



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

```
1 package main
2 import (
3     "encoding/json"
4     "flag"
5     "fmt"
6     "net"
7     "proto"
8     "strconv"
9     "github.com/skorobogatov/input"
10 )
11 func interact(conn *net.TCPConn) {
12     defer conn.Close()
13     encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
14     var stro proto.Stroca
15
16     for {
17         fmt.Printf("command = ")
18         command := input.Get()
19         switch command {
20             case "quit":
21                 send_request(encoder, "quit", nil)
22                 return
23             case "stroca":
24                 fmt.Printf("str: ")
25                 fmt.Scanf("%s", &(stro.Stroca))
26                 send_request(encoder, "stroca", &stro)
27             case "substring":
28                 if len(stro.Stroca) == 0 {
29                     fmt.Printf("error: stroca is not set\n")
30                     continue
31                 }
32                 var str string
33                 fmt.Printf("substring = ")
34                 fmt.Scanf("%s", &str)
35                 send_request(encoder, "substring", &str)
36             default:
37                 fmt.Printf("error: unknown command\n")
38                 continue
39         }
40     }
41 }
```

Листинг 2 — client.go(продолжение)

```

1      var resp proto.Response
2      if err := decoder.Decode(&resp); err != nil {
3          fmt.Printf("error: %v\n", err)
4          break
5      }
6      switch resp.Status {
7      case "ok":
8          fmt.Printf("ok\n")
9      case "failed":
10         if resp.Data == nil {
11             fmt.Printf("error: data field is absent in response\n")
12         } else {
13             var errorMsg string
14             if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
15                 fmt.Printf("error: malformed data field in response\n")
16             } else {
17                 fmt.Printf("failed: %s\n", errorMsg)
18             }
19         }
20     case "result":
21         if resp.Data == nil {
22             fmt.Printf("error: data field is absent in response\n")
23         } else {
24             var resultInsex []int
25             var indexString string
26             if err := json.Unmarshal(*resp.Data, &resultInsex); err != nil {
27                 fmt.Printf("error: malformed data field in response\n")
28             } else {
29                 for i, num := range resultInsex {
30                     if i > 0 {
31                         indexString += ","
32                     }
33                     indexString += strconv.Itoa(num)
34                 }
35                 fmt.Printf("result = %s\n", indexString)
36             }
37         }
38     default:
39         fmt.Printf("error: server reports unknown status %q\n", resp.
40             Status)
41     }
42 }
43 func send_request(encoder *json.Encoder, command string, data interface
44     {}) {
45     var raw json.RawMessage
46     raw, _ = json.Marshal(data)
47     encoder.Encode(&proto.Request{Command: command, Data: &raw})
48 }
49 func main() {
50     var addrStr string
51     flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address
52         and port")
53     flag.Parse()
54     if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
55         fmt.Printf("error: %v\n", err)
56     } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
57         fmt.Printf("error: %v\n", err)
58     } else {
59         interact(conn)
60     }
61 }

```

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             ),
38         conn:   conn,
39         enc:    json.NewEncoder(conn),
40         stro:   proto.Stroca{Stroca: ""},
41     }
42 }
43
44 func (client *Client) serve() {
45     defer client.conn.Close()
46     decoder := json.NewDecoder(client.conn)
47     for {
48         var req proto.Request
49         if err := decoder.Decode(&req); err != nil {
50             client.logger.Error("cannot decode message", "reason", err)
51             break
52         } else {
53             client.logger.Info("received command", "command", req.Command)
54             if client.handleRequest(&req) {
55                 client.logger.Info("shutting down connection")
56                 break
57             }
58         }
59     }
60 }

```

Листинг 5 — server.go(продолжение)

