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 Тараканов В. Д..

Преподаватель Посевин Д. П.

Москва 2024

1 Задание

Выполнение лабораторной работы состоит из двух частей.

◦ Разработать вариант протокола из таблиц 1–3. Протокол должен базироваться на текстовых сообщениях в формате JSON. Результатом разработки протокола должен быть набор типов языка Go, представляющих сообщения, и документация к ним в виде комментариев в исходном тексте.

◦ Написать на языке Go клиент и сервер, взаимодействующие по разработанному протоколу.

2 Результаты

Исходный код программы представлен в листингах 1– 3.

Листинг 1: client.go

```
1  package main
2
3  import (
4  "awesomeProject/sample/src/proto"
5  "encoding/json"
6  "flag"
7  "fmt"
8  "net"
9
10 "github.com/skorobogatov/input"
11 )
12
13
14 func interact(conn *net.TCPConn) {
15   defer conn.Close()
16   encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
17   for {
18
19     fmt.Printf("command = ")
20     command := input.Gets()
21
22
23     switch command {
24       case "quit":
25       send_request(encoder, "quit", nil)
26       return
27       case "add":
```

```
28            var frac string
29            fmt.Printf("numerator = ")
30            frac = input.Gets()
31            send_request(encoder, "add", &frac)
32            case "maxseq":
33            send_request(encoder, "maxseq", nil)
34            default:
35            fmt.Printf("error: unknown command\n")
36            continue
37          }
38
39
40          var resp proto.Response
41          if err := decoder.Decode(&resp); err != nil {
42            fmt.Printf("error: %v\n", err)
43            break
44          }
45
46          switch resp.Status {
47            case "ok":
48            fmt.Printf("ok\n")
49            case "failed":
50            if resp.Data == nil {
51              fmt.Printf("error: data field is absent in response\n")
52            } else {
53              var errorMsg string
54              if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil
    {
55                fmt.Printf("error: malformed data field in response\n")
56              } else {
57                fmt.Printf("failed: %s\n", errorMsg)
58              }
59            }
60            case "result":
61            if resp.Data == nil {
62              fmt.Printf("error: data field is absent in response\n")
63            } else {
64              var frac int
65              if err := json.Unmarshal(*resp.Data, &frac); err != nil {
66                fmt.Printf("error: malformed data field in response\n")
67              } else {
68                fmt.Printf("result: %d\n", frac)
69              }
70            }
71            default:
```

```
72          fmt.Printf("error: server reports unknown status %q\n", resp.
    Status)
73        }
74      }
75    }
76
77    func send_request(encoder *json.Encoder, command string, data
    interface{}) {
78      var raw json.RawMessage
79      raw, _ = json.Marshal(data)
80      encoder.Encode(&proto.Request{command, &raw})
81    }
82
83    func main() {
84      var addrStr string
85      flag.StringVar(&addrStr, "addr", "185.104.251.226:9999", "specify
    ip address and port")
86      flag.Parse()
87
88
89      if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
90        fmt.Printf("error: %v\n", err)
91      } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
92        fmt.Printf("error: %v\n", err)
93      } else {
94        interact(conn)
95      }
96    }
97
```

