



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

ИУ6-31БВ

Бокова В.О.

(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.

Порядок выполнения

1. Ознакомьтесь с разделом "3. Map, файлы, интерфейсы, многопоточность и многое другое" курса <https://stepik.org/course/54403/info>
2. Сделайте форк данного репозитория в GitHub, склонируйте получившуюся копию локально, создайте от мастера ветку dev 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 dev 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

Ход работы

А) Задание «Calculator» (рис 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.

Рисунок 1

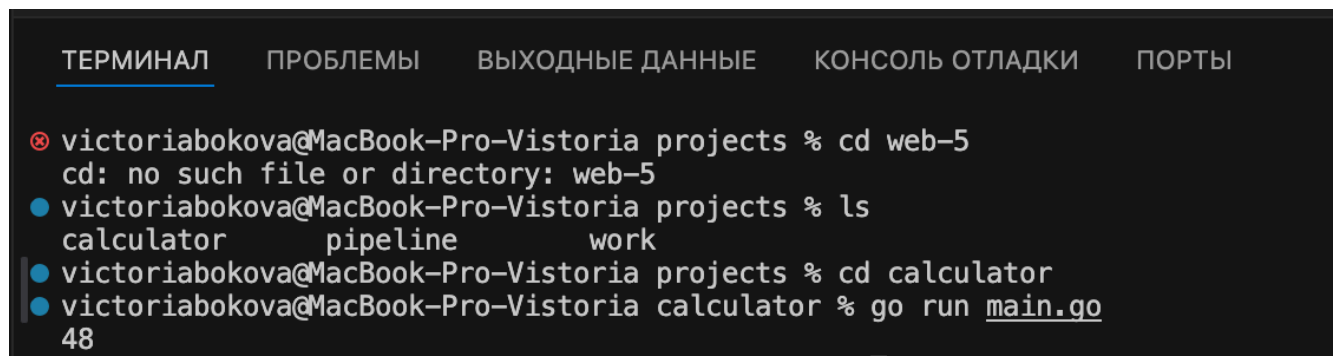
Решение:

```
package main import
"fmt"
// calculator - функция, которая принимает три канала:
// firstChan, secondChan, stopChan, и возвращает канал resChan типа <-chan int.
// Она запускает горутину, которая обрабатывает значения из каналов.
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    resChan := make(chan int) // Создаем канал для результата go
    func() { // Запускаем анонимную горутину
        defer close(resChan) // Закрываем канал при завершении //
        select позволяет ожидать значения из любого из каналов
        select {
            // Получение значения из firstChan case val := <-
            firstChan: // Получаем значение из канала
                resChan <- val * val // Квадрат значения и отправка в resChan
```

```

// Получение значения из secondChan case val := <-
secondChan: // Получаем значение из канала
    resChan <- val * 3 // Умножение значения на 3 и отправка в resChan
// Получение сигнала из stopChan case <-
stopChan: // Получаем сигнал из канала
    return // Завершаем горутину
}
}()
return resChan // Возвращаем канал результата
}
func main() {
    ch1 := make(chan int)      // Создаем канал ch1 ch2 := make(chan int)      // Создаем
    канал ch2 ch3 := make(chan struct{}) // Создаем канал ch3 (для сигнала остановки) res
    := calculator(ch1, ch2, ch3) // Вызываем функцию calculator и получаем канал res ch2 <-
    16 // Отправляем значение 16 в ch2 fmt.Println(<-res) // Получаем результат из канала
    res и выводим его
}

```



```

ТЕРМИНАЛ  ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ПОРТЫ

❌ victoriabokova@MacBook-Pro-Vistoria projects % cd web-5
cd: no such file or directory: web-5
● victoriabokova@MacBook-Pro-Vistoria projects % ls
calculator      pipeline        work
● victoriabokova@MacBook-Pro-Vistoria projects % cd calculator
● victoriabokova@MacBook-Pro-Vistoria calculator % go run main.go
48

```

Рисунок 2 – вывод результата

В) Задача «Pipeline» (рис 3)

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()`

Рисунок 3

Решение: package
main

```

import "fmt" func removeDuplicates(inputStream <-chan string, outputStream
chan<- string) {
    defer close(outputStream) // Закрываем выходной канал при завершении
    var prevValue string // Храним предыдущее значение for value := range
    inputStream {
        if value != prevValue { // Проверка на дубликат

```

```

        outputStream <- value // Отправка в выходной канал, если значение не дубликат
        prevValue = value      // Обновляем предыдущее значение
    }
}
}
func main() {
    in := make(chan string) out
    := make(chan string) go
    func() {
        defer close(in) // Закрываем входной канал при завершении in
        <- "one" in <- "two" in <- "two" in <- "three" in <- "three" in <-
        "three" in <- "four" in <- "five"
    }()
    go removeDuplicates(in, out) for
    value := range out {
        fmt.Println(value)
    }
}

```

```

ТЕРМИНАЛ  ПРОБЛЕМЫ  1  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ПОРТЫ
● victoriabokova@MacBook-Pro-Vistoria projects % cd pipeline
● victoriabokova@MacBook-Pro-Vistoria pipeline % go run main.go
one
two
three
four
five

```

Рисунок 4 – результат программы

C) Задача «Work» (рис 5):

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

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

Рисунок 5

Решение:

```

package main

import (
    "fmt"
    "sync"
)

func work() {

```

```
// Здесь ваш код, который выполняется в каждой горутине
fmt.Println("WORK!")
}

func main() {
var wg sync.WaitGroup // Создаем WaitGroup для синхронизации
wg.Add(10) // Увеличиваем счетчик на 10, так как запускаем 10 горутин

for i := 0; i < 10; i++ {
go func() {
defer wg.Done() // Уменьшаем счетчик после завершения горутин
work() // Вызываем функцию work
}() }

wg.Wait() // Ждем завершения всех горутин
fmt.Println("Все горутин завершили!")
}
```

```

ТЕРМИНАЛ  ПРОБЛЕМЫ  1  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ПОРТЫ
victoriabokova@MacBook-Pro-Vistoria pipeline % cd ../
victoriabokova@MacBook-Pro-Vistoria projects % cd work
victoriabokova@MacBook-Pro-Vistoria work % go run main.go
WORK!
WORK!
WORK!
WORK!
WORK!
WORK!
WORK!
WORK!
WORK!
WORK!
Все горутин завершили!
victoriabokova@MacBook-Pro-Vistoria work %

```

Рисунок 6 - тестирование

Заключение: В процессе выполнения лабораторной работы были изучены основы асинхронного программирования на Golang, а также получены практические навыки написания программ с использованием данной концепции программирования.

Использованные источники

- <https://github.com/ValeryBMSTU/web-5>
- <https://stepik.org/course/54403/info>