```
1 func (client *Client) respond(status string, data interface{}) {
2     var raw json.RawMessage
3     raw, _ = json.Marshal(data)
4     client.enc.Encode(&proto.Response{Status: status, Data: &raw})
5 }
6
7 func main() {
8     var addrStr string
9     flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address
10     and port")
11     flag.Parse()
12     if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
13         log.Error("address resolution failed", "address", addrStr)
14     } else {
15         log.Info("resolved TCP address", "address", addr.String())
16         if listener, err := net.ListenTCP("tcp", addr); err != nil {
17             log.Error("listening failed", "reason", err)
18         } else {
19             for {
20                 if conn, err := listener.AcceptTCP(); err != nil {
21                     log.Error("cannot accept connection", "reason", err)
22                 } else {
23                     log.Info("accepted connection", "address", conn.RemoteAddr().
24                     String())
25                     go NewClient(conn).serve()
26                 }
27             }
28 }
```

Результат запуска представлен на рисунке 1, 2

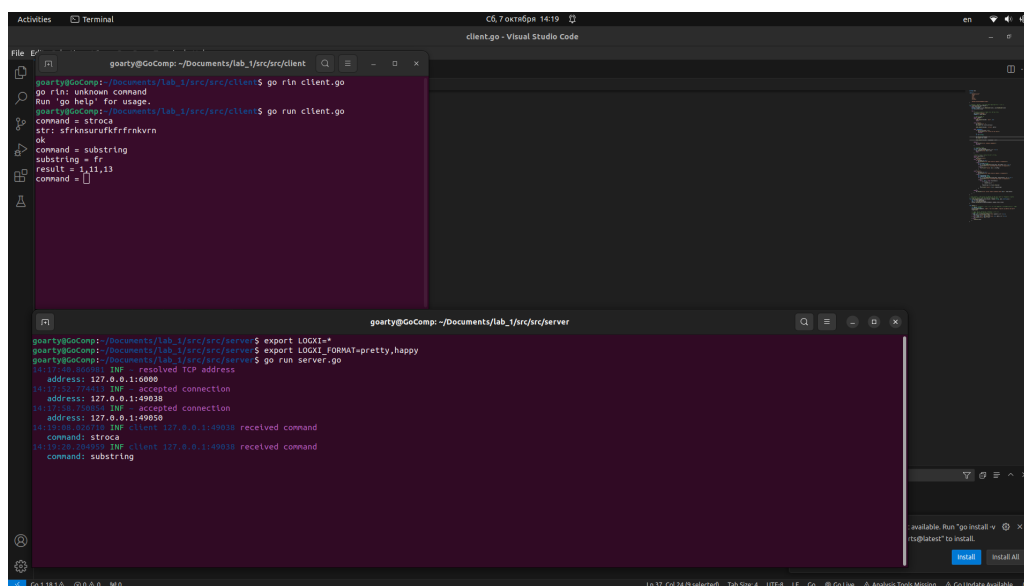


Рис. 1 — Выполнения протокола при 1-м пользователе

The screenshot shows a Visual Studio Code interface with a terminal window. The terminal is divided into three panes. The top-left pane shows the execution of a client program with the following output:

```
goarty@GoComp: ~/Documents/lab_1/src/src/client$ go run client.go
error: dial tcp 127.0.0.1:6000: connect: connection refused
goarty@GoComp: ~/Documents/lab_1/src/src/client$ go run client.go
command = stroca
str: jrlnlvldrgjgtgtgtjlv
ok
command = substring
substring = gt
result = 32,14,16
command =
```

The top-right pane shows the execution of the same client program with different input:

```
goarty@GoComp: ~/Documents/lab_1/src/src/client$ go run client.go
command = stroca
str: krlrllrjgdvlldgubf
ok
command = substring
substring = gt
result =
command =
```

The bottom pane shows the execution of a server program with the following output:

```
goarty@GoComp: ~/Documents/lab_1/src/src/server$ export LOGXI="
goarty@GoComp: ~/Documents/lab_1/src/src/server$ export LOGX_FORMAT=pretty,happy
goarty@GoComp: ~/Documents/lab_1/src/src/server$ go run server.go
14:23:59.10001 INF resolved TCP address
address: 127.0.0.1:6000
14:23:59.10001 INF accepted connection
address: 127.0.0.1:37954
14:23:59.10001 INF accepted connection
address: 127.0.0.1:37962
14:23:59.10001 INF client 127.0.0.1:37954 received command
command: stroca
14:23:59.10001 INF client 127.0.0.1:37962 received command
command: stroca
14:23:59.10001 INF client 127.0.0.1:37954 received command
command: substring
14:23:59.10001 INF client 127.0.0.1:37962 received command
command: substring
```

Рис. 2 — Выполнения протокола при 2-х пользователях