Concurrency in Go

Real World Basics

Chris Bruce
Piper
@chrisabruce

An Analogy...

Single CPU

Single CPU

Multi-Threaded

Multi-Core/CPU


VPS
920-863-6505
1-866-863-6505

Where the Problems Exist

Race Conditions


KNOCK! KNOCK!
Race Condition
Who's There?

Deadlocks

Go's Answer

Go Routines

Channels

Scheduler

Mutexes & Semaphores

Realworld Patterns

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    for {
        go func() {
            fmt.Println(time.Now())
        }()
    }
}

```

Wait! Not this!

These...

Errgroup

```go
var g errgroup.Group
var urls = []string{
    "http://www.golang.org/",
    "http://www.google.com/",
    "http://www.somestupidname.com/",
}
for _, url := range urls {
    // Launch a goroutine to fetch the URL.
    url := url // https://golang.org/doc/faq#closures_and_goroutines
    g.Go(func() error {
        // Fetch the URL.
        resp, err := http.Get(url)
        if err == nil {
            resp.Body.Close()
        }
        return err
    })
}
// Wait for all HTTP fetches to complete.
if err := g.Wait(); err == nil {
    fmt.Println("Successfully fetched all URLs.")
}
```

Bounds

```go
package main

import "fmt"

func main() {
    maxLimit := 10
    longList := [1000]int{}
    sem := make(chan bool, maxLimit)

    for _, i := range longList {
        sem <- true
        go func(work int) {
            defer func() { <-sem }()
            doStuff(work)
        }(i)
    }

    for i := 0; i < cap(sem); i++ {
        sem <- true
    }
}

func doStuff(work int) {
    fmt.Printf("stuff: %d\n", work)
}
```

Timeouts

```
2
3  import (
4      "context"
5      "fmt"
6      "time"
7  )
8
9  func main() {
10     // Pass a context with a timeout to tell a blocking function that it
11     // should abandon its work after the timeout elapses.
12     ctx, cancel := context.WithTimeout(context.Background(), 1*time.Second)
13     defer cancel()
14
15     go func() {
16         time.Sleep(10 * time.Minute)
17     }()
18
19     select {
20     case <-ctx.Done():
21         fmt.Println(ctx.Err()) // prints "context deadline exceeded"
22     }
23 }
```

sync.Mutex

```go
type Message struct {
    To string,
    From string,
    Body string,
}

type PushNotifQueue struct {
    sync.Mutex
    Msgs   []*Message
}

func (q *PushNotifQueue) Queue(msg *Message) {
    q.Lock()
    q.Msgs = append(q.Msgs, msg)
    q.Unlock()
}
```

Thank You.

We are hiring: cb@playpiper.com
(go, rust, react)

Chris Bruce
Piper
@chrisabruce