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 ИУ9-32Б Волохов А. В.

Преподаватель Посевин Д. П.

Москва 2023

1 Задание

Целью работы является знакомство с принципами разработки протоколов прикладного уровня и их реализацией на языке Go. Необходимо реализовать протокол редактирования предложения с возможностью вставлять и удалять слова Исходный код программы представлен в листингах 1– 2– 3– 4– 5– 6.

Листинг 1 — proto.go

```
1  package proto
2
3  import "encoding/json"
4
5  type Request struct {
6    Command string `json:"command"`
7    Data *json.RawMessage `json:"data"`
8  }
9
10 type Response struct {
11   Status string `json:"status"`
12   Data *json.RawMessage `json:"data"`
13 }
14
15 type Word struct {
16   Word string `json:"word"`
17   Key string `json:"num"`
18 }
```

Листинг 2 — server.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "flag"
6    "fmt"
7    "github.com/mgutz/logxi/v1"
8    "net"
9    "strings"
10 )
11
12 import "file/lab1/src/proto"
13
14 type Client struct {
15   logger log.Logger
16   conn   *net.TCPConn
17   enc    *json.Encoder
18   sent   map[string]string
19 }
20
21 func NewClient(conn *net.TCPConn) *Client {
22   return &Client{
23     logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())
       ),
24     conn:   conn,
25     enc:    json.NewEncoder(conn),
26     sent:   make(map[string]string),
27   }
28 }
29
30 func (client *Client) serve() {
31   defer client.conn.Close()
32   decoder := json.NewDecoder(client.conn)
33   for {
34     var req proto.Request
35     if err := decoder.Decode(&req); err != nil {
36       client.logger.Error("cannot decode message", "reason", err)
37       break
38     } else {
39       client.logger.Info("received command", "command", req.Command)
40       if client.handleRequest(&req) {
41         client.logger.Info("shutting down connection")
42         break
43       }
44     }
45   }
46 }
```

Листинг 3 — server.go - продолжение

```
1  func (client *Client) handleRequest(req *proto.Request) bool {
2    switch req.Command {
3    case "quit":
4      client.respond("ok", nil)
5      return true
6    case "add":
7      errorMsg := ""
8      if req.Data == nil {
9        errorMsg = "data field is absent"
10     } else {
11       var word proto.Word
12       if err := json.Unmarshal(*req.Data, &word); err != nil {
13         errorMsg = "malformed data field"
14       } else {
15         client.logger.Info("adding a word", "value", word.Word, "key",
     word.Key)
16         client.sent[word.Key] = word.Word
17       }
18     }
19     if errorMsg == "" {
20       client.respond("ok", nil)
21     } else {
22       client.logger.Error("adding failed", "reason", errorMsg)
23       client.respond("failed", errorMsg)
24     }
25   case "del":
26     if req.Data == nil {
27       client.logger.Error("deletion failed", "reason", "key is missing
     in data field")
28       client.respond("failed", "key is missing in data field")
29     } else {
30       keyToDelete := strings.Trim(string(*req.Data), "\"")
31       deletedWord, ok := client.sent[keyToDelete]
32       if ok {
33         delete(client.sent, keyToDelete)
34         client.respond("result", &proto.Word{
35           Key:  keyToDelete,
36           Word: deletedWord,
37         })
38       } else {
39         client.logger.Error("deletion failed", "reason", "key not found"
     )
40         client.respond("failed", "key not found")
41       }
42     }
43   case "sent":
44     concatenatedValues := ""
45     for _, value := range client.sent {
46       concatenatedValues += value + " "
47     }
48     client.respond("result", &proto.Word{
49       Key:  "sentence",
50       Word: concatenatedValues,
51     })
52
53
```

Листинг 4 — server.go - продолжение

```
default:
    client.logger.Error("unknown command")
    client.respond("failed", "unknown command")
  }
  return false
}

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

Листинг 5 — client.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "flag"
6    "fmt"
7    "github.com/skorobogatov/input"
8    "net"
9  )
10
11 import "file/lab1/src/proto"
12
13 func interact(conn *net.TCPConn) {
14   defer conn.Close()
15   encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
16   for {
17     fmt.Printf("command = ")
18     command := input.Gets()
19     switch command {
20     case "quit":
21       send_request(encoder, "quit", nil)
22       return
23     case "add":
24       var word proto.Word
25       fmt.Printf("word = ")
26       word.Word = input.Gets()
27       fmt.Printf("key = ")
28       word.Key = input.Gets()
29       send_request(encoder, "add", &word)
30     case "del":
31       fmt.Printf("key = ")
32       var key = input.Gets()
33       send_request(encoder, "del", key)
34     case "sent":
35       send_request(encoder, "sent", nil)
36     default:
37       fmt.Printf("error: unknown command\n")
38       continue
39     }
40     var resp proto.Response
41     if err := decoder.Decode(&resp); err != nil {
42       fmt.Printf("error: %v\n", err)
43       break
44     }
```

Листинг 6 — client.go - продолжение

