## Jan 20, 2025 | [Golang Training | Interns Batch Jan 2025](https://www.google.com/calendar/event?eid=NHN0cTB1ZzJnMWFjdmxkNTZvMWx2Zzh2OWdfMjAyNTAxMjBUMDQzMDAwWiBzYWhpbC5iaGVrZUBqb3Noc29mdHdhcmUuY29t)

Trainer: [Sagar Sonwane](mailto:sagar.sonwane@joshsoftware.com) Mentor: [Sparsh Arora](mailto:sparsh.arora@joshsoftware.com)

Notes and Assignment

* Are GoRoutines and thread same?

| GoRoutines are not equ ivalent to thread, Threads are more expensive to create and switching takes longer time. One single OS thread can run multiple GoRoutines. GoRoutines are abstraction over thread.  Millions of goroutines can be scheduled per second. |
| --- |

* Why not to use time.Sleep to wait for goroutines?

| 1. It disturbs the responsiveness of the program. 2. time.Sleep() means our program has a race condition. |
| --- |

* how golang schedular manages goroutines, which causes the sequence to be random?

| Order of execution of threads are completely dependent on the OS based on priority and I/O operation. But since GoRoutines are abstraction over thread they all have same priority. Therefore we cannot control the order of execution and this causes the sequence to be random.  Reference link: https://dev.to/gophers/what-are-goroutines-and-how-are-they-scheduled-2nj3 |
| --- |

* Buffered channel of capacity 1 is used to make sure only 1 go routine can access the critical section at a time. write a code to justify this statement

package main

import (

"fmt"

"sync"

)

func write(ch chan \*int, wg \*sync.WaitGroup) {

var v \*int = <-ch

\*v++

fmt.Println("Value in first go routine is", \*v)

wg.Done()

}

func write2(ch chan \*int, wg \*sync.WaitGroup) {

var v \*int = <-ch

\*v++

fmt.Println("Value in second go routine is", \*v)

wg.Done()

}

func main() {

ch := make(chan \*int, 1)

var wg sync.WaitGroup

var data int = 10

ch <- &data

wg.Add(1)

go write(ch, &wg)

ch <- &data

wg.Add(1)

go write2(ch, &wg)

wg.Wait()

fmt.Println("In main ", data)

// fmt.Println("Hello World")

}

* Find use cases for buffered and unbuffered channels

| 1. Unbuffered channels are used for synchronized communication between goroutines. It is usually for direct communication between sender and receiver, ensuring that the both ends are ready to exchange data. 2. Buffered channels is used when we want flexibility. Such that sender and receiver can send and receive data as per their availability.(independent operation)   Reference link: https://www.linkedin.com/pulse/buffered-channel-vs-unbuffered-go-hamed-yarandi |
| --- |

* Understand and prepare a document on how Go schedular manages goroutines

| Threads that internally multiplexed on the cores of the processor.  Its the job of the schedular to manage threads. Threads can be infinite and these threads are coupled together to run on a single core.  There are some methods to map goroutines to the threads:  1:1 mapping of goroutines to threads  one goroutine is mapped to the one thread  M:N mapping.  M goroutines are mapped to the n threads.  Reference link: https://medium.com/@sanilkhurana7/understanding-the-go-scheduler-and-looking-at-how-it-works-e431a6daacf |
| --- |
|  |

* Handling critical section using mutex

| var data int = 10  func write(wg \*sync.WaitGroup, m \*sync.Mutex) {   m.Lock()  data++  m.Unlock()  wg.Done()  fmt.Println("The data value in first go routine is ", data) }  func write2(wg \*sync.WaitGroup, m \*sync.Mutex) {   m.Lock()  data += 2  m.Unlock()  wg.Done()  fmt.Println("The data value in second go routine is ", data) }  func main() {   var wg sync.WaitGroup  var m sync.Mutex   fmt.Println("Initial Data is ", data)  wg.Add(1)  go write(&wg, &m)  wg.Add(1)  go write2(&wg, &m)   time.Sleep(2 \* time.Second)  } |
| --- |

* **How scheduler manages goroutines :** [**https://www.youtube.com/watch?v=5LfqrEGwDmE**](https://www.youtube.com/watch?v=5LfqrEGwDmE) [**https://segmentfault.com/a/1190000040186074/en**](https://segmentfault.com/a/1190000040186074/en)
* **CSP:**  [**http://www.usingcsp.com/**](http://www.usingcsp.com/) [**https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf**](https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf) [**https://go.dev/blog/codelab-share**](https://go.dev/blog/codelab-share)

[**https://go.dev/doc/codewalk/sharemem/**](https://go.dev/doc/codewalk/sharemem/)

* And, also try to understand all the go env settings - [**runtime.GOMAXPROCS**](https://go.dev/pkg/runtime#GOMAXPROCS)