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: **Основы web-программирования**

Студент ИУ6-31Б _____ Минбулатов А.А.
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Шульман В.Д.
(Подпись, дата) (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".

2.Сделайте форк данного репозитория в GitHub, склонируйте получившуюся копию локально, создайте от мастера ветку дев 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 дев 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

Задачи:

1. 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!

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

work() ничего не принимает и не возвращает. Пакет "sync" уже импортирован.

3. 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 задания calculator представлен на рисунке 1:

ts > calculator > main.go > calculator

```go
package main

import (
    "fmt"
    "time"
)

// 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 {
    ch := make(chan int)
    var val int

    go func() {
        defer close(ch)
        select {
        case val = <-firstChan:
            ch <- val * val
        case val = <-secondChan:
            ch <- val * 3
        case <-stopChan:
            return
        }
    }()
    return ch
}
func main() {

    firstChan := make(chan int)
    secondChan := make(chan int)
    stopChan := make(chan struct{})

    outputChan := calculator(firstChan, secondChan, stopChan)

    go func() {
        time.Sleep(5 * time.Second)
        close(stopChan)
    }()

    go func() {
        firstChan <- 5
        secondChan <- 10
        firstChan <- 21
        secondChan <- 13
    }()

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

Рис.1—Код программы calculator

Тест задания представлен на рисунке 2.

```
user@testcomp:~/Documents/lab5/web-5/projects/calculator$ go run main.go
25
user@testcomp:~/Documents/lab5/web-5/projects/calculator$
```

Рис. 2 — Результат работы

Код задания work:

```go
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(wg *sync.WaitGroup) {
            defer wg.Done()
            work()
        }(wg)
    }

    wg.Wait()
}
```

Рис.3 — Код программы work

Результат work представлен на рисунке 4.

```
said@said-V15x-V17xRNx:~/projects/lab5/lab5-3$ go run main.go
карамбола
карамбола
карамбола
карамбола
карамбола
карамбола
карамбола
карамбола
карамбола
карамбола
```

Рис.4 — Результат работы

Код задания pipeline:

```go
package main

import "fmt"

// реализовать removeDuplicates(in, out chan string)
func removeDuplicates(inputStream chan string, outputStream chan string) {
    str := ""
    for v := range inputStream {
        if str == "" {
            str = v
            outputStream <- v
        } else if str != v {
            str = v
            outputStream <- v
        } else if str == v {
            continue
        }
    }
    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 func() {
        inputStream <- "p"
        inputStream <- "b"
        inputStream <- "b"
        inputStream <- "r"
        close(inputStream)
    }()

    go removeDuplicates(inputStream, outputStream)

    for value := range outputStream {
        fmt.Println(value)
    }
}
```

Рис.5 — Код программы pipeline

Результат работы pipeline на рисунке 6.

```
said@said-V15x-V17xRNx:~/projects/lab5/lab5-1$ go run main.go
12321
```

Рис.6 — Результат работы

Вывод: асинхронность Golang помогает выполнять параллельные задачи, а также вычислять результат для входного потока данных.