

## 목차

• Goroutine

Channel

Select

- 프로세스(Process)
  - 명령어, 사용자 데이터, 시스템 영역, 실행 과정에서 수집한 다양한 종류의 리소스로 구성된 독립적 실행 단위
- 프로그램(Program)
  - 프로세스의 명령어와 사용자 데이터를 초기화하는데 사용할 명령어와 데이 터를 담은 파일
- 스레드(thread)
  - 프로그램이나 프로세서보다 좀 더 가볍고 작은 실행단위
  - 프로세스에 의해 생성되며 독립적 제어 흐름과 스택을 가짐
  - 프로세스의 일부분

- Go 루틴(goroutine)
  - Go 프로그램에서 동시에 실행할 수 있는 최소 단위
  - OS에서 관리하는게 아닌 go runtime에서 관리
  - 프로세스 > 스레드 > goroutine
  - 특징
    - 굉장히 가벼움(4k) -> 보통 m 단위
    - 이론상 몇 만개를 사용해도 가능
    - 크기가 동적으로 증가
    - 채널을 활용한 동기화

```
package main
       import (
           "fmt"
           "time"
       func test(){
           fmt.Println( a...: "hi")
9
10
11 ▶
       func main() {
           go test()
12
           time.Sleep( d: 10)
13
14
15
        hi
```

```
package main
       jimport (
           "fmt"
           "time"
      func test(num int){
           fmt.Print( a...: "goroutine Num : ", num, " ")
10
     func main() {
           for i := 0 ; i < 10 ; i++{
12
               go test(i)
15
           time.Sleep( d: 1000000000)
16
```

