Урок Спринт 2.6

Cond

Давайте рассмотрим следующую задачу:

- Есть 3 горутины: G1, G2, G3
- Горутина G1 получает данные из некоторого источника
- Горутины G2 и G3 должны дождаться того, как горутина G1 получит эти данные что-то с ними сделать

То есть G2 и G3 должны быть заблокированы до того момента, как G1 не даст им какой-то сигнал.

Стандартная библиотека предоставляет нам объект `sync.Cond`. Конструктор его принимает в качестве аргумента `Locker` — мы будем использовать объект `Mutex`

```
cond := sync.NewCond(&sync.Mutex{})
```

У него есть три важные проассоциированные функции

- `Signal()` - отправляет сигнал одной горутине
- `Broadcast()` - отправляет сигнал всем горутинам
- `Wait()` - ожидает сигнал

Напишем функцию слушателя, которая ждет сигнала о возможности начала обработки данных:

```
func listen(name string, data map[string]string, c *sync.Cond) {
    c.L.Lock()
    c.Wait()

    fmt.Printf("[%s] %s\n", name, data["key"])

    c.L.Unlock()
}
```

Напишем горутину, которая получает данные и посылает слушателям сигнал о начале их обработки.

```
func broadcast(name string, data map[string]string, c *sync.Cond) {
```

```
    time.Sleep(time.Second)

    c.L.Lock()

    data["key"] = "value"

    fmt.Printf("[%s] данные получены\n", name)

    // отправляем сигнал слушателям
    c.Broadcast()
    c.L.Unlock()
}
```

Вызов `cond.Wait` обязательно нужно вызывать до вызова `cond.Broadcast`, иначе слушатели повиснут навсегда.

Скомпануем имеющиеся функции:

```
package main

import (
    "fmt"
    "os"
    "os/signal"
    "sync"
    "time"
)

func listen(name string, data map[string]string, c *sync.Cond) {
    c.L.Lock()
    c.Wait()

    fmt.Printf("[%s] %s\n", name, data["key"])

    c.L.Unlock()
}

func broadcast(name string, data map[string]string, c *sync.Cond) {
    time.Sleep(time.Second)

    c.L.Lock()

    data["key"] = "value"

    fmt.Printf("[%s] данные получены\n", name)

    c.Broadcast()
    c.L.Unlock()
}

func main() {
    data := map[string]string{}
```

```
    cond := sync.NewCond(&sync.Mutex{})

    go listen("слушатель 1", data, cond)
    go listen("слушатель 2", data, cond)

    go broadcast("источник", data, cond)

    ch := make(chan os.Signal, 1)
    signal.Notify(ch, os.Interrupt)
    <-ch
}
```

Алгоритм использования Cond следующий:

— создать и инициализировать экземпляр `sync.Cond` с помощью `sync.NewCond(&sync.Mutex{})`

— ожидать выполнения условия

```
cond.L.Lock()
for !condition {
  cond.Wait()
}
// выполнение задачи
cond.L.Unlock()
```

— сигнализировать о выполнении условия `cond.Signal()`