



Go Concurrency Part 2

Jin Hyun Kim
Fall, 2019

Go Concurrency

- select + channel
- channel

Channel Sync

```
package main

import "fmt"

func main() {
    done := make(chan bool)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(i)
        }
        done <- true
    }()

    // 위의 Go루틴이 끝날 때까지 대기
    <-done
}
```

<https://brownbears.tistory.com/315>

select

- Select a channel communication out of multiple channels simultaneously running
 - select문은 복수 채널들을 기다리면서 준비된 (데이터를 보내온) 채널을 실행하는 기능을 제공
 - select문은 여러 개의 case문에서 각각 다른 채널을 기다리다가 준비가 된 채널 case를 실행
 - select문은 case 채널들이 준비되지 않으면 계속 대기하게 되고, 가장 먼저 도착한 채널의 case를 실행
 - 만약 복수 채널에 신호가 오면, Go 런타임이 랜덤하게 그 중 한 개를 선택
 - select문에 default 문이 있으면, case문 채널이 준비되지 않더라도 계속 대기하지 않고 바로 default문을 실행

select

```
1 package main
2
3 import "time"
4
5 func main() {
6     done1 := make(chan bool)
7     done2 := make(chan bool)
8
9     go run1(done1)
10    go run2(done2)
11
12    EXIT:
13        for {
14            select {
15                case <-done1:
16                    println("run1 완료")
17
18                case <-done2:
19                    println("run2 완료")
20                    break EXIT
21            }
22        }
23    }
24
25    func run1(done chan bool) {
26        time.Sleep(1 * time.Second)
27        done <- true
28    }
29
30    func run2(done chan bool) {
31        time.Sleep(2 * time.Second)
32        done <- true
33    }
```

<http://golang.site/go/article>

- 첫번째 run1()이 1초간 실행되고 done1 채널로부터 수신하여 해당 case를 실행
- 다시 for 루프를 돈다. for루프를 다시 돌면서 다시 select문이 실행되는데, 다음 run2()가 2초후에 실행되고 done2 채널로부터 수신하여 해당 case를 실행
- done2 채널 case문에 break EXIT 이 있는데, 이 문장으로 인해 for 루프를 빠져나와 EXIT 레이블로 이동

select

```
1 package main
2
3 import "time"
4
5 func main() {
6     done1 := make(chan bool)
7     done2 := make(chan bool)
8
9     go run1(done1)
10    go run2(done2)
11
12    EXIT:
13    for {
14        select {
15            case <-done1:
16                println("run1 완료")
17
18            case <-done2:
19                println("run2 완료")
20                break EXIT
21        }
22    }
23 }
24
25 func run1(done chan bool) {
26     time.Sleep(1 * time.Second)
27     done <- true
28 }
29
30 func run2(done chan bool) {
31     time.Sleep(2 * time.Second)
32     done <- true
33 }
```

<http://golang.site/go/article>

default

fmt.Println("default") }

- 위의 문장을 마지막 case 뒤에 삽입
- 실험

```

package main

import (
    "fmt"
    "time"
)

func process(ch chan string) {
    time.Sleep(10 * time.Second)
    ch <- "process successful"
}

func scheduling(){
    //do something
}

func main() {
    ch := make(chan string)
    go process(ch)
    for {
        time.Sleep(1 * time.Second)
        select {
        case v := <-ch:
            fmt.Println("received value: ", v)
            return
        default:
            fmt.Println("no value received")
        }

        scheduling()
    }
}

```

- select: Producer and Consumer

```

package main

import (
    "fmt"
    "time"
)

func consuming (scheduler chan string){
    select {
    case <- scheduler:
        fmt.Println("이름을 입력받았습니다.")
    case <-time.After(5 * time.Second):
        fmt.Println("시간이 지났습니다.")
    }
}

func producing(scheduler chan string){
    var name string
    fmt.Print("이름:")
    fmt.Scanln(&name)
    scheduler <- name
}

func main() {
    scheduler := make(chan string)
    go consuming(scheduler)
    go producing(scheduler)

    time.Sleep(100 * time.Second)
}

```

- select: Producer and Consumer with timeout


```
package main
```

```
import (  
    "fmt"  
    "time"  
)
```

```
func consuming (scheduler chan string){  
    select {  
        case <- scheduler:  
            fmt.Println("이름을 입력받았습니다.")  
        case <-time.After(5 * time.Second):  
            fmt.Println("시간이 지났습니다.")  
    }  
}
```

```
func producing(scheduler chan string){  
    var name string  
    fmt.Print("이름:")  
    fmt.Scanln(&name)  
    scheduler <- name  
}
```

```
func main() {  
    scheduler := make(chan string)  
    go consuming(scheduler)  
    go producing(scheduler)  
  
    time.Sleep(100 * time.Second)  
}
```

```
package main
```

```
import (  
    "fmt"  
    "time"  
)
```

```
var scheduler chan string
```

```
func consuming (prompt string){ fmt.Println("consuming 호출됨")  
    select {  
        case scheduler <- prompt:  
            fmt.Println("이름을 입력받았습니다 : ", <- scheduler)  
        case <-time.After(5 * time.Second):  
            fmt.Println("시간이 지났습니다.")  
    }  
}
```

```
func producing (console chan string) {  
    var name string  
    fmt.Print("이름:")  
    fmt.Scanln(&name)  
    console <- name  
}
```

```
func main() {  
    console := make(chan string, 1)  
    scheduler = make(chan string, 1)  
  
    go func(){  
        consuming(<-console)  
    }()  
  
    go producing(console)  
  
    time.Sleep(100 * time.Second)  
}
```

