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 ИУ6-33Б _____ С.В. Сонин
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ В.Д. Шульман
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Цель работы:

Изучение основ асинхронного программирования с использованием языка Golang.

Задание 1:

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.

Код программы:

```
package main

import "fmt"

func calculator(firstChan <-chan int, secondChan <-chan int,
stopChan <-chan struct{}) <-chan int {
    answerChan := make(chan int)
    go func() {
        defer close(answerChan)
        select {
        case val := <-firstChan:
            answerChan <- val * 2
        case val := <-secondChan:
            answerChan <- val * 3
        case <-stopChan:
```

```
        }
    }()
    return answerChan
}
func main() {
    var firstChan = make(chan int, 1)
    var secondChan = make(chan int, 1)
    var stopChan = make(chan struct{}, 1)
    stopChan <- struct{}{}
    fmt.Println(<-calculator(firstChan, secondChan, stopChan))
}
```

```
X:\Git\web-5\projects\calculator>go run main.go
10
```

Рисунок 1 – Результат работы программы.

Задание 2:

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

Код программы:

```
package main

import "fmt"

// реализовать removeDuplicates(in, out chan string)

func removeDuplicates(in, out chan string) {
    defer close(out)
    var lastString string
    for i := range in {
        if i != lastString {
            out <- i
        }
        lastString = i
    }
}
```

```go
func main() {
    var in = make(chan string)
    var out = make(chan string)
    var str string
    fmt.Scan(&str)
    go removeDuplicates(in, out)
    go func() {
        defer close(in)
        for _, i := range str {
            in <- string(i)
        }
    }()
    for i := range out {
        fmt.Print(i)
    }

}
```

```
X:\Git\web-5\projects\pipeline>go run main.go
1111112222223333333344444555555666666669999
1234569
X:\Git\web-5\projects\pipeline>
```

Рисунок 2 – Результат работы программы.

Задание 3:

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

Код программы:

```go
    package main

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

```go
func work(wg *sync.WaitGroup) {
    defer wg.Done()
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}

func main() {
    wg := new(sync.WaitGroup)
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go work(wg)

    }
    wg.Wait()
}
```

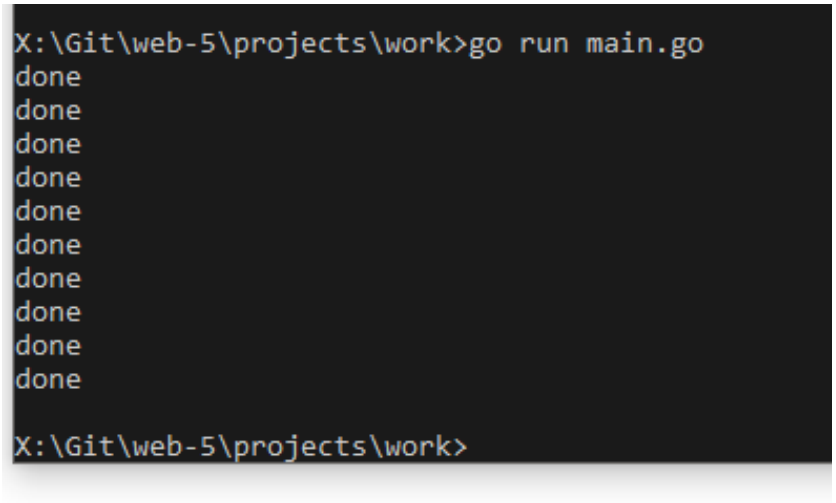

Рисунок 3 – Результат работы программы.

Вывод:

Изучил основы асинхронного программирования с использованием языка Golang.

Список использованных источников:

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

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