



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

**ОТЧЕТ**

по лабораторной работе № 5

**Название:** Основы асинхронного программирования на Golang

**Дисциплина:** Языки интернет-программирования

Студент

ИУ6-33Б  
(Группа)

06.09.2024  
(Подпись, дата)

Н.Н. Товарас  
(И.О. Фамилия)

Преподаватель

14.09.2024  
(Подпись, дата)

В.Д. Шульман  
(И.О. Фамилия)

Москва, 2024

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

Ход работы:

1) Задание: 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()`

Решение:

```
package main
```

```
import "fmt"
```

```
func removeDuplicates(inputStream <-chan string, outputStream chan<- string) {  
    var prev string  
    firstValue := true // Флаг для первого значения  
  
    for val := range inputStream {  
        if firstValue || val != prev {  
            outputStream <- val  
            prev = val  
            firstValue = false  
        }  
    }  
    close(outputStream) // Закрываем канал  
}
```

```
func main() {  
    // здесь должен быть код для проверки правильности работы функции  
    removeDuplicates(in, out chan string)  
    input := make(chan string)  
    output := make(chan string)
```

```
    go func() {  
        input <- "hello"  
        input <- "world"  
        input <- "world" // Дубликат  
        input <- "golang"  
        input <- "golang" // Дубликат
```

```

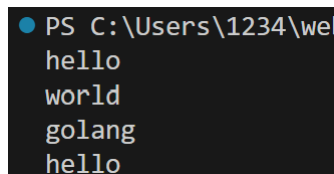
    input <- "hello"
    close(input) // Закрываем канал input
  }()

  go removeDuplicates(input, output)

  for i := range output {
    fmt.Println(i)
  }
}

```

Результат:



```

PS C:\Users\1234\we
hello
world
golang
hello

```

Рис.1 — Результат 1

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

Решение:

```
package main
```

```

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

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

func main() {
    var wg sync.WaitGroup // Создаем WaitGroup

    for i := 0; i < 10; i++ {
        wg.Add(1) // Увеличиваем счетчик горутин
        go func() {
            defer wg.Done() // Уменьшаем счетчик горутин после выполнения
            work()
        }()
    }

    wg.Wait() // Ожидаем, пока все горутин завершат работу
    fmt.Println("All work done!")
}

```

Результат:

```
PS C:\Users\1234\web-5> go run
done
done
done
done
done
done
done
done
done
done
done
All work done!
```

Рис.2 — Результат 2

3)Задание: 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.

Решение:

```
package main
```

```
import "fmt"
```

```
// Реализация функции calculator
```

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

```
    output := make(chan int) // Канал для возвращаемого результата
```

```
    go func() {
```

```

    defer close(output) // Закрываем канал

    select {
    case val := <-firstChan:
        output <- val * val // Если получили из первого канала квадрат
    case val := <-secondChan:
        output <- val * 3 // Если получили из второго канала умножение на 3
    case <-stopChan:
        // Если получили сигнал из stopChan, просто выходим
        return
    }
}

return output
}

func main() {
    // Каналы для тестирования
    firstChan := make(chan int)
    secondChan := make(chan int)
    stopChan := make(chan struct{})

    // Запуск функции calculator
    resultChan := calculator(firstChan, secondChan, stopChan)

    // Пример использования
    go func() {

        // Отправляем значение в первый канал для получения квадрата
        // firstChan <- 4

        // Отправляем значение во второй канал для умножения на 3
        secondChan <- 5

        // Можно также отправить сигнал завершения
        // close(stopChan)
    }()

    for result := range resultChan {
        fmt.Println(result)
    }
}

```

Результат:

```

PS C:\Users\1234\web-5>
15

```

Рис.3 - Результат 3

Заключение:

Я научился пользоваться каналами при работе с горутинами.