

# Distributed Communication 7th practice

Li Jianhao  
lijianhao288@hotmail.com

## 1 Basics

### 1.1 Example without goroutine number limit

package: "runtime"  
func NumGoroutine() int  
return the number of goroutines that currently exist.

package: "sync/atomic"  
func LoadUint64(addr \*uint64) (val uint64)  
Get the value of the uint64 atomically.  
func StoreUint64(addr \*uint64, val uint64)  
Store the value of the uint64 atomically.

package: "math/rand"  
func Intn(n int) int  
 $[0, n)$

```
package main                                     1
import (                                          2
    "fmt"                                       3
    "math/rand"                                4
    "runtime"                                  5
    "sync"                                      6
    "sync/atomic"                              7
    "time"                                      8
)                                                9
                                                10
var jobQueue = make(chan string, 100)          11
var maxGo uint64                               12
var wg sync.WaitGroup                          13
                                                14
func main() {                                   15
    go goroutineCounter()                      16
                                                17
                                                18
```

```

    start := time.Now()
    wg.Add(1)
    go linkSender()

    wg.Add(1)
    go workerCreator()

    wg.Wait()

    fmt.Println("Max goroutine number:", atomic.LoadUint64(&maxGo))
    duration := time.Since(start)
    fmt.Println("Time:", duration)
}
func goroutineCounter() {
    for {
        n := runtime.NumGoroutine()
        u := uint64(n)
        if u > maxGo {
            atomic.StoreUint64(&maxGo, u)
        }
        time.Sleep(50 * time.Millisecond)
    }
}
func linkSender() {
    defer wg.Done()
    links := []string{}
    var numOfLink = 1000
    for i := 0; i < numOfLink; i++ {
        fakeLink := fmt.Sprintf("http://web%d.com", i)
        links = append(links, fakeLink)
    }
    for _, link := range links {
        jobQueue <- link
    }
    close(jobQueue)
}
func workerCreator() {
    defer wg.Done()
    for link := range jobQueue {
        wg.Add(1)
        go worker(link)
    }
}
func worker(l string) {
    defer wg.Done()
    fmt.Println(linkTest(l))
}
func linkTest(link string) string {
    time.Sleep(500 * time.Millisecond)
    if rand.Intn(2) == 1 {
        return link + ": Good"
    } else {
        return link + ": Bad"
    }
}

```

Listing 1: Without limit

...	1
http://web395.com: Good	2
http://web392.com: Good	3
http://web400.com: Bad	4
http://web397.com: Bad	5
http://web396.com: Bad	6
http://web401.com: Bad	7
http://web402.com: Bad	8
http://web394.com: Bad	9
http://web399.com: Bad	10
Max goroutine number: 1002	11
Time: 507.788444ms	12

## 1.2 Limit the number of goroutines

package main	1
	2
import (	3
"fmt"	4
"math/rand"	5
"runtime"	6
"sync"	7
"sync/atomic"	8
"time"	9
)	10
	11
var workerPool = make(chan int, 50)	12
var jobQueue = make(chan string, 100)	13
var maxGo uint64	14
var wg sync.WaitGroup	15
	16
func main() {	17
go goroutineCounter()	18
	19
start := time.Now()	20
wg.Add(1)	21
go linkSender()	22
	23
wg.Add(1)	24
go workerCreator()	25
	26
wg.Wait()	27
	28
fmt.Println("Max_goroutine_number:", atomic.LoadUint64(&maxGo))	29
duration := time.Since(start)	30
fmt.Println("Time:", duration)	31
}	32
	33
func goroutineCounter() {	34
for {	35
n := runtime.NumGoroutine()	36
u := uint64(n)	37
if u > maxGo {	38
atomic.StoreUint64(&maxGo, u)	39
}	40
time.Sleep(200 * time.Millisecond)	41
}	42

}	43
func linkSender() {	44
defer wg.Done()	45
links := []string{}	46
var numOfLink = 1000	47
for i := 0; i < numOfLink; i++ {	48
fakeLink := fmt.Sprintf("http://web%d.com", i)	49
links = append(links, fakeLink)	50
}	51
for _, link := range links {	52
jobQueue <- link	53
}	54
close(jobQueue)	55
}	56
	57
func workerCreator() {	58
defer wg.Done()	59
for link := range jobQueue {	60
workerPool <- 1	61
wg.Add(1)	62
go worker(link)	63
}	64
}	65
	66
func worker(link string) {	67
defer wg.Done()	68
defer func() { <-workerPool }()	69
fmt.Println(linkTest(link))	70
}	71
	72
func linkTest(link string) string {	73
time.Sleep(500 * time.Millisecond)	74
if rand.Intn(2) == 1 {	75
return link + ":_Good"	76
} else {	77
return link + ":_Bad"	78
}	79
}	80
}	81

Listing 2: With limit

...	1
http://web989.com: Bad	2
http://web976.com: Bad	3
http://web991.com: Bad	4
http://web999.com: Good	5
http://web997.com: Bad	6
http://web955.com: Bad	7
http://web956.com: Bad	8
Max goroutine number: 54	9
Time: 10.043942287s	10