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	28.09.24	Пономаренко В.М.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		28.09.24	Шульман В.Д.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель:
изучение основ асинхронного программирования с использованием языка GoLang

Задание:
Решить три задач на тему асинхронного программирования на языке GoLang

Ход работы:

1. Задача 1:
   Условие:
   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
   -в случае, если аргумент будет получен из канала stopChan, нужно просто завершить работу функции.
   Функция calculator должна быть неблокирующей, сразу возвращая управление.

   Листинг:

```go
package main

import (
    "fmt"
)

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    ans := make(chan int) //создание возвращаемого канала

    // чтобы сделать функцию неблокирующей, убираем
    // в  анонимную функцию, вызываемую через горутину
    // логику select т.к. в противном случае выполнение функции блокируется до момента поступления значения
    go func() {
        defer close(ans) //
        select {
        case a := <-firstChan:
            ans <- a * a
        case a := <-secondChan:
            ans <- a * 3
        case <-stopChan:
            return
        }
    }()
    return ans
}
```

Рисунок 1 - листинг функции calculator

```go
func main() {
    first := make(chan int)
    second := make(chan int)
    stop := make(chan struct{})
    var a, b int
    fmt.Println("Put 1 for square, 2 for multiplication by 3 or 3 for stop:")
    fmt.Scan(&a)
    if a == 1 || a == 2 {
        fmt.Println("Put a number")
        fmt.Scan(&b)
    }
    var ansChan <-chan int
    switch a {
    case 1:
        ansChan = calculator(first, second, stop)
        first <- b

    case 2:3
        ansChan = calculator(first, second, stop)
        second <- b

    case 3:
        close(stop)                              // Закрываем канал и сигнализируем остановку
        ansChan = calculator(first, second, stop) // Запускаем функцию для остановки
    }
    ans, ok := <-ansChan // Получаем результат и статус
    if ok {
        fmt.Println("Result is: ", ans)
    } else {
        fmt.Println("No result, calculator was stopped.")
    }

}

```

Рисунок 2 - листинг тестирующей программы

Результаты работы:

```
Put 1 for square, 2 for multiplication by 3 or 3 for stop:
2
Put a number
23
Result is:  69
```

Рисунок 3 - результат 1

```
1
Put a number
10
Result is:  100
```

Рисунок 4 - результат 2

```
Put 1 for square, 2 for multiplication by 3 or 3 for stop:
3
No result, calculator was stopped.
```

Рисунок 5 - результат 3

2. Задача 2 **pipline**:

Условие:

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

Листинг:

```go
package main

import (
    "bufio"
    "fmt"
    "os"
)

func removeDuplicates(inputStream chan string, outputStream chan string) {
    prev := ""
    for v := range inputStream {
        if prev == "" || v != prev {
            outputStream <- v
        }
        prev = v
    }
    close(outputStream)// закрываем канал вывода, т.к.используем буферизированные каналы
    // если не закрыть, то range в main никогда не завершится (блокировка канала)
}

func main() {
    inputStream := make(chan string, 5)
    outputStream := make(chan string, 5)
    //для теста создаем каналы с типом строка
    // с размером буфера 5
    s := bufio.NewScanner(os.Stdin)
    //заполнение канала через ввод с клавиатуры
    fmt.Printf("Put %d strings\n", cap(inputStream))
    for i := 0; i < cap(inputStream); i++ {
        s.Scan()
        a := s.Text()
        inputStream <- a
    }
    close(inputStream) // закрываем канал ввода
    // т.к. removeDuplicates может ожидать новые данные из канала и будет заблокирована до момента поступления, которого нет..
    removeDuplicates(inputStream, outputStream)
    fmt.Printf("In outputStream: ")
    for v := range outputStream {
        fmt.Print(v, ";")
    }

}
```

Рисунок 6 - листинг задачи 2

Результаты работы:

```
Put 5 strings
hello world
hi
q
q
er
In outputStream: hello world;hi;q ;er;
```

Рисунок 7 - результат 1

```
Put 5 strings
мке
привет мир
привет мир
ты
ты
In outputStream: мке ;привет мир;ты;
```

Рисунок 8 - результат 2

3. Задача 3 **work**:
   Условие: Внутри функции main необходимо в отдельных горутинах вызвать функцию work() 10 раз и дождаться результатов выполнения вызванных функций.

   Листинг:

```go
package main

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

func work() {
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}

func main() {
    wait := new(sync.WaitGroup) // определяем группу горутин для совместного выполнения
    for i := 0; i < 10; i++ {
        wait.Add(1) // увеличиваем счетчик внутренних активных элементов
        // анонимная горутина
        go func(w *sync.WaitGroup) {
            defer w.Done() // отложенный сигнал о завршершении горутины
            work()
        }(wait)
    }
    wait.Wait() // ожидание выполнения всех горутин, блокировка main
    fmt.Println("Completed")
}

```

Рисунок 9 - листинг задачи 3

Результат работы:

Рисунок 10 - результат 1

Заключение:
В результате проделанной работы были изучены основы асинхронного программирования на GoLang, а также необходимые для этого инструменты языка, такие как горутины, анонимные горутины, каналы, синхронизация горутин, базовые элементы пакета “sync”. Также были решены представленные задачи.

Список используемых источников:

1) https://stepik.org/course/54403/syllabus
2) https://habr.com/ru/articles/337528/