goroutine Num : 9 goroutine Num : 6 goroutine Num : 7 goroutine Num : 8 goroutine Num : 2 goroutine Num : 3 goroutine Num : 1 goroutine Num : 0 goroutine Num : 4 goroutine Num : 5 Process finished with exit code 0

```
package main
         import (
             "fmt"
             "runtime"
             "time"
         func test(num int){
             fmt.Print( a...: "goroutine Num : ", num, " ")
 10
 11
         func main() {
 12 ▶
 13
             fmt.Println(runtime.GOMAXPROCS( n: 0))
             runtime.GOMAXPROCS(runtime.NumCPU())
 14
             fmt.Println(runtime.GOMAXPROCS( n: 0))
             for i := 0 ; i < 10 ; i++{
 16
                 go test(i)
 18
             time.Sleep( d: 1000000000)
 19
20
20
goroutine Num : 9 goroutine Num : 1 goroutine Num : 2 goroutine Num : 3 goroutine Nu
```

```
package main
      import (
          "fmt"
          "time"
      func main() {
8
          for i := 0 ; i < 10 ; i++{
9
             go func() {
10
                fmt.Println( a...: "closer : ", i)
12
             }()
                                     13
                                      closer: 10
14
         time.Sleep( d: 1000000000)
                                      closer: 10
15
                                      closer: 10
16
                                      closer: 10
                                      closer: 10
                                      closer: 10
                                      closer: 10
                                      closer: 10
                                      closer: 10
                                      closer: 10
```

```
package main
 2
       jimport (
 3
           "fmt"
           "time"
 6
      func main() {
           for i := 0 ; i < 10 ; i++{
               go func() {
10
                  fmt.Println( a...: "closer : ", i)
11
              }()
12

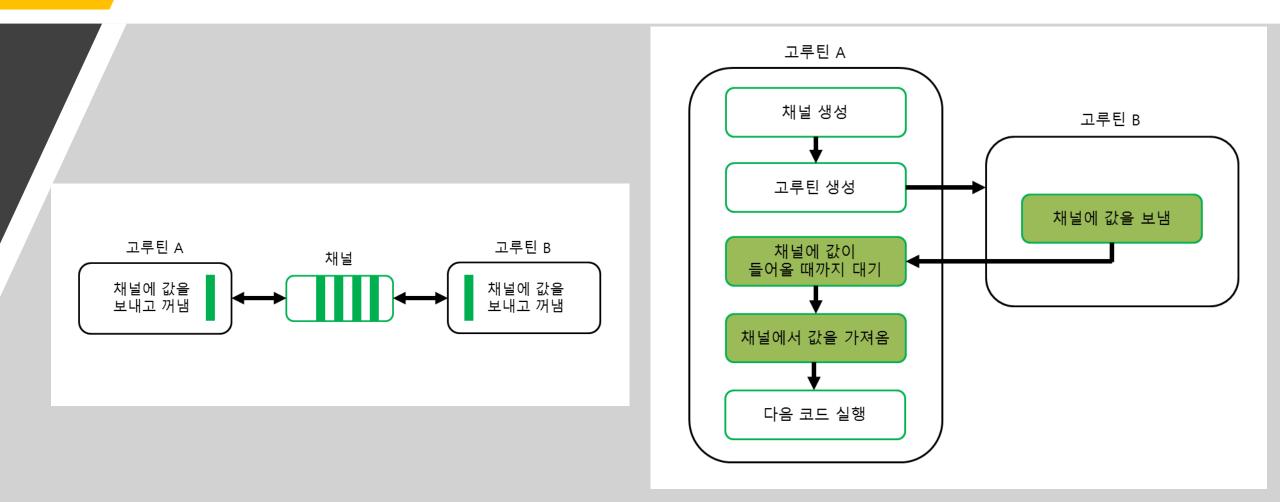
⊕<4 go setup calls>

13
              time.Sleep( d: 1)
                                                closer: 0
14
                                                closer: 1
           time.Sleep( d: 1000000000)
15
                                                closer: 2
16
                                                closer: 3
17
                                                closer: 4
                                                closer: 5
                                                closer: 6
                                                closer: 7
                                                closer: 8
                                                closer: 9
```

```
package main
      jimport (
          "fmt"
          "time"
5
       func main() {
          for i := 0 ; i < 10 ; i++{
9
              go func(num int) {
10
                  fmt.Println( a...: "closer : ", num)
11
              }(i)
12
                                          <4 go setup calls>
13
                                          closer: 9
          time.Sleep( d: 1000000000)
14
                                          closer: 0
15
                                          closer: 1
1.6
                                          closer: 2
                                          closer: 5
                                          closer: 6
                                          closer: 7
                                          closer: 8
                                          closer: 3
                                          closer: 4
```

```
package main
2
3
      import (
          "fmt"
5
          "time"
6
      func main() {
         buffer := []int{1,2,3,4,5}
         for i := 0 ; i < 10 ; i++{
10
             go func(num int) {
                buffer[3] *= num+1
12
                fmt.Println(num, "closer : ", i, buffer)
13
             }(i)
14
             //time.Sleep(1)
                                      9 closer: 10 [1 2 3 160 5]
16
                                      2 closer: 10 [1 2 3 7680 5]
         time.Sleep( d: 1000000000)
                                      0 closer: 10 [1 2 3 160 5]
18
                                      1 closer: 10 [1 2 3 320 5]
                                      5 closer: 10 [1 2 3 2073600 5]
                                     3 closer: 10 [1 2 3 160 5]
                                      4 closer: 10 [1 2 3 345600 5]
                                      8 closer: 10 [1 2 3 345600 5]
                                      7 closer: 10 [1 2 3 7680 5]
                                      6 closer: 10 [1 2 3 14515200 5]
```

- 채널(Channel)
  - Goroutine 끼리 데이터를 주고 받기 위한 통신 메커니즘
  - 특징
    - 모든 타입을 채널로 사용할 수 있음(Reference Type)
    - 각 채널마다 특정 데이터 타입으로 데이터를 교환(원소 타입, element type)
    - 채널로 데이터를 보내는 Goroutine이 존재해야 함
    - 채널 선언은 chan, 닫으려먼 close
    - 채널의 방향을 설정해야 함



