Golang Concurrency Bugs

Gufran Mirza
gufranmirza

What this talk is about

1. Different types Concurrency Bugs Golang
2. Common pitfalls while writing concurrent programs in Golang
3. Solutions to some of common concurrency Bugs

Contents

How Concurrency is achieved in Golang

1. In Go, concurrency is achieved by using **Goroutines**
2. Golang runtime takes care of creating actual threads and scheduling of goroutines
3. Goroutines != threads
4. Goroutines uses **channels** to communicate safely between themselves

What are the Blocking Bugs ?

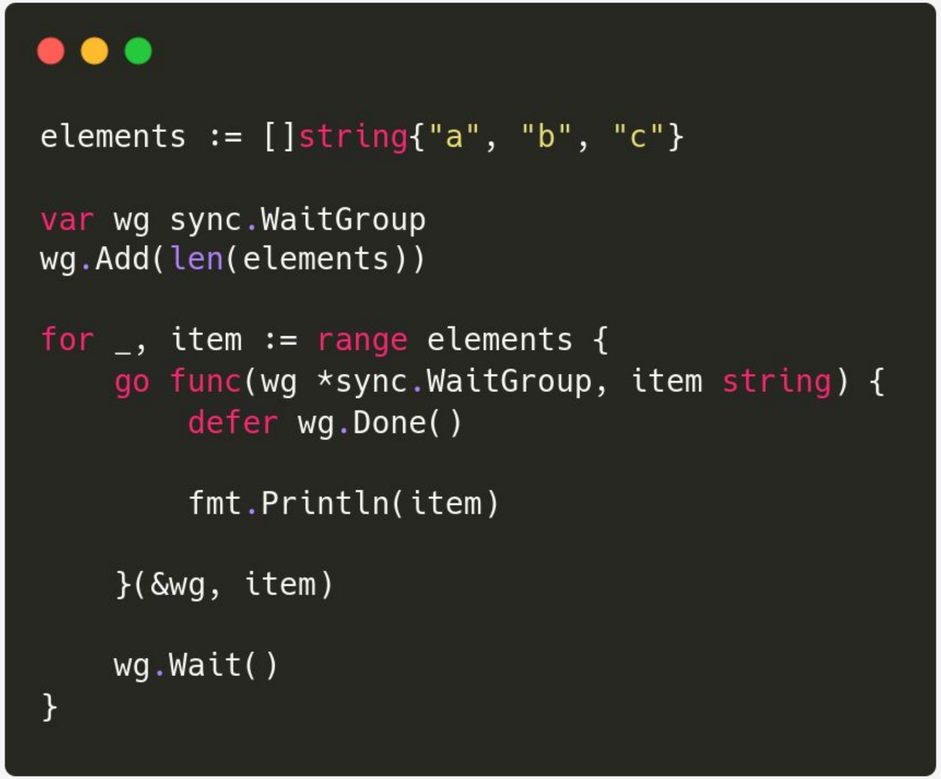
1. When one or more goroutines conduct operations that wait for resources, and these resources are never available

2. If this happens, you program will wait indefinitely for resources to become available

3. Caused by misuse of Mutex, WaitGroup and Channels etc.

1. A blocking bug caused by WaitGroup

1. Waiting on cond.Wait(), until signal is received
2. Never received expected cond.Signal() from goroutines
3. Goroutines panicked, or you forgot call cond.Done() in it
4. Most likely to happen due to misplacement of wg.Add(), wg.Done(), wg.Wait()

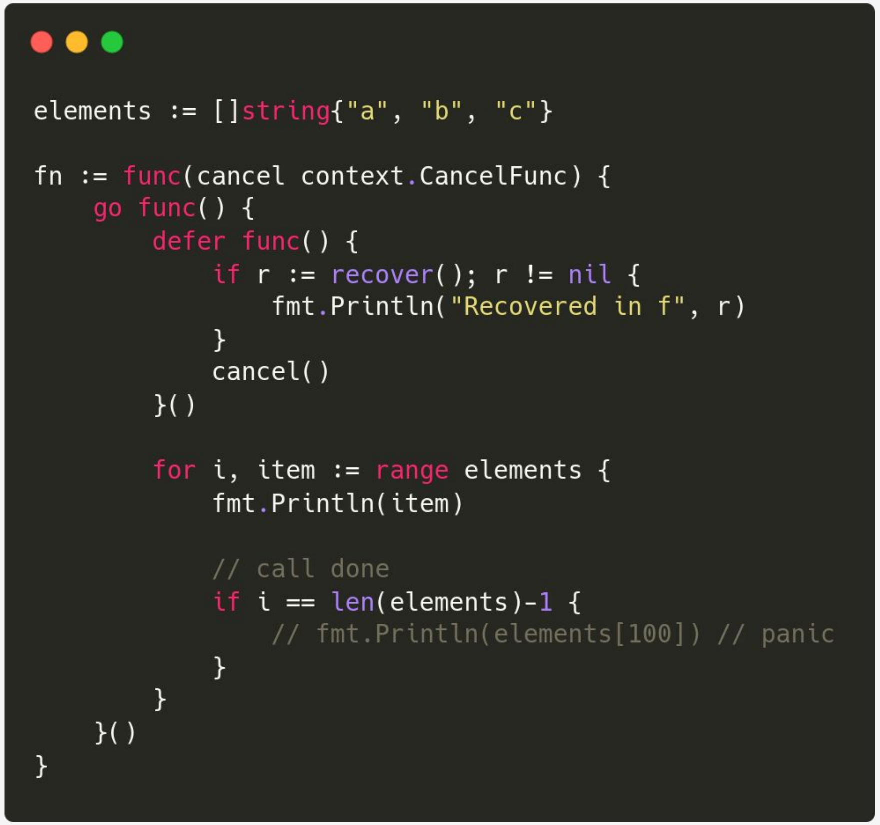


```
elements := []string{"a", "b", "c"}

var wg sync.WaitGroup
wg.Add(len(elements))

for _, item := range elements {
    go func(wg *sync.WaitGroup, item string) {
        defer wg.Done()

        fmt.Println(item)

    }(&wg, item)

    wg.Wait()
}
```

