



**«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)_____

О т ч е т

по лабораторной работе №5

Название лабораторной работы: Основы асинхронного программирования
на Golang

Дисциплина: Языки интернет-программирования

Студент гр. ИУ6-33Б

(Подпись, дата)

О.С. Кашу
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В.Д. Шульман
(И.О. Фамилия)

Москва, 2024

1. ВВЕДЕНИЕ

1.1 Цель

Изучение основ асинхронного программирования с использованием языка Golang.

1.2 Задания

1.2.1 Задание 1

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

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. В конце закрыть канал

Функция должна называться `removeDuplicates()`

1.2.2

1.2.3

2. ХОД РАБОТЫ

2.1 Задание 1

Для выполнения данного задания нужно считывать в цикле канал *inputStream* и сравнивать полученное значение с предыдущим. Если они не равны, то новое значение отправляется в канал `outputStream`.

Ниже представлен листинг кода с тестирующей функцией

```
package main

import (
    "fmt"
    "time"
)

func removeDuplicates(inputStream chan string, outputStream chan string) {
    var a string
    for v := range inputStream {
        if a != v {
            outputStream <- v
        }
        a = v
    }
    close(outputStream)
}
```

```

}

func printer(c chan string) {
    for {
        msg := <-c
        fmt.Println(msg)
        time.Sleep(time.Second * 1)
    }
}

func main() {
    inputStream := make(chan string)
    outputStream := make(chan string)

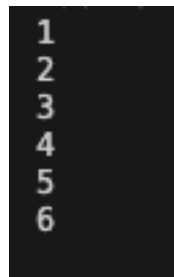
    go removeDuplicates(inputStream, outputStream)
    go printer(outputStream)

    for _, i := range "112334456" {
        inputStream <- string(i)
    }

    var input string
    fmt.Scanln(&input)
}

```

Ниже приведен вывод данной программы (рис. 1)



```

1
2
3
4
5
6

```

Рисунок 1. Вывод тестирующей программы в 1 задании

2.2 Задание 2

В данном задании используется *sync.WaitGroup* для синхронизации горутин и устанавливает блокировку, пока не завершит выполнение вся группа горутин.

Ниже приведен листинг кода

```

package main

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

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

func main() {
    wg := new(sync.WaitGroup)

```

```

    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func() {
            work()
            wg.Done()
        }()
    }

    wg.Wait()
}

```

Таким образом, выведется 10 раз слово “done”.

2.3 Задание 3

В данном задании внутри функции calculator нужно создать анонимную функцию и с помощью select-case проверять каналы.

Ниже представлен листинг кода

```

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 {
    output := make(chan int)
    go func() {
        defer close(output)
        select {
            case a := <-firstChan:
                output <- a * a
            case b := <-secondChan:
                output <- b * 3
            case <-stopChan:
                return
        }
    }()
    return output
}

func main() {
    ch1, ch2 := make(chan int), make(chan int)
    stop := make(chan struct{})

    r := calculator(ch1, ch2, stop)
    ch1 <- 4
    // ch2 <- 3
    close(stop)
    fmt.Println(<-r)
}

```

Вывод программы при отправке значения в первый канал 4 – 16.

3. ВЫВОД

Изучены возможности асинхронного программирования в Golang и использования каналов для связи между горутинами.