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 №3.2
«Протокол одноранговой сети»
по курсу: «Компьютерные сети»

Выполнил:
Студент группы ИУ9-32Б
Гнатенко Т. А.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Целью данной работы является разработка одноранговой сетевой службы

Задачи

Вычисление интеграла (полносвязное) Топология: полносвязная. Информация, известная пиру при запуске: его IP-адрес и порт, а также IP-адреса и порты возможных соседей. Описание службы: каждый пир через стандартный поток ввода может принять команду на численное вычисление определённого интеграла функции одной переменной, заданной строкой (доступны четыре арифметические операции, а также синус и косинус). Замечание: пир должен разбить интервал, на котором вычисляется интеграл, на равные отрезки по числу соседних пиров и разослать каждому соседнему пиру задание на вычисление интеграла на своём отрезке.

Решение

server.go

```
package main

import (
    "encoding/json"
    "net/http"

    . "lab3_2/proto"

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

var AllConnections map[string]bool

func handleRegister(w http.ResponseWriter, r *http.Request)
 ↪  {
    var conf Node
```

```
    conf.Addr = r.RemoteAddr
    for addr := range AllConnections {
        conf.Connections = append(conf.Connections, addr)
    }
    encoded, _ := json.Marshal(conf)
    AllConnections[conf.Addr] = true
    w.Write(encoded)
}

func handleLogout(w http.ResponseWriter, r *http.Request) {
    delete(AllConnections, r.RemoteAddr)
}

func handleUpdatePeers(w http.ResponseWriter, r
 ↪  *http.Request) {
    var conf Node

    for addr := range AllConnections {
        conf.Connections = append(conf.Connections, addr)
    }
    encoded, err := json.Marshal(conf)
    if err != nil {
        log.Error(err.Error())
        return
    }
    w.Write(encoded)
}

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/register", handleRegister)
    mux.HandleFunc("/logout", handleLogout)
    mux.HandleFunc("/updatepeers", handleUpdatePeers)

    AllConnections = make(map[string]bool)

    server := http.Server{
        Addr:    "0.0.0.0:8000",
        Handler: mux,
```

```
    }
    log.Info("Starting server on 0.0.0.0:8000")
    err := server.ListenAndServe()
    if err != nil {
        log.Error(err.Error())
    }
}
```

client.go

```
package main

import (
    "bufio"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "os"
    "strings"

    . "lab3_2/proto"

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

const url string = "http://localhost:8000"

func main() {
    var conf Node
    resp, err := http.Get(url + "/register")
    if err != nil {
        log.Error(err.Error())
        return
    }
    body, _ :=
 ↪  bufio.NewReader(resp.Body).ReadBytes('\u00C6')
    if err := json.Unmarshal(body, &conf); err != nil {
        log.Error(err.Error())
        return
```

```
    }

    conf.Run(handleServer, handleClient)
}

func handleServer(n *Node) {
    mux := http.NewServeMux()

    handleConnection := func(w http.ResponseWriter, r
 ↪  *http.Request) {
        var pack Package
        body, err :=
 ↪  bufio.NewReader(r.Body).ReadBytes('\u00C6')
        if err != nil && err != io.EOF {
            log.Error(err.Error())
            fmt.Fprintf(w, "I'm chereshnya")
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        if err := json.Unmarshal(body, &pack); err != nil {
            log.Error(err.Error())
            fmt.Fprintf(w, "I'm teepot")
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        log.Info("task from ", pack.From)
        log.Info("desc", "integral ", pack.Data.Expr, "upper
 ↪  bound ", pack.Data.A, "lower bound", pack.Data.B,
 ↪  "result", pack.Data.Calculate())
    }

    mux.HandleFunc("/", handleConnection)
    server := http.Server{
        Addr:    n.Addr[:len(n.Addr)-2] +
 ↪  string(n.Addr[len(n.Addr)-1]),
        Handler: mux,
    }
    go server.ListenAndServe()
}
```

```go
func handleClient(n *Node) {
    for {
        message := InputString()
        splited := strings.Split(message, " ")
        switch splited[0] {
        case "exit":
            http.Get(url + "/logout")
            os.Exit(0)
        case "calc":
            n.UpdatePeers()
            fmt.Print("enter integral: ")
            var a, b float32
            integral, _ :=
 ↪  bufio.NewReader(os.Stdin).ReadString('\n')
            fmt.Print("enter bounds: ")
            fmt.Scan(&a, &b)
            n.SendToAll(Integral{
                Expr: integral,
                A:    float64(a),
                B:    float64(b),
            })

        default:
            log.Warn("unknown command, please try again")
        }
    }
}

func InputString() string {
    msg, _ := bufio.NewReader(os.Stdin).ReadString('\n')
    return strings.Replace(msg, "\n", "", -1)
}
```

