



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

ИУ6-31Б
(Группа)

(Подпись, дата)

О.В. Белякова
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В.Д. Шульман
(И.О. Фамилия)

Москва, 2024

Цель работы – изучение основ асинхронного программирования с использованием языка Golang.

Порядок выполнения:

1. Ознакомиться с разделом 3 курса Stepik
2. Сделать форк репозитория с лабораторной работой
3. Выполнить задания в директории projects
4. Сделать отчет
5. Зафиксировать изменения и отправить изменения
6. Сделать Pull Request

Выполнение:

Задание 1 – pipeline (рис. 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!

Рисунок 1 – текст задания pipeline

Выполненное задание (рис. 2,3)

```

1 package main
2
3 import (
4     "fmt"
5     "reflect"
6 )
7
8 func removeDuplicates(inputStream <-chan string, outputStream chan<- string) {
9     defer close(outputStream)
10
11     var lastVal string
12     firstIteration := true
13
14     for val := range inputStream {
15         if firstIteration || val != lastVal {
16             outputStream <- val
17             lastVal = val
18             firstIteration = false
19         }
20     }
21 }

```

Рисунок 2 – код программы задание 1

```

23 func main() {
24     input := make(chan string, 5)
25     output := make(chan string, 5)
26
27     // Заполняем входной канал
28     go func() {
29         defer close(input)
30         input <- "a"
31         input <- "a"
32         input <- "b"
33         input <- "b"
34         input <- "c"
35     }()
36
37     go removeDuplicates(input, output)
38
39     var results []string
40     for val := range output {
41         results = append(results, val)
42     }
43
44     expected := []string{"a", "b", "c"}
45     if reflect.DeepEqual(results, expected) {
46         fmt.Println("Expected: ", expected, "Got: ", results)
47     }
48 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● admin1@Ubuntu:~/lab-5-bmstu$ go run projects/pipeline/main.go
Expected: [a b c] Got: [a b c]
○ admin1@Ubuntu:~/lab-5-bmstu$

```

Рисунок 3 – пример выполнения

Задание 2 – work (рис. 4)

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

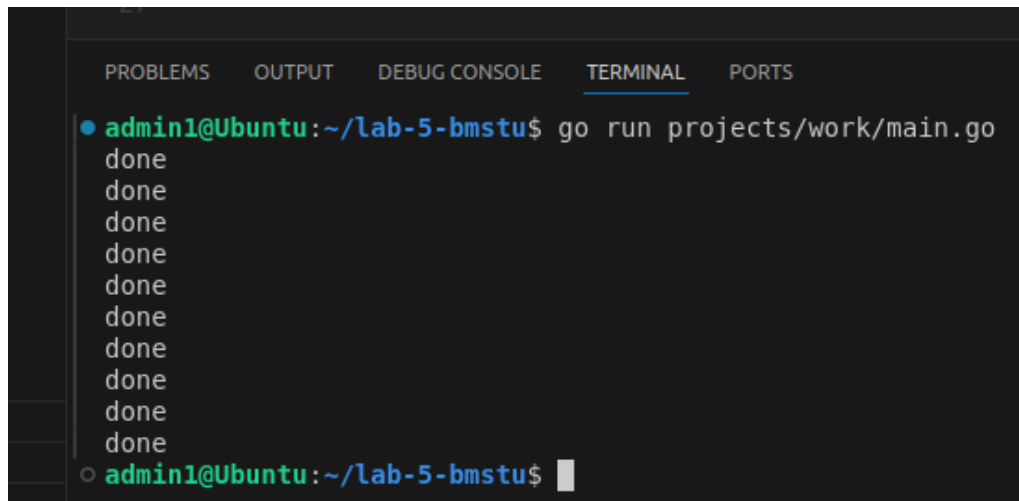
Функция `work()` ничего не принимает и не возвращает. Пакет `"sync"` уже импортирован.

Рисунок 4 – текст задания `work`

Выполнение задания (рис. 5,6)

```
1  package main
2
3  import (
4      "fmt"
5      "sync"
6      "time"
7  )
8
9  func work() {
10     time.Sleep(time.Millisecond * 50)
11     fmt.Println("done")
12 }
13
14 func main() {
15     var wg sync.WaitGroup
16     const workersCount = 10
17
18     for i := 0; i < workersCount; i++ {
19         wg.Add(1)
20         go func() {
21             defer wg.Done()
22             work()
23         }()
24     }
25     wg.Wait()
26 }
```

Рисунок 5 – код программы



```
admin1@Ubuntu:~/lab-5-bmstu$ go run projects/work/main.go
done
done
done
done
done
done
done
done
done
done
done
admin1@Ubuntu:~/lab-5-bmstu$
```

Рисунок 6 – пример выполнения

Задание 3 – calculator (рис. 7)

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 7 – текст задания

Выполнение задания (рис. 8,9)

```

projects > calculator > -o main.go > main
1  package main
2
3  import (
4      "fmt"
5      "time"
6  )
7
8  func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
9      resultChan := make(chan int)
10
11      go func() {
12          defer close(resultChan)
13
14          select {
15              case val := <-firstChan:
16                  resultChan <- val * val
17              case val := <-secondChan:
18                  resultChan <- val * 3
19              case <-stopChan:
20                  return
21          }
22      }()
23
24      return resultChan
25  }
26

```

Рисунок 8 – код программы

```

27  func main() {
28      firstChan := make(chan int, 1)
29      secondChan := make(chan int, 1)
30      stopChan := make(chan struct{})
31      resultChan := calculator(firstChan, secondChan, stopChan)
32
33      firstChan <- 4
34      select {
35          case result := <-resultChan:
36              if result == 16 {
37                  fmt.Println("Expected 16, got: ", result)
38              }
39          case <-time.After(time.Second):
40              fmt.Println("Timeout waiting for result")
41      }
42
43      secondChan <- 5
44      resultChan = calculator(firstChan, secondChan, stopChan)
45      select {
46          case result := <-resultChan:
47              if result == 15 {
48                  fmt.Println("Expected 15, got: ", result)
49              }
50          case <-time.After(time.Second):
51              fmt.Println("Timeout waiting for result")
52      }
53  }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

• admin1@Ubuntu:~/lab-5-bmstu$ go run projects/calculator/main.go
Expected 16, got: 16
Expected 15, got: 15
○ admin1@Ubuntu:~/lab-5-bmstu$

```

Рисунок 9 – пример выполнения

Заключение:

Выполнили задания, связанные с асинхронным программированием на Golang.