



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)

Д.Ю. Воронин

(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" курса <https://stepik.org/course/54403/info>
2. Сделайте форк данного репозитория в GitHub, склонируйте получившуюся копию локально, создайте от мастера ветку dev 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 dev 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

#### 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 Calculator

#### Решение:

```
package main
```

```
import (  
    "fmt"  
)
```

```
// реализовать calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-  
chan struct{ }) <-chan int  
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{ })  
<-chan int {  
    out := make(chan int)  
    go func() {  
        defer close(out)  
        select {  
        case value := <-firstChan:  
            out <- value * value  
        case value := <-secondChan:
```

```

        out <- value * 3
    case <-stopChan:

    }
}()
return out
}

func main() {
    channel1, channel2 := make(chan int), make(chan int)
    stopChan := make(chan struct{ })
    calc := calculator(channel1, channel2, stopChan)

    // Тестирование трех случаев
    channel1 <- 5
    //channel2 <- 100
    //stopChan <- struct{ }{ }
    fmt.Println(<-calc)
}

```

### Тестирование

- 1) Передадим в канал 1 значение 5. Тогда мы должны получить его квадрат. Результат представлен на Рисунке 2.

```

admin@MBP-admin web-5 % go run "/Users/admin/Desktop/WEB/web-5/projects/calculator/main.go"
25

```

Рисунок 2 – Тестирование 1 задачи Calculator

- 2) Передадим в канал 2 значение 100. Тогда мы должны получить произведение переданного числа и числа 3. Результат представлен на Рисунке 3.

```

admin@MBP-admin web-5 % go run "/Users/admin/Desktop/WEB/web-5/projects/calculator/main.go"
300

```

Рисунок 3 – Тестирование 2 задачи Calculator

3) Передадим значение в канал stopChan. Тогда функция должна вернуть ничего. Результат представлен на Рисунке 4.

```
admin@MBP-admin web-5 % go run "/Users/admin/Desktop/WEB/web-5/projects/calculator/main.go"
0
```

Рисунок 4 – Тестирование 3 задачи Calculator

## 2. Задача Pipeline

### 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 5 – Условие задачи Pipeline

### Решение:

```
package main
```

```
import "fmt"
```

```
// реализовать removeDuplicates(in, out chan string)
```

```
func removeDuplicates(inputStream chan string, outputStream chan string) {
    last := ""
    for {
        s, opened := <-inputStream
        if !opened {
            break
        }
        if last != s {
            outputStream <- s
            last = s
        }
    }
}
```

```

    }
    close(outputStream)
}

func main() {
    // здесь должен быть код для проверки правильности работы функции
removeDuplicates(in, out chan string)
    inputStream := make(chan string)
    outputStream := make(chan string)
    go removeDuplicates(inputStream, outputStream)
    s := "1112244115668" // Значение для тестирования
    go func() {
        defer close(inputStream)
        for _, symbol := range s {
            inputStream <- string(symbol)
        }
    }()

    for symbol := range outputStream {
        fmt.Print(symbol)
    }
}

```

### Тестирование

```

admin@MBP-admin web-5 % go run "/Users/admin/Desktop/WEB/web-5/projects/pipeline/main.go"
1241568%

```

Рисунок 6 – Тестирование задачи Pipeline

### 3. Задача Work

#### 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 7 – Условие задачи Work

#### Решение:

```
package main

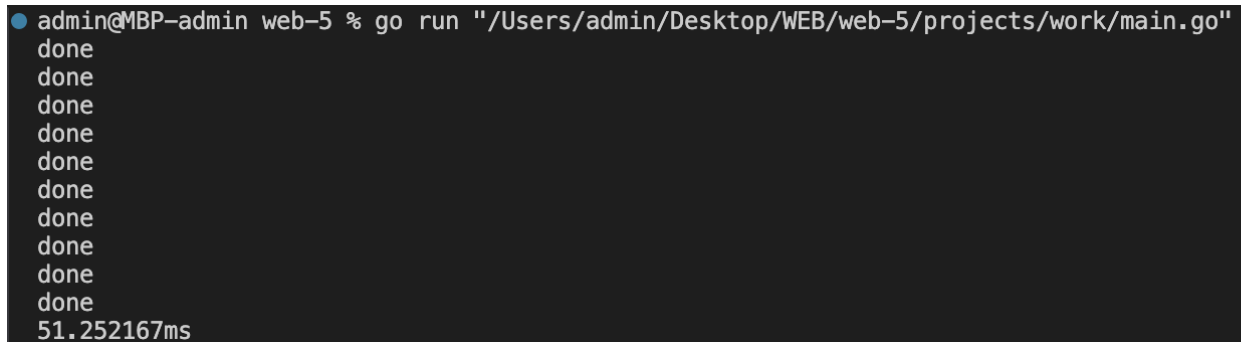
import (
    "fmt"
    "sync"
    "time"
)

func work() {
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}

func main() {
    start := time.Now()
    // необходимо в отдельных горутинах вызвать функцию work() 10 раз и
    // дождаться результатов выполнения вызванных функций
    wg := new(sync.WaitGroup)
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            work()
        }()
    }
}
```

```
}  
wg.Wait()  
t := time.Now()  
elapsed := t.Sub(start)  
fmt.Println(elapsed)  
}
```

## Тестирование



```
admin@MBP-admin web-5 % go run "/Users/admin/Desktop/WEB/web-5/projects/work/main.go"  
done  
done  
done  
done  
done  
done  
done  
done  
done  
done  
done  
51.252167ms
```

Рисунок 8 – Тестирование задачи Work

4. Загрузим решения на GitHub и сделаем Pull request из dev в master с помощью интерфейса GitHub.

### Заключение:

Язык программирования Golang предоставляет возможность асинхронной работы при помощи горутинов, что позволяет выполнять задачи параллельно.

### Список использованных источников:

<https://github.com/ValeryBMSTU/web-5>

<https://stepik.org/course/54403/info>