```
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 word proto.Word
        if err := json.Unmarshal(*resp.Data, &word); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          fmt.Printf("result: word \"%s\" with key = %s \n", word.Word,
    word.Key)
        }
      }
    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, &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)
  }
}
```

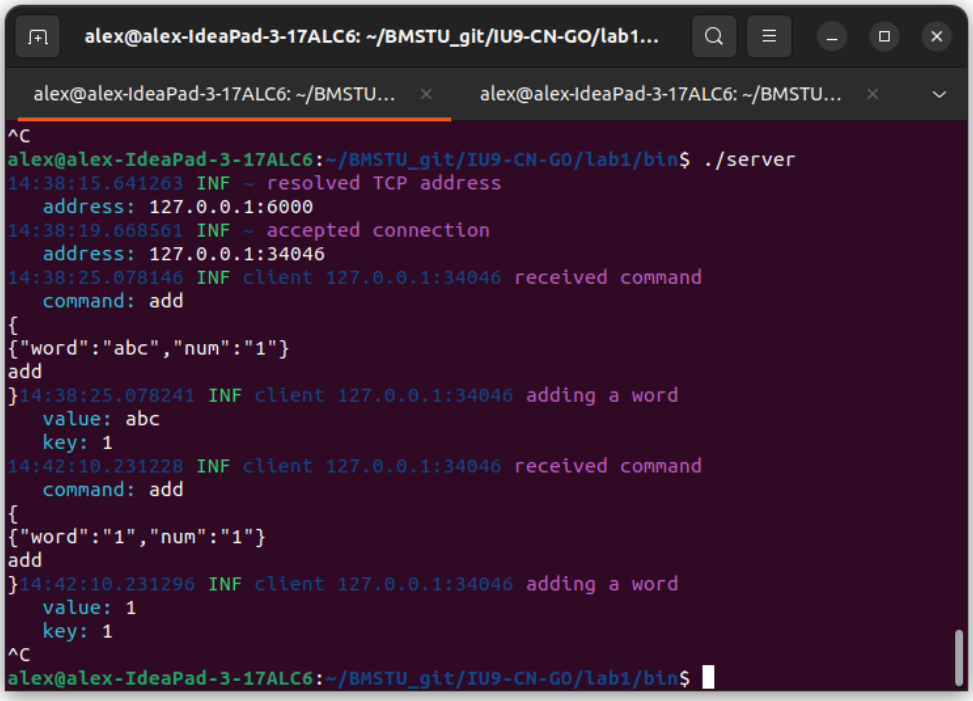

Рис. 1 — Сервер

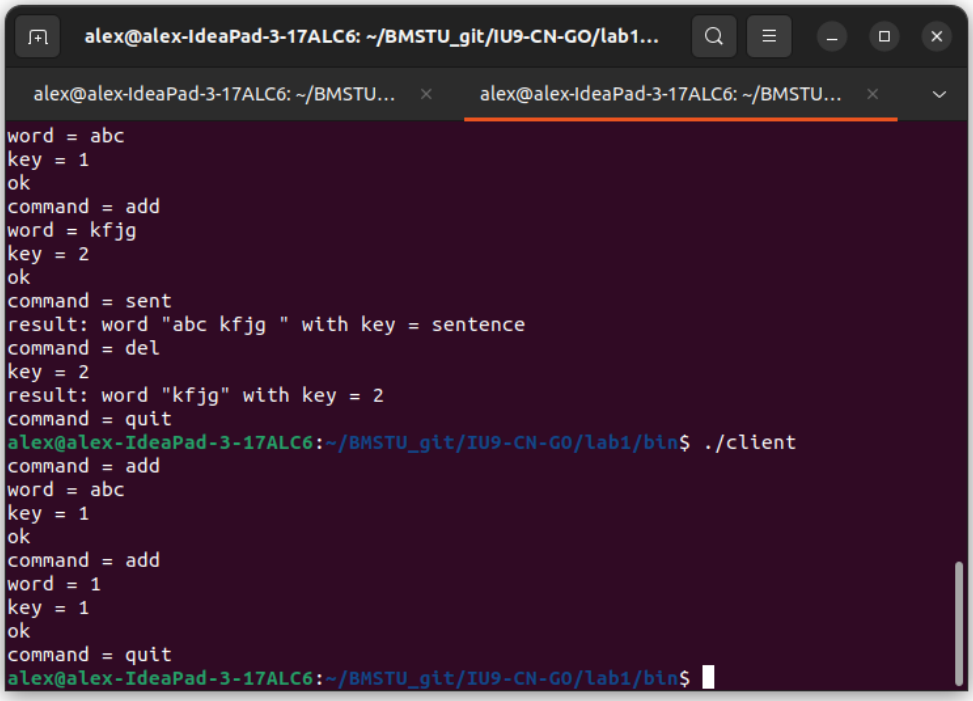

Рис. 2 — Клиент