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» Вариант 26

Студент группы ИУ9-32Б Павлов И. П.

Преподаватель Посевин Д. П.

1 Цель работы

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

2 Условие

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

3 Реализация 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);
        // * "calc" - передача нового значения 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" или "add";
        // * "failed" - в процессе выполнения команды произошла ошибка;
        // * "result" - ответ вычислен
        Status string `json:"status"`
        // Если Status == "failed", то в поле Data находится сообщение об ошибке.
        // Если Status == "result", в поле Data должен лежать Answer
        // В противном случае, поле Data пустое.
        Data *json.RawMessage `json:"data"`
// Conditions -- условия броска
type Conditions struct {
        // Angle -- угол броска
        Angle string `json:"ang"`
        // Speed -- скорость броска
        Speed string `json:"spd"`
// Answer -- ответ на задачу.
```

4 Реализация server.go

```
package main
import (
        "encoding/json"
        "flag"
        "fmt"
        "github.com/mgutz/logxi/v1"
        "math"
        "math/big"
        "net"
        "strconv"
)
import "lab1/src/proto"
const g float64 = 10
// Client - состояние клиента.
type Client struct {
        logger log.Logger // Объект для печати логов
              *net.TCPConn // Объект ТСР-соединения
               *json.Encoder // Объект для кодирования и отправки сообщений
        sum
               *big.Rat
                             // Текущая сумма полученных от клиента дробей
        count int64
                             // Количество полученных от клиента дробей
}
// NewClient - конструктор клиента, принимает в качестве параметра
// объект ТСР-соединения.
func NewClient(conn *net.TCPConn) *Client {
        return &Client{
                logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())),
                conn:
                        json.NewEncoder(conn),
                enc:
                sum:
                        big.NewRat(0, 1),
                count: 0,
        }
}
// serve - метод, в котором реализован цикл взаимодействия с клиентом.
// Подразумевается, что метод serve будет вызаваться в отдельной qo-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
                        }
                }
       }
}
// handleRequest - метод обработки запроса от клиента. Он возвращает true,
// если клиент передал команду "quit" и хочет завершить общение.
func (client *Client) handleRequest(req *proto.Request) bool {
        switch req.Command {
        case "quit":
                client.respond("ok", nil)
                return true
        case "calc":
                errorMsg := ""
                var v, alpha float64
                if req.Data == nil {
                        errorMsg = "data field is absent"
                } else {
                        var cond proto.Conditions
                        if err := json.Unmarshal(*req.Data, &cond); err != nil {
                                errorMsg = "malformed data field"
                        } else {
                                v, err = strconv.ParseFloat(cond.Speed, 64)
                                if err != nil {
                                        errorMsg = "incorrect speed input"
                                alpha, err = strconv.ParseFloat(cond.Angle, 64)
                                if err != nil {
                                        errorMsg = "incorrect angle input"
                                }
                        }
                }
                if errorMsg == "" {
                        radAlpha := alpha * (math.Pi / 180.0)
                        h := math.Pow(v*math.Sin(radAlpha), 2) / (2 * g)
                        ans := strconv.FormatFloat(h, 'f', 2, 64)
                        client.respond("result", &proto.Answer{Result: ans})
                        client.respond("ok", nil)
                } else {
                        client.logger.Error("calculation failed", "reason", errorMsg)
                        client.respond("failed", errorMsg)
                }
        default:
                client.logger.Error("unknown command")
                client.respond("failed", "unknown command")
        return false
}
// respond - вспомогательный метод для передачи ответа с указанным статусом
// и данными. Данные могут быть пустыми (data == nil).
func (client *Client) respond(status string, data interface{}) {
        var raw json.RawMessage
        raw, _ = json.Marshal(data)
        client.enc.Encode(&proto.Response{status, &raw})
```

```
}
        // Работа с командной строкой, в которой может указываться необязательный ключ -addr.
        var addrStr string
        flag.StringVar(&addrStr, "addr", "localhost:6000", "specify ip address and port")
        flag.Parse()
        // Разбор адреса, строковое представление которого находится в переменной addrStr.
        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())
                // 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 до-программы для обслуживания клиентов.
                                        go NewClient(conn).serve()
                                }
                        }
               }
       }
}
```

5 Реализация client.go

```
switch command {
                case "quit":
                        send_request(encoder, "quit", nil)
                        return
                case "calc":
                        var cond proto.Conditions
                        fmt.Printf("angle = ")
                        cond.Angle = input.Gets()
                        fmt.Printf("speed = ")
                        cond.Speed = input.Gets()
                        send_request(encoder, "calc", &cond)
                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.
                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)
                                7
                        }
                case "result":
                        if resp.Data == nil {
                                fmt.Printf("error: data field is absent in response\n")
                        } else {
                                var ans proto.Answer
                                if err := json.Unmarshal(*resp.Data, &ans); err != nil {
                                        fmt.Printf("error: malformed data field in response\n")
                                } else {
                                        fmt.Printf("result: %s\n", ans.Result)
                default:
                        fmt.Printf("error: server reports unknown status %q\n", resp.Status)
                }
       }
}
// send_request - вспомогательная функция для передачи запроса с указанной командой
// и данными. Данные могут быть пустыми (data == nil).
func send_request(encoder *json.Encoder, command string, data interface{}) {
```

// Отправка запроса.

```
var raw json.RawMessage
        raw, _ = json.Marshal(data)
        encoder.Encode(&proto.Request{command, &raw})
}
func main() {
        // Работа с командной строкой, в которой может указываться необязательный ключ -addr.
        var addrStr string
        flag.StringVar(&addrStr, "addr", "localhost:6000", "specify ip address and port")
        flag.Parse()
        // Разбор адреса, установка соединения с сервером и
        // запуск цикла взаимодействия с сервером.
        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)
        }
}
```

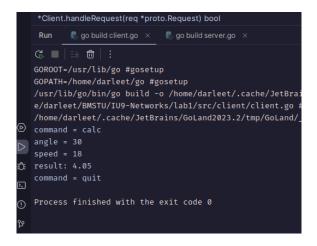


Рис. 1: Вывод при нормальных исходных данных

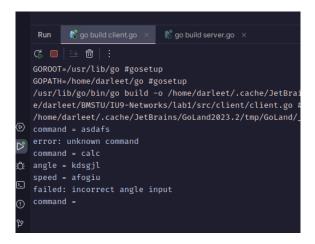


Рис. 2: Вывод при некорректных входных данных