Buffered Channel

```
package main

import "fmt"

func main() {
    c := make(chan int)
    c <- 1 //수신루틴이 없으므로 데드락
    fmt.Println(<-c) //코멘트해도 데드락 (별도의 Go루틴없기 때문)
}
```

```
package main

import "fmt"

func main() {
    ch := make(chan int, 1)

    //수신자가 없더라도 보낼 수 있다.
    ch <- 101

    fmt.Println(<-ch)
}
```

Buffered Channel

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     numbers := make(chan int, 5)
9     counter := 10
10
11     for i := 0; i < counter; i++ {
12         select {
13             case numbers <- i:
14             default:
15                 fmt.Println("Not enough space for", i)
16         }
17     }
18
19     for i := 0; i < counter+5; i++ {
20         select {
21             case num := <-numbers:
22                 fmt.Println(num)
23             default:
24                 fmt.Println("Nothing more to be done!")
25                 break
26         }
27     }
28 }
29
```

Terminate Go Routine

```
1 package main
2
3 import ...
4
5
6
7
8 func main() {
9     c1 := make(chan string)
10    go func() {
11        time.Sleep(time.Second * 3)
12        c1 <- "c1 OK"
13    }()
14
15    select {
16    case res := <-c1:
17        fmt.Println(res)
18    case <-time.After(time.Second * 1):
19        fmt.Println(a...: "timeout c1")
20    }
21
22    c2 := make(chan string)
23    go func() {
24        time.Sleep(3 * time.Second)
25        c2 <- "c2 OK"
26    }()
27
28    select {
29    case res := <-c2:
30        fmt.Println(res)
31    case <-time.After(4 * time.Second):
32        fmt.Println(a...: "timeout c2")
33    }
34 }
```

Nil Channel

```
1 package main
2
3 import (
4     "fmt"
5     "math/rand"
6     "time"
7 )
8
9 func add(c chan int) {
10     sum := 0
11     t := time.NewTimer(time.Second)
12
13     for {
14         select {
15             case input := <-c:
16                 sum = sum + input
17             case <-t.C:
18                 c = nil
19                 fmt.Println(sum)
20         }
21     }
22 }
23
24 func send(c chan int) {
25     for {
26         c <- rand.Intn(10)
27     }
28 }
29
30 func main() {
31     c := make(chan int)
32     go add(c)
33     go send(c)
34
35     time.Sleep(3 * time.Second)
36 }
```

Define Order

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func A(a, b chan struct{}) {
9     <-a
10    fmt.Println( a...: "A()!")
11    time.Sleep(time.Second)
12    close(b)
13 }
14
15 func B(a, b chan struct{}) {
16     <-a
17    fmt.Println( a...: "B()!")
18    close(b)
19 }
20
21 func C(a chan struct{}) {
22     <-a
23    fmt.Println( a...: "C()!")
24 }
25
26 func main() {
27     x := make(chan struct{})
28     y := make(chan struct{})
29     z := make(chan struct{})
30
31     go C(z)
32     go A(x, y)
33     go C(z)
34     go B(y, z)
35     go C(z)
36
37     close(x)
38     time.Sleep(3 * time.Second)
39 }
```

- **close(a)** closes the channel a, that synchronizes with a routine waiting for the channel a

Mutex

```
1 package main
2
3 import (
4     "fmt"
5     "os"
6     "strconv"
7     "sync"
8     "time"
9 )
10
11 var (
12     m sync.Mutex
13     v1 int
14 )
15
16 func change(i int) {
17     m.Lock()
18     time.Sleep(time.Second)
19     v1 = v1 + 1
20     if v1%10 == 0 {
21         v1 = v1 - 10*i
22     }
23     m.Unlock()
24 }
25
26 func read() int {
27     m.Lock()
28     a := v1
29     m.Unlock()
30     return a
31 }
```

```
33 func main() {
34     if len(os.Args) != 2 {
35         fmt.Println(a... "Please give me an integer!")
36         return
37     }
38
39     numGR, err := strconv.Atoi(os.Args[1])
40     if err != nil {
41         fmt.Println(err)
42         return
43     }
44     var waitGroup sync.WaitGroup
45
46     fmt.Printf(format: "%d ", read())
47     for i := 0; i < numGR; i++ {
48         waitGroup.Add(delta: 1)
49         go func(i int) {
50             defer waitGroup.Done()
51             change(i)
52             fmt.Printf(format: "-> %d", read())
53         }(i)
54     }
55
56     waitGroup.Wait()
57     fmt.Printf(format: "-> %d\n", read())
58 }
```