출처: http://pyrasis.com/book/GoForTheReallyImpatient/Unit34

```
1
      package main
2
      import "fmt"
      func sum(a ,b int, c chan int){
5
          c <- a + b
      func main() {
          c := make(chan int)
9
          go sum( a: 1, b: 2,c)
10
          n := <- c
11
          fmt.Println(n)
12
13
    ↑ <4 go setup calls>
```

```
package main
       import "fmt"
       func sum(a ,b int, c chan int){
           c <- a + b
       func test(c chan string){
           c <- "hihi"
10
     func main() {
11 ▶
           channel := make(chan int)
           var channelString = make(chan string)
13
           go sum( a: 1, b: 2, channel)
14
           go test(channelString)
          n := <- channel
16
           temp := <- channelString
17
18
           fmt.Println(n, temp)
19
20
                <4 go setup calls>
                3 hihi
```

```
package main
      ⇒import (
           "fmt"
           "time"
       func main() {
 8
                                                           <4 go setup calls>
           channel := make(chan int)
 9
                                                           Goroutine: 0
           count := 3
10
                                                           Main Func : 0
           go func() {
                                                           Goroutine: 1
               for i:= 0 ; i < count ; i++{
12
                                                           Main Func : 1
                   channel <- i
13
                                                           Goroutine: 2
                   fmt.Println( a...: "Goroutine : ", i)
14
                                                           Main Func: 2
                   time.Sleep( d: 1000000000)
15
16
17
           }()
18
           for i:= 0 ; i < count ; i++{
               <- channel
20
               fmt.Println( a...: "Main Func : ", i)
21
22
23
24
```

- 동기 채널
- 값이 들어오면, 받아서 사용

```
package main
       import (
           "fmt"
      func main() {
7
           channel := make(chan int, 5)
8
9
           count := 10
           go func() {
10
               for i:= 0 ; i < count ; i++{
11
                   channel <- i
12
                   fmt.Println( a...: "Goroutine : ", i)
13
14
           }()
15
16
           for i:= 0 ; i < count ; i++{
17
18
               <- channel
19
               fmt.Println( a...: "Main Func : ", i)
20
21
22
```

• 비동기 채널

<4 go setup calls>

Goroutine: 0
Goroutine: 1
Goroutine: 2

Goroutine: 3

Goroutine: 4

Goroutine: 5

Main Func: 0

Main Func : 1 Main Func : 2

Main Func : 3

Main Func : 4

Main Func : 5

Main Func : 6

Goroutine: 6

Goroutine: 7

Goroutine: 8

Goroutine: 9

Main Func: 7

Main Func: 8

Main Func: 9

- 버퍼에 값이 있으면, 알아서 사용
- 보통 버퍼가 가득 차면 값을 꺼내서 사용

```
input Channel
      package main
                                                       out channel: 0
      import (
                                                       out channel: 1
          "fmt"
                                                       input Channel
                                                       input Channel
                                                       out channel: 2
     func main() {
                                                       out channel: 3
         channel := make(chan int)
         count := 10
                                                       input Channel
         go func() {
                                                       input Channel
             for i:= 0 ; i < count ; i++{
                                                       out channel: 4
                 channel <- i
                                                       out channel: 5
                fmt.Println( a...: "input Channel")
                                                       input Channel
                //time.Sleep(1)
14
15
                                                       input Channel
             close(channel)
16
                                                       out channel: 6
         }()
17
                                                       out channel: 7
18
                                                       input Channel
         for i := range channel{
19
             fmt.Println( a...: "out channel : ", i)
                                                       input Channel
20
                                                       out channel: 8
22
                                                       out channel: 9
23
                                                       input Channel
24
```

- Print 함수와 화면 표시 순서가 다름
- Main goroutine이 먼저 실행 된다고 해도, 화면 IO 부분에서 순서가 바뀔 수 있음

```
package main
        jimport (
             "fmt"
5
6
        func main() {
             channel := make(chan int)
8
                                                                           input Channel
                                                                           out channel: 0
             count := 10
9
                                                                           out channel: 1
             go func() {
10
                                                                           input Channel
                                                                           input Channel
                   for i:= 0 ; i < count ; i++{
                                                                           out channel: 2
                        channel <- i
                                                                           out channel: 3
                                                                           input Channel
                        fmt.Println( a...: "input Channel")
                                                                           input Channel
                        //time.Sleep(1)
14
                                                                           out channel: 4
                                                                           out channel: 5
                                                                           input Channel
                   //close(channel)
                                                                           input Channel
                                                                           out channel: 6
             }()
17
                                                                           out channel: 7
18
                                                                           input Channel
                                                                           input Channel
             for i := range channel{
19
                                                                           out channel: 8
                   fmt.Println( a...: "out channel : ", i)
                                                                           out channel: 9
                                                                           input Channel
                                                                           fatal error: all goroutines are asleep - deadlock!
                                                                           goroutine 1 [chan receive]:
                                                                           main.main()
                                                                              C:/Users/holly/Desktop/workspace_go/src/main/2day.go:19 +0x11d
24
```

