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

import (
    "fmt"
    "math/rand"
    "runtime"
    "sync"
    "sync/atomic"
    "time"
)

var jobQueue = make(chan string, 100)
var maxGo uint64
var wg sync.WaitGroup

func main() {
    go goroutineCounter()

    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

```
...
http://web395.com: Good
http://web392.com: Good
http://web400.com: Bad
http://web397.com: Bad
http://web396.com: Bad
http://web401.com: Bad
http://web402.com: Bad
http://web394.com: Bad
http://web399.com: Bad
Max goroutine number:  1002
Time:  507.788444ms
```

The version with select:

```
package main

import (
    "fmt"
    "math/rand"
    "runtime"
    "sync"
    "sync/atomic"
    "time"
)

var jobQueue = make(chan string, 100)
var maxGo uint64
var wg sync.WaitGroup
var stopper = make(chan int)

func main() {
    go goroutineCounter()

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

    wg.Add(1)
    go workerCreator()

    wg.Wait()
    stopper <- 0
    fmt.Println("Max goroutine number: ", maxGo)
    duration := time.Since(start)
    fmt.Println("Time: ", duration)
}
func goroutineCounter() {
    for {
        select {
            case <- stopper:
                fmt.Println("goroutineCounter stop")
                return
            default:
                n := runtime.NumGoroutine()
                u := uint64(n)
                if u > maxGo {
                    atomic.StoreUint64(&maxGo, u)
                }
                time.Sleep(50 * time.Millisecond)
        }
```

```go
    }
}

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 2: Without limit (Select)

```
...
http://web919.com: Good
http://web923.com: Good
http://web931.com: Bad
http://web966.com: Bad
http://web977.com: Bad
http://web927.com: Bad
http://web946.com: Bad
http://web981.com: Bad
http://web929.com: Bad
goroutineCounter stop
Max goroutine number:  1002
Time:  885.9704ms
```

1.2 Limit the number of goroutines

```go
package main

import (
    "fmt"
    "math/rand"
    "runtime"
    "sync"
    "sync/atomic"
    "time"
)

var workerPool = make(chan int, 50)
var jobQueue = make(chan string, 100)
var maxGo uint64
var wg sync.WaitGroup

func main() {
    go goroutineCounter()

    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(200 * 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 {
        workerPool <- 1
        wg.Add(1)
        go worker(link)
    }
}

func worker(link string) {
    defer wg.Done()
    defer func() { <-workerPool }()
    fmt.Println(linkTest(link))
}

func linkTest(link string) string {
    time.Sleep(500 * time.Millisecond)
    if rand.Intn(2) == 1 {
        return link + ":␣Good"
    } else {
        return link + ":␣Bad"
    }
}
```

Listing 3: With limit

```
...
http://web989.com: Bad
http://web976.com: Bad
http://web991.com: Bad
http://web999.com: Good
http://web997.com: Bad
http://web955.com: Bad
http://web956.com: Bad
Max goroutine number:  54
Time:  10.043942287s
```