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 «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 1

по курсу «Компьютерные сети»

«Простейший протокол прикладного уровня»

Студент группы ИУ9-31Б Горбунов А. Д.

Преподаватель Посевин Д. П.

Москва 2023

1 Задание

Протокол многократного поиска подстрок в строке(вариант 2)

2 Результаты

Исходный код программы представлен в листинге 1, 2, 3, 4, 5

Листинг 1 — client.go

```
package main
import (
  "encoding/json"
  "flag"
  "fmt"
  "net"
  "proto"
  "strconv"
  "github.com/skorobogatov/input"
)
func interact(conn *net.TCPConn) {
  defer conn.Close()
  encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
  var stro proto.Stroca

  for {
    fmt.Printf("command = ")
    command := input.Gets()
    switch command {
    case "quit":
      send_request(encoder, "quit", nil)
      return
    case "stroca":
      fmt.Printf("str: ")
      fmt.Scanf("%s", &(stro.Stroca))
      send_request(encoder, "stroca", &stro)
    case "substring":
      if len(stro.Stroca) == 0 {
        fmt.Printf("error: stroca is not set\n")
        continue
      }
      var str string
      fmt.Printf("substring = ")
      fmt.Scanf("%s", &str)
      send_request(encoder, "substring", &str)
    default:
      fmt.Printf("error: unknown command\n")
      continue
    }
```

Листинг 2 — client.go(продолжение)

```
        var resp proto.Response
    if err := decoder.Decode(&resp); err != nil {
      fmt.Printf("error: %v\n", err)
      break
    }
    switch resp.Status {
    case "ok":
      fmt.Printf("ok\n")
    case "failed":
      if resp.Data == nil {
        fmt.Printf("error: data field is absent in response\n")
      } else {
        var errorMsg string
        if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          fmt.Printf("failed: %s\n", errorMsg)
        }
      }
    case "result":
      if resp.Data == nil {
        fmt.Printf("error: data field is absent in response\n")
      } else {
        var resultInsex []int
        var indexString string
        if err := json.Unmarshal(*resp.Data, &resultInsex); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          for i, num := range resultInsex {
            if i > 0 {
              indexString += ","
            }
            indexString += strconv.Itoa(num)
          }
          fmt.Printf("result = %s\n", indexString)
        }
      }
    default:
      fmt.Printf("error: server reports unknown status %q\n", resp.
    Status)
    }
  }
}
func send_request(encoder *json.Encoder, command string, data interface
    {}) {
  var raw json.RawMessage
  raw, _ = json.Marshal(data)
  encoder.Encode(&proto.Request{Command: command, Data: &raw})
}
func main() {
  var addrStr string
  flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address
     and port")
  flag.Parse()
  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    fmt.Printf("error: %v\n", err)
  } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
    fmt.Printf("error: %v\n", err)
  } else {
    interact(conn)
  }
}
```

Листинг 3 — proto.go

```
1  package proto
2  import "encoding/json"
3
4  type Request struct {
5    Command string `json:"command"`
6    Data *json.RawMessage `json:"data"`
7  }
8
9  type Response struct {
10   Status string `json:"status"`
11   Data *json.RawMessage `json:"data"`
12 }
13
14 type Stroca struct {
15   Stroca string `json:"stroca"`
16 }
```

Листинг 4 — server.go

```
1  package main
2  import (
3    "encoding/json"
4    "flag"
5    "fmt"
6    "net"
7    "proto"
8    "strings"
9    log "github.com/mgutz/logxi/v1"
10 )
11
12 func evaluatePolynomial(str string, substr string) []int {
13   indices := []int{}
14   startIndex := 0
15   for {
16     index := strings.Index(str[startIndex:], substr)
17     if index == -1 {
18       break
19     }
20     indices = append(indices, startIndex+index)
21     startIndex += index + len(substr)
22   }
23
24   return indices
25 }
26
27 type Client struct {
28   logger log.Logger
29   conn   *net.TCPConn
30   enc    *json.Encoder
31   stro   proto.Stroca
32 }
33
34 func NewClient(conn *net.TCPConn) *Client {
35   return &Client{
36     logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())
       ),
37     conn:   conn,
38     enc:    json.NewEncoder(conn),
39     stro:   proto.Stroca{Stroca: ""},
40   }
41 }
42
43 func (client *Client) serve() {
44   defer client.conn.Close()
45   decoder := json.NewDecoder(client.conn)
46   for {
47     var req proto.Request
48     if err := decoder.Decode(&req); err != nil {
49       client.logger.Error("cannot decode message", "reason", err)
50       break
51     } else {
52       client.logger.Info("received command", "command", req.Command)
53       if client.handleRequest(&req) {
54         client.logger.Info("shutting down connection")
55         break
56       }
57     }
58   }
59 }
```

