



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 5

Название: Основы асинхронного программирования на Golang

Дисциплина: Языки интернет-программирования

Студент

ИУ6-31Б

(Группа)

(Подпись, дата)

Л. А. Круглов

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В. Д. Шульман

(И.О. Фамилия)

Москва, 2024

Цель работы: изучение основ асинхронного программирования с использованием языка Golang.

Задание:

1. Ознакомьтесь с разделом "3. Map, файлы, интерфейсы, многопоточность и многое другое".
2. Сделайте форк данного репозитория в GitHub, клонируйте получившуюся копию локально, создайте от мастера ветку дев и переключитесь на неё
3. Выполните задания. Ссылки на задания содержатся в README-файлах в директории projects
4. Сделайте отчёт и поместите его в директорию docs
5. Зафиксируйте изменения, сделайте коммит и отправьте полученное состояние ветки дев в ваш удаленный репозиторий GitHub
6. Через интерфейс GitHub создайте Pull Request dev --> master
7. На защите лабораторной работы продемонстрируйте открытый Pull Request. PR должен быть направлен в master ветку вашего репозитория

Ход работы:

1. Задача «Calculator»

1) Условие:

Вам необходимо написать функцию calculator следующего вида:

```
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int
```

Функция получает в качестве аргументов 3 канала, и возвращает канал типа <-chan int.

- в случае, если аргумент будет получен из канала firstChan, в выходной (возвращенный) канал вы должны отправить квадрат аргумента.
- в случае, если аргумент будет получен из канала secondChan, в выходной (возвращенный) канал вы должны отправить результат умножения аргумента на 3.
- в случае, если аргумент будет получен из канала stopChan, нужно просто завершить работу функции.

Функция calculator должна быть неблокирующей, сразу возвращая управление. Ваша функция получит всего одно значение в один из каналов - получили значение, обработали его, завершили работу.

После завершения работы необходимо освободить ресурсы, закрыв выходной канал, если вы этого не сделаете, то превысите предельное время работы.

Рисунок 1 - Условие задачи №1

2) Решение:

```
package main
import (
    "fmt"
    "sync"
)
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{})
<-chan int {
    ch := make(chan int)
    go func() {
        select {
        case inp := <-firstChan:
            ch <- inp * inp
        case inp := <-secondChan:
            ch <- inp * 3

        case _ = <-stopChan:
            break
        }
        close(ch)
    }()
    return ch
}
func main() {
    fch := make(chan int)
    sch := make(chan int)
    stop := make(chan struct{})
```

```

ch := calculator(fch, sch, stop)
wg := sync.WaitGroup{}
wg.Add(1)
go func() {
    for ans := range ch {
        fmt.Println(ans)
    }
    wg.Done()
}()
fch <- 10
wg.Wait()
}

```

3) Тестирование:

```

leonid@MBP-Leonid lab_web_5 % go run ./projects/calculator/main.go
100

```

Рисунок 2 - Тестирование задачи №1

2. Задача «Pipeline»

1) Условие:

Напишите элемент конвейера (функцию), что запоминает предыдущее значение и отправляет значения на следующий этап конвейера только если оно отличается от того, что пришло ранее.

Ваша функция должна принимать два канала - inputStream и outputStream, в первый вы будете получать строки, во второй вы должны отправлять значения без повторов. В итоге в outputStream должны остаться значения, которые не повторяются подряд. Не забудьте закрыть канал ;)

Функция **должна** называться `removeDuplicates()`

Выводить или вводить ничего не нужно!

Рисунок 3 - Условие задачи №2

2) Решение:

```

package main
import (
    "fmt"
    "sync"
)
func removeDuplicates(inputStream chan string, outputStream chan string) {
    var last string
    for s := range inputStream {
        if s != last {
            outputStream <- s
        }
        last = s
    }
    close(outputStream)
}

```

```

    }

func main() {
    input := make(chan string)
    output := make(chan string)
    go removeDuplicates(input, output)
    wg := sync.WaitGroup{}
    wg.Add(1)
    go func() {
        for ans := range output {
            fmt.Println(ans)
        }
        wg.Done()
    }()
    input <- "Hello"
    input <- "Hello"
    input <- "World"
    close(input)
    wg.Wait()
}

```

3) Тестирование:

```

leonid@MBP-Leonid lab_web_5 % go run ./projects/pipeline/main.go
Hello
World

```

Рисунок 4 - Тестирование задачи №2

3. Задача «Work»

1) Условие:

Внутри функции `main` (функцию объявлять не нужно), вам необходимо в отдельных горутинах вызвать функцию `work()` 10 раз и дождаться результатов выполнения вызванных функций.

Функция `work()` ничего не принимает и не возвращает. Пакет `"sync"` уже импортирован.

Рисунок 5 - Условие задачи №3

2) Решение:

```

package main

import (
    "fmt"
    "sync"
)

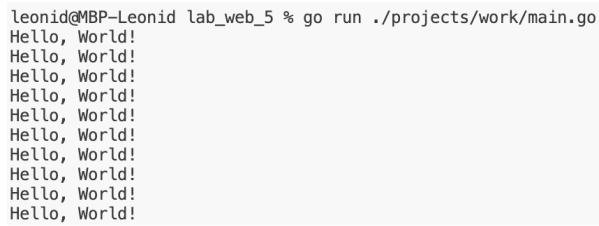
func work() {
    fmt.Println("Hello, World!")
}

func main() {
    wg := sync.WaitGroup{}
    wg.Add(10)

```

```
    for i := 0; i < 10; i++ {  
        go func() {  
            work()  
            wg.Done()  
        }()  
    }  
    wg.Wait()  
}
```

3) Тестирование:



```
leonid@MBP-Leonid lab_web_5 % go run ./projects/work/main.go  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!  
Hello, World!
```

Рисунок 6 - Тестирование задачи №3

Заключение: асинхронность Golang помогает выполнять параллельные задачи, а также вычислять результат для входного потока данных.

Список источников:

- Сайт: <https://stepik.org/>
- Сайт: <https://go.dev/>