vertauscht.

S. 69, Zeile 14 von unten:
`if n == -1 {`

S. 65, Zeile 11 von unten:
`Liefert das erste Element von x.`

S. 71, Zeile 7 von unten:
kritischer Abschnitt ist frei,

S. 71, Zeile 3-2 von unten:
... des durch `x.mutex` geschützten Zugriffs auf `x` ...

S. 72, Zeile 10-11:
```
if x.val <= 0 { x.cs.Unlock()
  } else { ...
```

S. 74, Zeile 20 ff.:
```
x.n2++
x.n1--
if x.n1 > 0 {
 ...
```

S. 78, Zeile 12:
`if ! free { ...`

S. 79, Zeile 7, 10, 14, 17:
`mutexLR`

S. 83, Zeile 14 von unten:

```
  e.Unlock()
```

S. 86, Zeile 15 von unten:

```
type ( Any interface{}
```

S. 87, Zeile 17 von unten:

```
// Vor.:  n > 1.  c, i und 0 sind für alle k < n definiert.
```

S. 90, Zeile 4, 11:

```
func (x *Imp) Enter (k uint, a Any) {
  x.in(a, k)
```

S. 90, Zeile 14, 16:

```
func (x *Imp) Leave (k uint, a Any) {
  x.out(a, k)
```

S. 91, Zeile 2:

```
import .  "cs"
```

S. 91, Zeile 9:

```
func WriterOut() { x.Leave(w, nil) }
```

S. 96, Zeile 8:

```
  fork[p].Lock(); fork[left(p)].Lock()
```

S. 96, Zeile 10-11:

```
func Unlock (p uint) {
  fork[p].Unlock(); fork[left(p)].Unlock()
```

S. 97, Zeile 14 von unten:

```
  status[p] = dining
```

S. 97, Zeile 10-8 von unten:

```
  status[p] = satisfied
  test(left(p))
  test(right(p))
```

S. 99, Zeile 9 von unten:

... mit einem auf diese Maximalzahl `maxR` initialisierten ...

S. 100, Zeile 15 von unten:

```
type Imp struct {
```

S. 100, Zeile 11 von unten:

```
            nB [M]int
```

S. 101, Zeile 19-21:

```
    for x.nB[i] > 0 && i <= x.val {
      x.val -= i
      x.nB[i]--
```

S. 101, Zeile 8-6 von unten:

```
func (x *Imp) V (n uint) {
  ...
  i:= uint(1)
```

S. 102, Zeile 6 von unten:
`n` zu Protokollzwecken und ...

12.4 *Fairness*

S. 108, Zeile 7:

```
import (. "time"; "math/rand")
```

S. 108, Zeile 16:

```
  Sleep(Duration(rand.Int63n(pause)))
```

S. 109, Zeile 2:
im Prozess `stop`

S. 109, Zeile 4:

```
halt = true
```

S. 109, Zeile 5:

```
falsify
```

12.5 *Verklemmungen*

S. 114, Zeile 14 von unten:
von ihnen angeforderten Betriebsmittel.

S. 118, Zeile 15:

```
          ... + cash == capital
```

S. 121, Zeile 15 von unten:
... herausstellen.

12.6 *Monitore*

S. 127, Zeile 16:
`func Inc (n uint)` {

S. 127, Zeile 19:
`func Value () uint` {

S. 143, Zeile 21-22:
... nach einem klaren und selbstdokumentierenden ...

S. 143, Zeile 5 von unten:
`b.Wait()`

S. 143, Zeile 3 von unten:
... { `b.Signal()` } }

S. 146, Zeile 14 von unten:
`barberFree = true`

S. 147, Zeile 16:
`Liefert das Minimum ...`

S. 148, Zeile 10 von unten:
Ein Prozess, der die ...

S. 150, Zeile 2 von unten:
`c.nB = 0`

S. 151, Zeile 9:
Wenn `Signal` nur als letzte Anweisung vorkommt, ...

S. 152, Zeile 1:
`func (c *Condition) Signal()` {

S. 152, Zeile 10 von unten:
`urgent.V()`

S. 152, Zeile 8 von unten:
`me.Unlock()`

S. 154, Zeile 17:
Aufruf von `Wait` durch einen anderen Prozess.

S. 155, Zeile 6:
`package mon`

S. 161, Zeile 7 von unten:
... in der Schleife innerhalb der Funktion `Func` –

S. 161, Zeile 1 von unten:
`vall` bei den ...

S. 162, Zeile 21 von unten:
```
var m mon.Monitor
```

S. 163, Zeile 12 von unten:
```
m = mon.New (5, f)
```

12.7 *Botschaftenaustausch*

S. 168, Zeile 10 von unten:
die Anweisung zum *Senden* von `a` auf `c`.

S. 169, Zeile 3 von unten:
```
import ("fmt"; . "time"; "math/rand")
```

S. 170, Zeile 3:
```
    Sleep(Duration(rand.Int63n(1e9))) // BesucherIn durch Tor
```

S. 171, Zeile 2 von unten:
```
func (x *Imp) Insert (b byte) { x.c <- b }
```

S. 173, Zeile 27:
```
go send (cryptchan, done) // Bote
```

S. 175, Zeile 2:
`w` auskommen.

S. 175, Zeile 10:
```
import (. "fmt"; "math/rand")
```

S. 177, Zeile 17 von unten:
... , weil hier Leser sowohl ein- als auch austreten können, ...

S. 177, Zeile 15 von unten:
... , noch im Fall `nR == 0`, weil ...

S. 182, Zeile 4:
... eines aus dem Puffer entnommenen ...

S. 185, Zeile 10:
`case buffer[in] = When ...`

S. 185, Zeile 12:
`case When (count > 0, cG) ...`

S. 186, Zeile 14 von unten:
Am Anfang der Funktion fehlt die Variablendeklaration
```
  var b byte
```

S. 189, Zeile 5 von unten bis Ende:
```
  if x.first == nil {
    x.first = n
  } else {
    x.last.next = n
  }
  x.last = n
}
```

12.8 *Netzweiter Botschaftenaustausch*

S. 194, Zeile 10-11:
Die Funktionen `Accept` (nach `ListenTCP`) und

S. 194, Zeile 4, 7 und 8 von unten:
`Port0+p`

S. 195, Zeile 14:
`import ( ... ; . "coder")`

S. 196, Zeile 17:
... müssen wir die Existenz eines Pakets

S. 197, Zeile 21 von unten:
zugreifen wollen, ...

S. 205, Zeile 5:
Im Server-Fall liefert die Funktion `Accept` jetzt nicht mehr ...