Листинг 2: proto.go

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

```
16
```

Листинг 3: server.go

```
1  package main
2
3  import (
4  "awesomeProject/sample/src/proto"
5  "encoding/json"
6  "flag"
7  "fmt"
8  "math"
9  "net"
10 "strconv"
11
12 log "github.com/mgutz/logxi/v1"
13 )
14
15
16 type Client struct {
17   logger log.Logger
18   conn   *net.TCPConn
19   enc    *json.Encoder
20   data   []int
21   count  int64
22 }
23
24
25 func NewClient(conn *net.TCPConn) *Client {
26   return &Client{
27     logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String
     ())),
28     conn:   conn,
29     enc:    json.NewEncoder(conn),
30     data:   make([]int, 1000),
31     count:  0,
32   }
33 }
34
35
36 func (client *Client) serve() {
37   defer client.conn.Close()
38   decoder := json.NewDecoder(client.conn)
39   for {
40     var req proto.Request
41     if err := decoder.Decode(&req); err != nil {
42       client.logger.Error("cannot decode message", "reason", err)
```

```
43          break
44        } else {
45          client.logger.Info("received command", "command", req.Command)
46          if client.handleRequest(&req) {
47            client.logger.Info("shutting down connection")
48            break
49          }
50        }
51      }
52    }
53
54    func maxS(nums []int) int {
55      currentSum := nums[0]
56      maxSum := nums[0]
57      for i := 1; i < len(nums); i++ {
58        currentSum = int(math.Max(float64(nums[i]), float64(currentSum+
       nums[i])))
59        maxSum = int(math.Max(float64(maxSum), float64(currentSum)))
60      }
61
62      return maxSum
63    }
64
65    func (client *Client) handleRequest(req *proto.Request) bool {
66      switch req.Command {
67        case "quit":
68        client.respond("ok", nil)
69        return true
70        case "add":
71        errorMsg := ""
72        if req.Data == nil {
73          errorMsg = "data field is absent"
74        } else {
75          var frac string
76          if err := json.Unmarshal(*req.Data, &frac); err != nil {
77            errorMsg = "malformed data field"
78          } else {
79            client.logger.Info("performing addition", "value", frac)
80            i, err := strconv.Atoi(frac)
81            if err != nil {
82
83              errorMsg = "bad num"
84            }
85            client.data = append(client.data, i)
86            client.count++
87
```

```
88          }
89        }
90        if errorMsg == "" {
91          client.respond("ok", nil)
92        } else {
93          client.logger.Error("addition failed", "reason", errorMsg)
94          client.respond("failed", errorMsg)
95        }
96        case "maxseq":
97        if client.count == 0 {
98          client.logger.Error("calculation failed", "reason", "division by
      zero")
99          client.respond("failed", "division by zero")
100       } else {
101         max := maxS(client.data)
102         client.respond("result", max)
103       }
104       default:
105       client.logger.Error("unknown command")
106       client.respond("failed", "unknown command")
107     }
108     return false
109   }
110
111   func (client *Client) respond(status string, data interface{}) {
112     var raw json.RawMessage
113     raw, _ = json.Marshal(data)
114     client.enc.Encode(&proto.Response{status, &raw})
115   }
116
117   func main() {
118
119     var addrStr string
120     flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip
      address and port")
121     flag.Parse()
122
123
124     if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
125       log.Error("address resolution failed", "address", addrStr)
126     } else {
127       log.Info("resolved TCP address", "address", addr.String())
128
129
130       if listener, err := net.ListenTCP("tcp", addr); err != nil {
131         log.Error("listening failed", "reason", err)
```

```
132         } else {
133
134           for {
135             if conn, err := listener.AcceptTCP(); err != nil {
136               log.Error("cannot accept connection", "reason", err)
137             } else {
138               log.Info("accepted connection", "address", conn.RemoteAddr()
      .String())
139
140               go NewClient(conn).serve()
141             }
142           }
143         }
144       }
145   }
146
```

Результат запуска представлен на рисунке 1– 2.

Рис. 1 — Результат

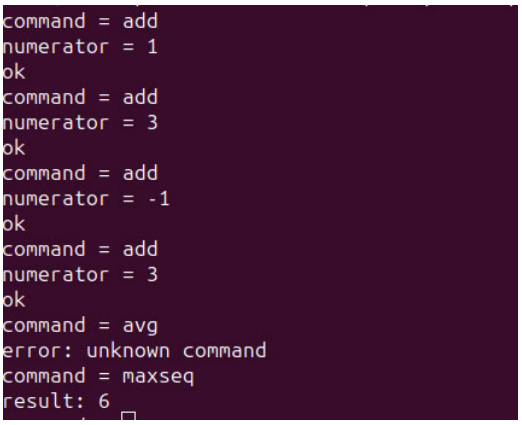

Рис. 2 — Результат