- 채널을 안 닫아주면 Deadlock 발생
- 무한정 대기하기 때문

```
package main
1
2
       import "fmt"
3
5
       func main() {
           channel := make(chan int)
 6
7
          go func() {
               channel <- 1
9
          }()
10
11
           a, ok := <- channel
12
          fmt.Println(a, ok)
13
14
           close(channel)
15
           a, ok = <- channel
16
          fmt.Println(a, ok)
17
18
19
20
       <4 go setup calls>
       1 true
       0 false
   ===
```

• 채널 오픈 유무 확인

```
package main
       import (
           "fmt"
           "time"
5
       func test1(c <-chan int){
8
           fmt.Println(<-c)</pre>
                                              10
                                             11
10
       func test2(c chan<- int){
                                             12
11
                                             13
           c <- 10
12
13
       func main() {
14
                                             16
           channel := make(chan int)
15
                                             17
16
                                             18
           go test1(channel)
                                             19
           go test2(channel)
18
                                             20
19
                                             21
           time.Sleep( d: 1000000)
20
21
22
          <4 go setup calls>
          10
```

```
package main
   import (
       "fmt"
       "time"
  func test1(c chan<- int){
       fmt.Println(<-c)</pre>
   func test2(c <-chan int){
       c <- 10
  func main() {
       channel := make(chan int)
       go test1(channel)
       go test2(channel)
       time.Sleep( d: 1000000)
⊞∖5 yu setup catts/
 # command-line-arguments
 .\2day.go:9:14: invalid operation: <-c (receive from send-only type chan<- int)
```

• <-chan : 보내기 전용

• chan<-: 받기 전용

.\2day.go:12:4: invalid operation: c <- 10 (send to receive-only type <-chan int)

- 셀렉트(Select)
  - 여러 개의 채널을 하나의 Select block에서 다룰 수 있음
  - 특징
    - Switch문과 같은 형식을 가짐 (인자가 Channel)
    - Goroutine 연산 여러 개를 컨트롤 가능

- 셀렉트(Select)
  - 여러 개의 채널을 하나의 Select block에서 다룰 수 있음
  - 특징
    - Switch문과 같은 형식을 가짐 (인자가 Channel)
    - Goroutine 연산 여러 개를 컨트롤 가능

```
package main
     import ...
c1 := make(chan int)
         c2 := make(chan string)
        go func() {
            for {
                i := <-c1
                fmt.Println( a...: "c1 :", i)
                time.Sleep(100 * time.Millisecond)
         }()
         go func() {
             for {
                c2 <- "Hello, world!"
                time.Sleep(500 * time.Millisecond)
         }()
         go func() {
             for { // 무한 루프
                select {
                case c1 <- 10:
                case s := <-c2:
                    fmt.Println( a...: "c2 :", s)
         }()
         time.Sleep(10 * time.Second)
```

```
i: 卫 go build 2day.go 🗵
 ↑ c1 : 10
    c1 : 10
    c2 : Hello, world!
     c1 : 10
     c1 : 10
    c1 : 10
    c1 : 10
     c1 : 10
     c2 : Hello, world!
     c1 : 10
     c2 : Hello, world!
     c1 : 10
     c1 : 10
     c1 : 10
     c1 : 10
     c2 : Hello, world!
     c1 : 10
     c2 : Hello, world!
     c1 : 10
     c1 : 10
     c1 : 10
     c1 : 10
```

```
package main
      import ...
     func main() {
          c1 := make(chan int)
          c2 := make(chan string)
10
          go func() {
            for {
                 i := <-c1
                 fmt.Println( a...: "c1 :", i)
14
                 time.Sleep(100 * time.Millisecond)
            }
          }()
18
          go func() {
            for {
19
20
                  c2 <- "Hello, world!"
                  time.Sleep(500 * time.Millisecond)
          }()
          go func() {
              for { // 무한 루프
                  select {
26
                  case <-time.After(50 * time.Millisecond):</pre>
28
                      fmt.Println( a...: "timeOut!")
                      return
29
                  case c1 <- 10:
30
                  case s := <-c2:
                     fmt.Println( a...: "c2 :", s)
34
          }()
38
          time.Sleep(10 * time.Second)
39
40
```

```
↑ ⊞<4 go setup calls>
c2 : Hello, world!
c1 : 10
timeOut!
```