2. A blocking bug caused by context(ctx)

1. Waiting on ctx.Done(), until signal is received
2. Never received expected ctx.Signal() from goroutines
3. Goroutines panicked, or you forgot call ctx.CancelFunc() in it
4. likely to happen if context is never cancelled(timedout)

Blocking Bugs

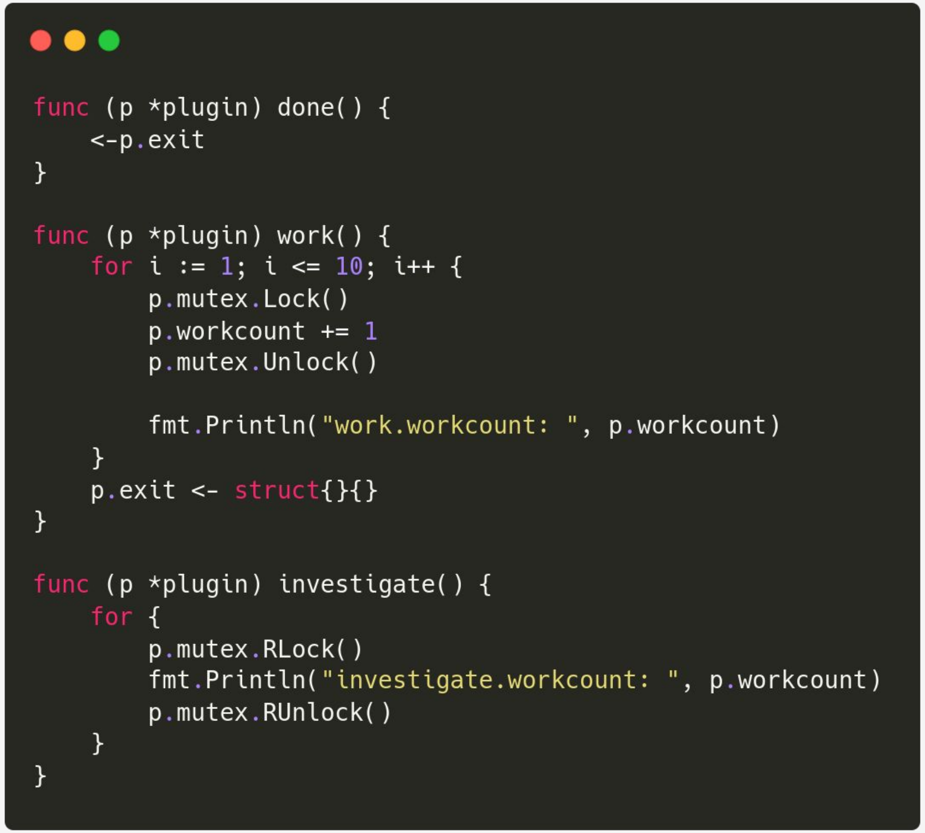


```go
elements := []string{"a", "b", "c"}

fn := func(cancel context.CancelFunc) {
    go func() {
        defer func() {
            if r := recover(); r != nil {
                fmt.Println("Recovered in f", r)
            }
            cancel()
        }()

        for i, item := range elements {
            fmt.Println(item)

            // call done
            if i == len(elements)-1 {
                // fmt.Println(elements[100]) // panic
            }
        }
    }()
}
```

3. A blocking bug due to wrong usage of channel with lock

1. Waiting on receive channel, until signal is received
2. Waiting to acquire a Lock on the resource
3. Goroutines panicked, or you forgot call mutex.Unlock(), or channel never signaled <- struct{}{}

Blocking Bugs



```go
func (p *plugin) done() {
    <-p.exit
}

func (p *plugin) work() {
    for i := 1; i <= 10; i++ {
        p.mutex.Lock()
        p.workcount += 1
        p.mutex.Unlock()

        fmt.Println("work.workcount: ", p.workcount)
    }
    p.exit <- struct{}{}
}

func (p *plugin) investigate() {
    for {
        p.mutex.RLock()
        fmt.Println("investigate.workcount: ", p.workcount)
        p.mutex.RUnlock()
    }
}
```

References

Credits

- https://songlh.github.io/paper/go-study.pdf

Github Repo

- https://github.com/gufranmirza/talks/tree/main/2022/02-golang-concurrency-bugs

Questions…?

Contact me

Gufran Mirza

Software engineer @IBM Cloud

@_imGufran

@gufranmirza

gufranmirza.com