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 Программно-аппаратное обеспечение систем искусственного интеллекта**

О Т Ч Е Т

по лабораторной работе № 5

Название: **Основы Асинхронного программирования в Go**

Дисциплина: Языки интернет программирования

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

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

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

Ход работы:

1. Необходимо в отдельных горутинах вызвать функцию work() 10 раз и дождаться результатов выполнения вызванных функций.

   Далее представлены программный код и результат выполнения написанного теста (рисунок 1)

```go
package main

import (
    "fmt"
    “time”
      “sync”
)

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(wg *sync.WaitGroup) {
            work()
            defer wg.Done()
        }(wg)
    }
    wg.Wait()
    fmt.Println("Completed")
}
```

```
PS D:\Progects Go\zad1> go run "d:\Progects Go\zad1\laba5.go"
done
done
done
done
done
done
done
done
done
```

Рисунок 1- задача 1 код и вывод

2. Написать 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.

Далее представлены программный код и результат выполнения (рисунок 2)

```go
package main

import (
    "fmt"
)

func removeDuplicates(inputStream chan string, outputStream chan string) {
    f := " "
    for v := range inputStream {
        if v != f || f == " " {
            outputStream <- v
        }
        f = v
    }
    close(outputStream)
}

func main() {
    channel1 := make(chan string, 3)
    channel2 := make(chan string, 3)
    str1 := "qw"
    str2 := "qw"
    str3 := "jnkj"
    channel1 <- str1
    channel1 <- str2
    channel1 <- str3
    close(channel1)
    removeDuplicates(channel1, channel2)
    for v := range channel2 {
        fmt.Print(v, "; ")
    }

}
```

```
PS D:\Progects Go\zad1> go run "d:\Progects Go\zad1\laba5.go"
qw; jnkj;
PS D:\Progects Go\zad1>
```

Рисунок 2- задача 2 код и вывод

3. 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.

Далее представлены программный код и результат выполнения в postman (рисунки 3)

```go
package main

import (
    "fmt"
)


    func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
        out := make(chan int)
        go func() {
            defer close(out)
            select {
            case x := <-firstChan:
                out <- x * x
            case x := <-secondChan:
                out <- x * 3
            case <-stopChan:
                return
            }
        }()
        return out
    }

    func main() {
        channel1 := make(chan int)
        channel2 := make(chan int)
        stopchan := make(chan struct{})
        resalt := calculator(channel1, channel2, stopchan)
        //channel1 <- 4
        channel2 <- 2
        fmt.Println(<-resalt)
    }
```

```
PS D:\Progects Go\zad1> go run "d:\Progects Go\zad1\laba5.go"
16
6
PS D:\Progects Go\zad1> go run "d:\Progects Go\zad1\laba5.go"
6
```

Рисунок 3-задача 3 код и вывод

Заключение : в результате проделанной работы были получены теоретические знания и практические навыки основ асинхронного программирования с использованием языка Golang. Были решены 3 задачи и написаны проверки правильности выполнения программы.
Результаты выполнения работы были отправлены на GitHub