Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №1
«Простейший протокол прикладного уровня»
по курсу: «Компьютерные сети»

Выполнил:
Студент группы ИУ9-31Б
Гречко Г.В.

Проверил:
Посевин Д.П.

Москва, 2022

Цели

Целью данной работы является знакомство с принципами разработки протоколов прикладного уровня и их реализацией на языке Go.

Задачи

Протокол вычисления наибольшей высоты, на которую поднимется камень, брошенный с земли под углом к горизонту.

Решение

Исходный код

client.go

```go
package main

import (
    "encoding/json"
    "flag"
    "fmt"
    "lab1/src/proto"
    "net"
)

// interact - функция, содержащая цикл взаимодействия с сервером.
func interact(conn *net.TCPConn) {
    defer conn.Close()
    encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
    for {
        // Чтение команды из стандартного потока ввода
        fmt.Printf("command = ")
        var command string
        fmt.Scan(&command)

        // Отправка запроса.
        switch command {
        case "quit":
            sendRequest(encoder, "quit", nil)
            return
        case "calculate":
            var task proto.Task
            fmt.Printf("angle = ")
            fmt.Scan(&task.Angle)
            fmt.Printf("velocity = ")
            fmt.Scan(&task.Velocity)
            sendRequest(encoder, "calculate", &task)
        default:
            fmt.Printf("error: unknown command\n")
            continue
        }

        // Получение ответа.
        var resp proto.Response
        if err := decoder.Decode(&resp); err != nil {
            fmt.Printf("error: %v\n", err)
            break
        }

        // Вывод ответа в стандартный поток вывода.
```

```
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 height float64
65                  if err := json.Unmarshal(*resp.Data, &height); err != nil
                     ↪  {
66                      fmt.Printf("error: malformed data field in
    ↪  response\n")
67                  } else {
68                      fmt.Printf("max height: %v\n", height)
69                  }
70              }
71          default:
72              fmt.Printf("error: server reports unknown status %q\n",
    ↪  resp.Status)
73          }
74      }
75  }
76
77  // sendRequest - вспомогательная функция для передачи запроса с указанной
     ↪  командой
78  // и данными. Данные могут быть пустыми (data == nil).
79  func sendRequest(encoder *json.Encoder, command string, data interface{})
     ↪  {
80      var raw json.RawMessage
81      raw, _ = json.Marshal(data)
82      encoder.Encode(&proto.Request{Command: command, Data: &raw})
83  }
84
85  func main() {
86      // Работа с командной строкой, в которой может указываться
         ↪  необязательный ключ -addr.
87      var addrStr string
88      flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip
    ↪  address and port")
89      flag.Parse()
90
91      // Разбор адреса, установка соединения с сервером и
92      // запуск цикла взаимодействия с сервером.
93      if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
94          fmt.Printf("error: %v\n", err)
95      } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
96          fmt.Printf("error: %v\n", err)
97      } else {
98          interact(conn)
```

```
    }
}
```

server.go

```go
package main

import (
    "encoding/json"
    "flag"
    "fmt"
    "lab1/src/proto"
    "math"
    "net"

    log "github.com/mgutz/logxi/v1"
)

const g = 9.8

// Client - состояние клиента.
type Client struct {
    logger log.Logger    // Объект для печати логов
    conn   *net.TCPConn  // Объект TCP-соединения
    enc    *json.Encoder // Объект для кодирования и отправки сообщений
    count  int64         // Количество полученных от клиента задач
}

// NewClient - конструктор клиента, принимает в качестве параметра
// объект TCP-соединения.
func NewClient(conn *net.TCPConn) *Client {
    return &Client{
        logger: log.New(fmt.Sprintf("client %s",
    ↪  conn.RemoteAddr().String())),
        conn:   conn,
        enc:    json.NewEncoder(conn),
        count:  0,
    }
}

// serve - метод, в котором реализован цикл взаимодействия с клиентом.
// Подразумевается, что метод serve будет вызываться в отдельной
    ↪  go-программе.
func (client *Client) serve() {
    defer client.conn.Close()
    decoder := json.NewDecoder(client.conn)
    for {
        var req proto.Request
        if err := decoder.Decode(&req); err != nil {
            client.logger.Error("cannot decode message", "reason", err)
            break
        } else {
            client.logger.Info("received command", "command",
    ↪  req.Command)
            if client.handleRequest(&req) {
                client.logger.Info("shutting down connection")
                break
            }
        }
    }
}

