|  |  |
| --- | --- |
| 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)**

**Отчет**

**по лабораторной работе № 5**

**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | А. В. Тимохин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В. Д. Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

Цель работы: ознакомиться с такими темами в программировании на Go, как интерфейс, JSON, параллелизм, горутина, канал и др. Научиться применять их на практике и решить несколько заданий с применением полученных знаний.

Задание 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 ;)

Код программы:

package main

import (

    "bufio"

    "fmt"

    "os"

    "strings"

)

func f(inputStream <-chan string) <-chan string {

    out := make(chan string)

    go func() {

        defer close(out)

        var prev string

        first := true

        for el := range inputStream {

            if first || prev != el {

                out <- el

                prev = el

                first = false

            }

        }

    }()

    return out

}

func main() {

    in := make(chan string)

    go func() {

        defer close(in)

        reader := bufio.NewReader(os.Stdin)

        for {

            fmt.Print("Введите строку (пустая строка для завершения ввода): ")

            input, err := reader.ReadString('\n')

            if err != nil {

                fmt.Fprintln(os.Stderr, "Ошибка чтения ввода:", err)

                return

            }

            input = strings.TrimSpace(input)

            if input == "" {

                return

            }

            in <- input

        }

    }()

    answer := f(in)

    var outputs []string

    for j := range answer {

        outputs = append(outputs, j)

    }

    fmt.Println("\nРезультат обработки:")

    for \_, line := range outputs {

        fmt.Println(line)

    }

}

Результат работы программы:

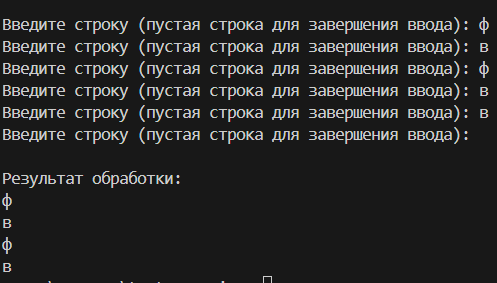


Рисунок 2 – результат работы программы

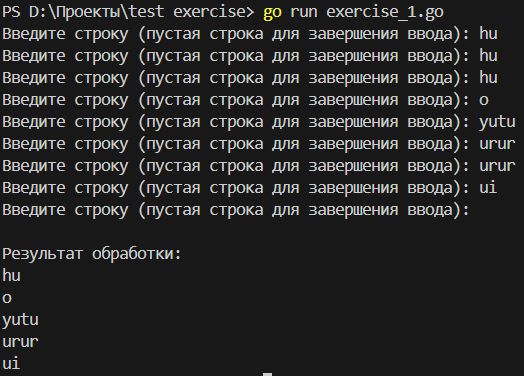


Рисунок 3 – результат работы программы

Задание 2:

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

Код программы:

package main

import (

    "fmt"

    "sync"

    "time"

)

func work(wg \*sync.WaitGroup) {

    defer wg.Done()

    fmt.Println("hello")

    time.Sleep(1 \* time.Second)

    fmt.Println("bye")

}

func main() {

    wg := new(sync.WaitGroup)

    for i := 0; i < 10; i++ {

        wg.Add(1)

        go work(wg)

    }

    wg.Wait()

    fmt.Println("end")

}

Результат работы программы:

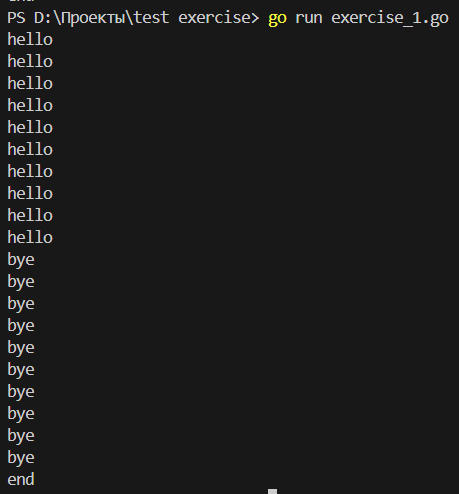


Рисунок 4 – результат работы программы

Задание 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 программы:

package main

import (

    "fmt"

)

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

    outChan := make(chan int)

    go func() {

        defer close(outChan)

        select {

        case val := <-firstChan:

            outChan <- val \* val

        case val := <-secondChan:

            outChan <- val \* 3

        case <-stopChan:

            return

        }

    }()

    return outChan

}

func main() {

    first := make(chan int)

    second := make(chan int)

    stop := make(chan struct{})

    result := calculator(first, second, stop)

    go func() {

        first <- 5

        close(first)

    }()

    if res, ok := <-result; ok {

        fmt.Printf("Квадрат числа: %d\n", res)

    }

    first2 := make(chan int)

    second2 := make(chan int)

    stop2 := make(chan struct{})

    result2 := calculator(first2, second2, stop2)

    go func() {

        second2 <- 7

        close(second2)

    }()

    if res, ok := <-result2; ok {

        fmt.Printf("Значение, умноженное на 3: %d\n", res)

    }

    first3 := make(chan int)

    second3 := make(chan int)

    stop3 := make(chan struct{})

    result3 := calculator(first3, second3, stop3)

    go func() {

        close(stop3)

    }()

    if res, ok := <-result3; ok {

        fmt.Printf("Результат: %d\n", res)

    } else {

        fmt.Println("Работа функции завершена без отправки значений.")

    }

}

Результат работы программы:

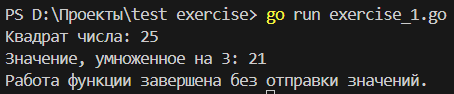


Рисунок 5 – результат работы программы

Вывод: я ознакомился с такими темами в программировании на Go, как интерфейс, JSON, параллелизм, горутина, канал и др. Научился применять их на практике и решил несколько заданий с применением полученных знаний.

Список источников:

https://stepik.org