|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 0**9.03.01 Компьютерные системы и сети**

**Отчет**

**по лабораторной работе № 5**

**Название: Основы асинхронного программирования на Golang**

**Дисциплина: Языки интернет программирования**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-33Б |  |  | Д.А. Лазутин |
|  | (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.

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {

ret := make(chan int)

go func(ret chan int) {

defer close(ret)

select {

case f := <-firstChan:

ret <- f \* f

case s := <-secondChan:

ret <- s \* 3

case <-stopChan:

return

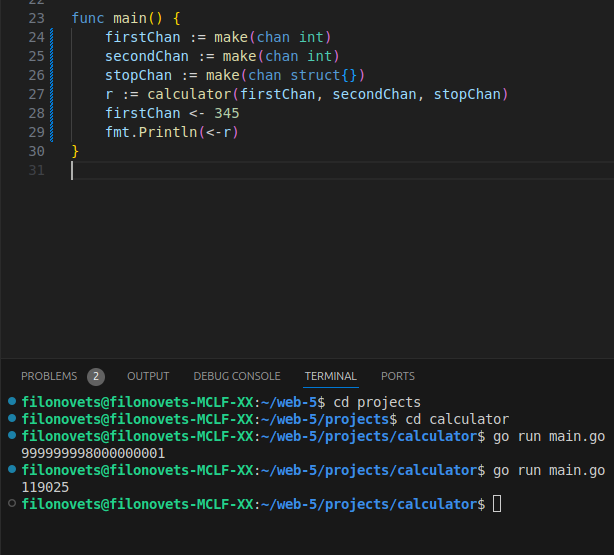
}

}(ret)

return ret

}

Тестирование. (рис 1)

Рисунок 1 — результаты тестирования.

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.

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!

Результаты тестирования приведены на рисунке 2.

func removeDuplicates(in, out chan string) {

go func(in, out chan string) {

temp := ""

for {

select {

case val, isOpen := <-in:

if isOpen {

if val != temp {

temp = val

out <- val

}

} else {

close(out)

return

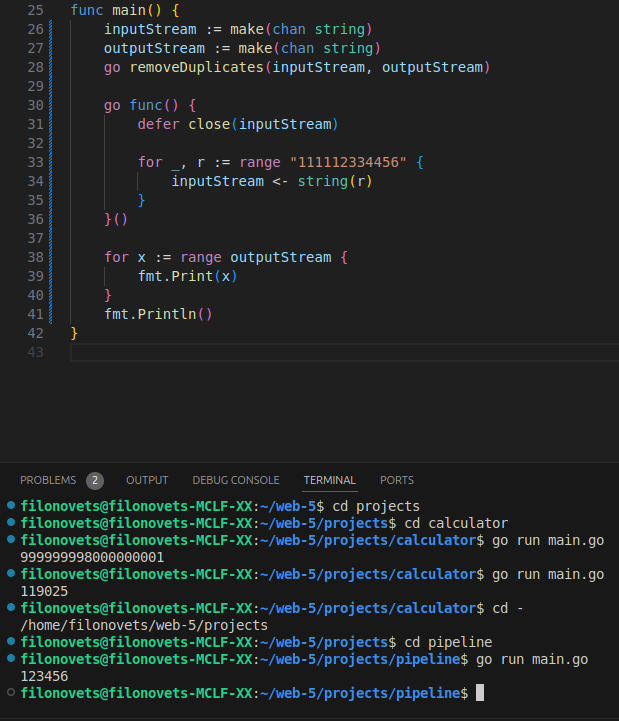
}

}

}

}(in, out)

}

Рисунок 2 — результаты тестирования.

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.

Функция work() ничего не принимает и не возвращает.

package main

import (

"fmt"

"time"

"sync"

)

func work() {

time.Sleep(time.Millisecond \* 50)

fmt.Println("done")

}

func main() {

var wg sync.WaitGroup

wg.Add(10)

for i:=0; i<10; i++{

go func(){

defer wg.Done()

work()

}()

}

wg.Wait()

}

Вывод. В ходе работы были изучены методы асинхронного программирования на примере выполнения трех работ.