```

```
55 // handleRequest - метод обработки запроса от клиента. Он возвращает
    ↪ true,
56 // если клиент передал команду "quit" и хочет завершить общение.
57 func (client *Client) handleRequest(req *proto.Request) bool {
58     switch req.Command {
59     case "quit":
60         client.respond("ok", nil)
61         return true
62     case "calculate":
63         errorMsg := ""
64         var x float64
65         if req.Data == nil {
66             errorMsg = "data field is absent"
67         } else {
68             var task proto.Task
69             if err := json.Unmarshal(*req.Data, &task); err != nil {
70                 errorMsg = "malformed data field"
71             } else {
72                 x = math.Pow(task.Velocity, 2) *
    ↪ math.Pow(math.Sin(task.Angle * math.Pi / 180), 2) / (2 * g)
73                 client.logger.Info("performing calculations", "value", x)
74                 client.count++
75             }
76         }
77         if errorMsg == "" {
78             client.respond("result", &x)
79         } else {
80             client.logger.Error("calculation failed", "reason", errorMsg)
81             client.respond("failed", errorMsg)
82         }
83     default:
84         client.logger.Error("unknown command")
85         client.respond("failed", "unknown command")
86     }
87     return false
88 }
89
90 // respond - вспомогательный метод для передачи ответа с указанным
    ↪ статусом
91 // и данными. Данные могут быть пустыми (data == nil).
92 func (client *Client) respond(status string, data interface{}) {
93     var raw json.RawMessage
94     raw, _ = json.Marshal(data)
95     client.enc.Encode(&proto.Response{Status: status, Data: &raw})
96 }
97
98 func main() {
99     // Работа с командной строкой, в которой может указываться
        ↪ необязательный ключ -addr.
100     var addrStr string
101     flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip
    ↪ address and port")
102     flag.Parse()
103
104     // Разбор адреса, строковое представление которого находится в
        ↪ переменной addrStr.
105     if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
106         log.Error("address resolution failed", "address", addrStr)
107     } else {
108         log.Info("resolved TCP address", "address", addr.String())
109
110         // Инициация слушания сети на заданном адресе.
```

```
        if listener, err := net.ListenTCP("tcp", addr); err != nil {
            log.Error("listening failed", "reason", err)
        } else {
            // Цикл приёма входящих соединений.
            for {
                if conn, err := listener.AcceptTCP(); err != nil {
                    log.Error("cannot accept connection", "reason", err)
                } else {
                    log.Info("accepted connection", "address",
    ↪ conn.RemoteAddr().String())

                    // Запуск go-программы для обслуживания клиентов.
                    go NewClient(conn).serve()
                }
            }
        }
    }
}
```

proto.go

```
package proto

import "encoding/json"

// Request -- запрос клиента к серверу.
type Request struct {
    // Поле Command может принимать три значения:
    // * "quit" - прощание с сервером (после этого сервер рвёт
     ↪  соединение);
    // * "calculate" - передача новой задачи на сервер;
    Command string `json:"command"`

    Data *json.RawMessage `json:"data"`
}

// Response -- ответ сервера клиенту.
type Response struct {
    // Поле Status может принимать три значения:
    // * "ok" - успешное выполнение команды "quit";
    // * "failed" - в процессе выполнения команды произошла ошибка;
    // * "result" - максимальная высота вычислена.
    Status string `json:"status"`

    // Если Status == "failed", то в поле Data находится сообщение об
     ↪  ошибке.
    // Если Status == "result", в поле Data должна лежать высота
    // В противном случае, поле Data пустое.
    Data *json.RawMessage `json:"data"`
}

// Task -- условие задачи для вычисления сервером
type Task struct {
    // Угол от горизонта в градусах
    Angle float64 `json:"angle"`

    // Общая скорость тела в начале движения
    Velocity float64 `json:"Velocity"`
}
```