Листинг 5 — server.go(продолжение)

```
func (client *Client) respond(status string, data interface{}) {
  var raw json.RawMessage
  raw, _ = json.Marshal(data)
  client.enc.Encode(&proto.Response{Status: status, Data: &raw})
}

func main() {
  var addrStr string
  flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address
    and port")
  flag.Parse()
  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    log.Error("address resolution failed", "address", addrStr)
  } else {
    log.Info("resolved TCP address", "address", addr.String())
    if listener, err := net.ListenTCP("tcp", addr); err != nil {
      log.Error("listening failed", "reason", err)
    } else {
      for {
        if conn, err := listener.AcceptTCP(); err != nil {
          log.Error("cannot accept connection", "reason", err)
        } else {
          log.Info("accepted connection", "address", conn.RemoteAddr().
    String())
          go NewClient(conn).serve()
        }
      }
    }
  }
}
```

Результат запуска представлен на рисунке 1, 2

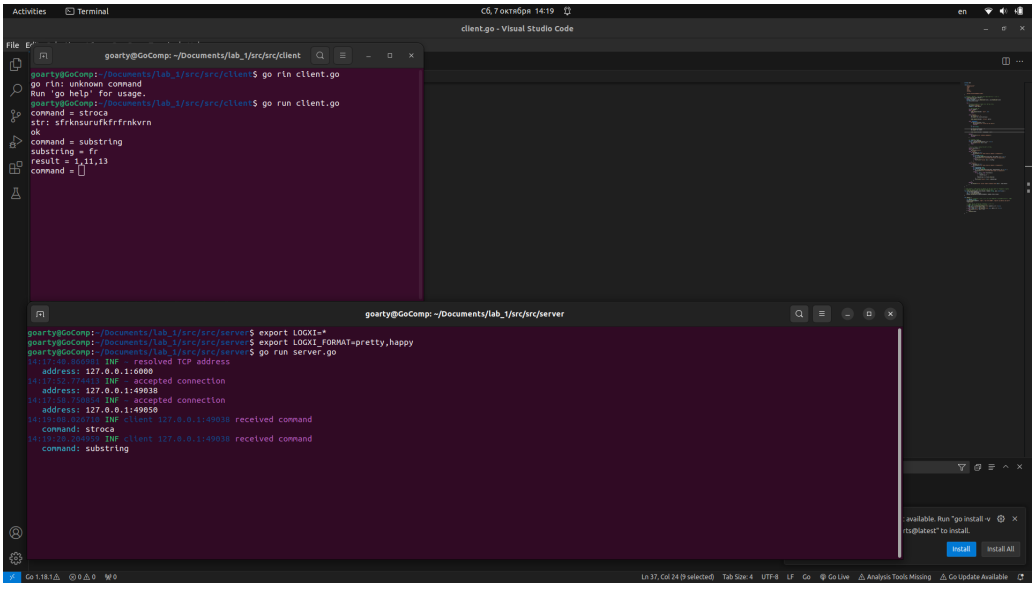

Рис. 1 — Выполнения протокола при 1-м пользователе

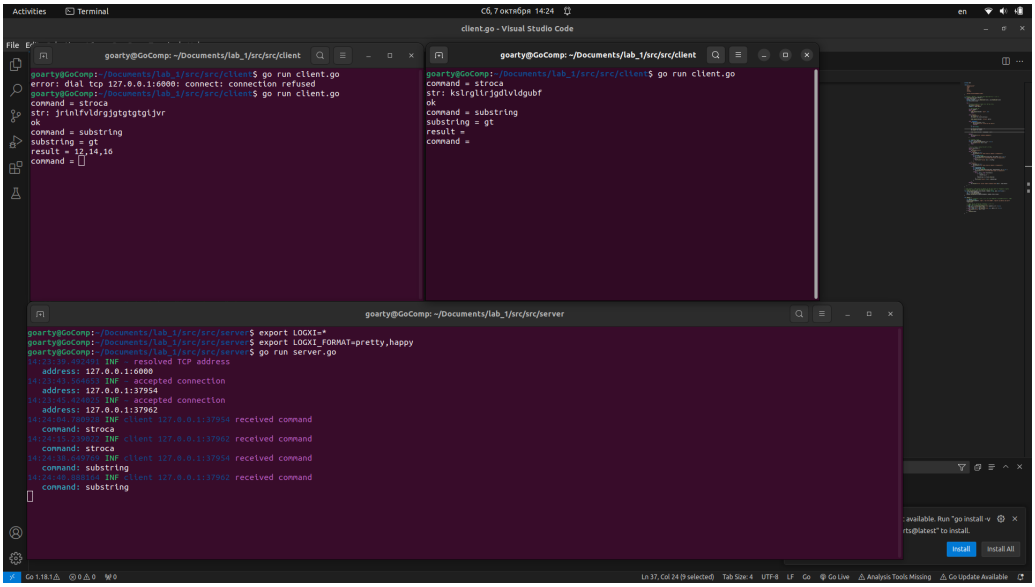

Рис. 2 — Выполнения протокола при 2-х пользователях