Пример работы

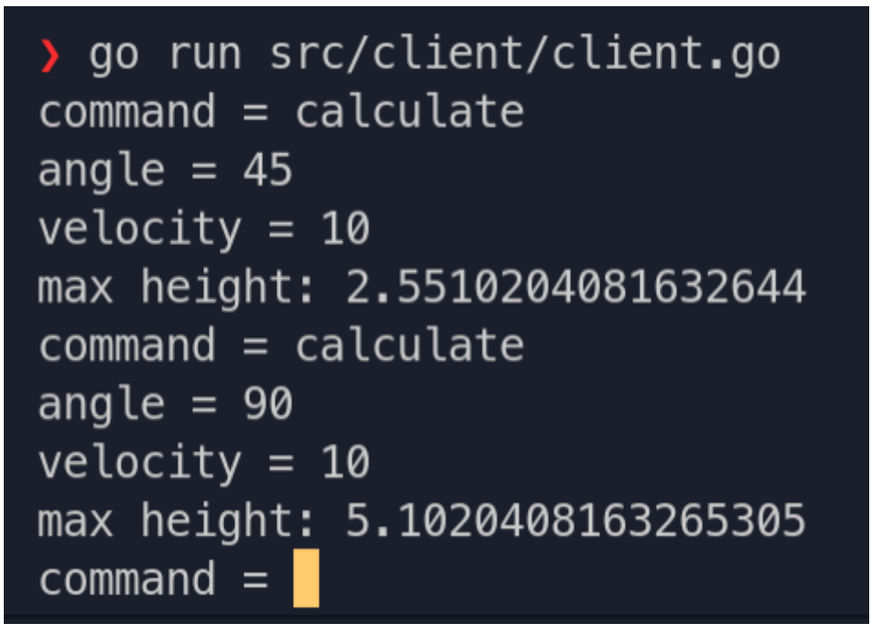

Рис. 1: Работа клиента

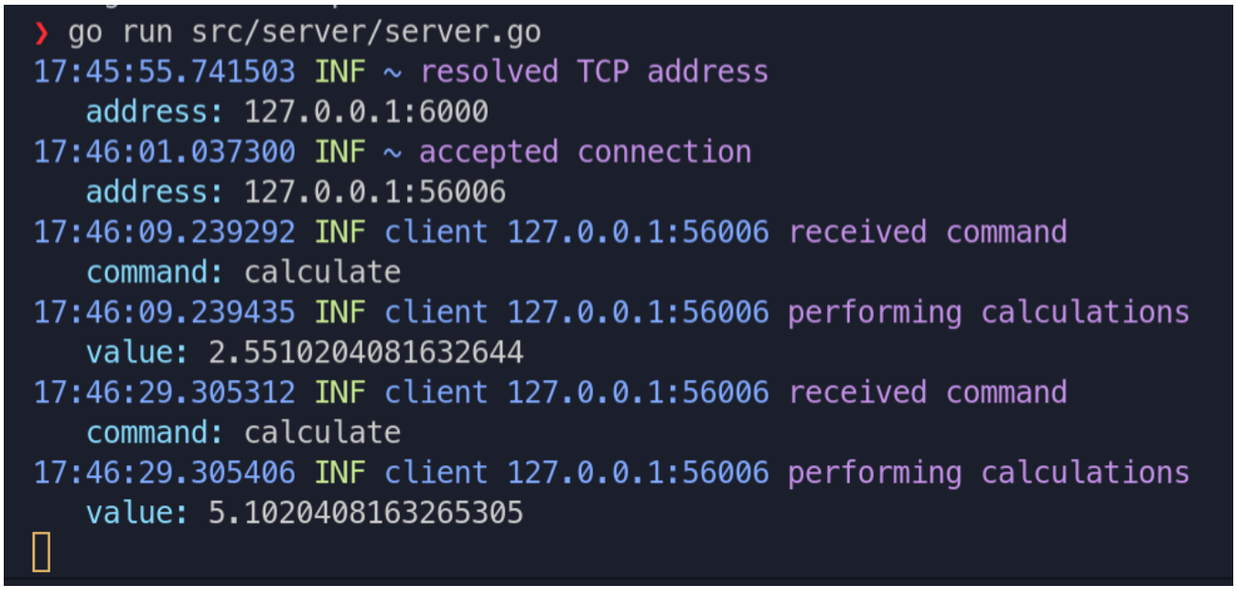

Рис. 2: Работа сервера