proto.go

```go
package proto

import (
    "bufio"
```

```
    "encoding/json"
    "fmt"
    "math"
    "net/http"
    "strconv"
    "strings"

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

const url string = "http://localhost:8000"

type Integral struct {
    Expr string  `json:"expr"`
    A    float64 `json:"a"`
    B    float64 `json:"b"`
}

func (i *Integral) Calculate() float64 {
    expr := strings.Split(i.Expr, " ")
    if len(expr) < 2 {
        if expr[0] == "x" {
            return i.B*i.B - i.A*i.A
        }
        if len(i.Expr) > 5 {
            switch i.Expr[:4] {
            case "sin(":
                return math.Cos(i.B) - math.Cos(i.A)
            case "cos(":
                return math.Sin(i.B) - math.Sin(i.A)
            }
        }
        x, err := strconv.Atoi(expr[0])
        if err != nil {
            fmt.Println("unknown integral, please try
 ↪  again")
            return 0
        }
        return float64(x) * (i.B - i.A)
    }
```

```
    x1 := Integral{Expr: expr[0], A: i.A, B: i.B}
    x2 := Integral{Expr: expr[2], A: i.A, B: i.B}
    switch expr[1] {
    case "+":
        return x1.Calculate() + x2.Calculate()
    case "-":
        return x1.Calculate() - x2.Calculate()
    case "/":
        {
            if expr[0] != "1" {
                fmt.Println("unknown integral, please try
↪   again")
                return 0
            }
            return math.Log(i.B) - math.Log(i.A)
        }
    case "*":
        {
            y1, err1 := strconv.Atoi(expr[0])
            y2, err2 := strconv.Atoi(expr[2])
            if err1 == nil {
                if err2 == nil {
                    return float64(y1*y2) * (i.B - i.A)
                }
                x := Integral{Expr: expr[2], A: i.A, B: i.B}
                return float64(y1) * x.Calculate()
            }
            if err2 == nil {
                x := Integral{Expr: expr[0], A: i.A, B: i.B}
                return float64(y2) * x.Calculate()
            }
            return (i.B*i.B*i.B - i.A*i.A*i.A) / 3
        }
    }

    fmt.Println("unknown integral, please try again")
    return 0
}

// Config - параметры пира
```

```
type Node struct {
    Addr        string   `json:"addr"`
    Connections []string `json:"connections"`
}

type Package struct {
    To   string   `json:"to"`
    From string   `json:"from"`
    Data Integral `json:"integral"`
}

func (n *Node) UpdatePeers() {
    resp, err := http.Get(url + "/updatepeers")
    if err != nil {
        log.Error("request failed", "reason", err.Error())
    } else {
        body, _ :=
 ↪  bufio.NewReader(resp.Body).ReadBytes('\u00C6')
        var foo Node
        if err := json.Unmarshal(body, &foo); err != nil {
            log.Error(err.Error())
            return
        }
        n.Connections = nil
        for _, v := range foo.Connections {
            if v != n.Addr {
                n.Connections = append(n.Connections, v)
            }
        }
    }
}

func (n *Node) Run(handleServer func(*Node), handleClient
 ↪  func(*Node)) {
    go handleServer(n)
    handleClient(n)
}

func (n *Node) SendToAll(message Integral) {
    var new_pack = Package{
```

```
            From: n.Addr,
            Data: message,
        }
        count := len(n.Connections)
        ab := (message.B - message.A) / float64(count)
        new_pack.Data.A = message.A
        new_pack.Data.B = new_pack.Data.A
        for _, addr := range n.Connections {
            new_pack.Data.A = new_pack.Data.B
            new_pack.Data.B += ab
            new_pack.To = addr
            fmt.Println(new_pack)
            n.Send(new_pack)
        }
    }

    func (n *Node) Send(pack Package) {
        json_packet, err := json.Marshal(pack)
        if err != nil {
            fmt.Println(err)
            return
        }
        _, err =
     ↪  http.Post("http://"+pack.To[:len(pack.To)-2]+string(pack.To[len(pack.T
     ↪  "application/json",
     ↪  strings.NewReader(string(json_packet)))
        if err != nil {
            fmt.Println(err)
        }
    }
```

Вывод

```
❯ make client
LOGXI=* LOGXI_FORMAT=pretty,happy go run client/*.go
calc
enter integral: sin(x)
enter bounds: 3 6
```

Рис. 1: отправка интеграла

```
❯ make client
LOGXI=* LOGXI_FORMAT=pretty,happy go run client/*.go
13:46:09.822505 INF ~ task from
   _: 127.0.0.1:60580
13:46:09.822678 INF ~ desc
   integral : sin(x)
   upper bound : 3
   lower bound: 4.5
   result: 0.7791966971696657
```

Рис. 2: решение частей интеграла другими пирами

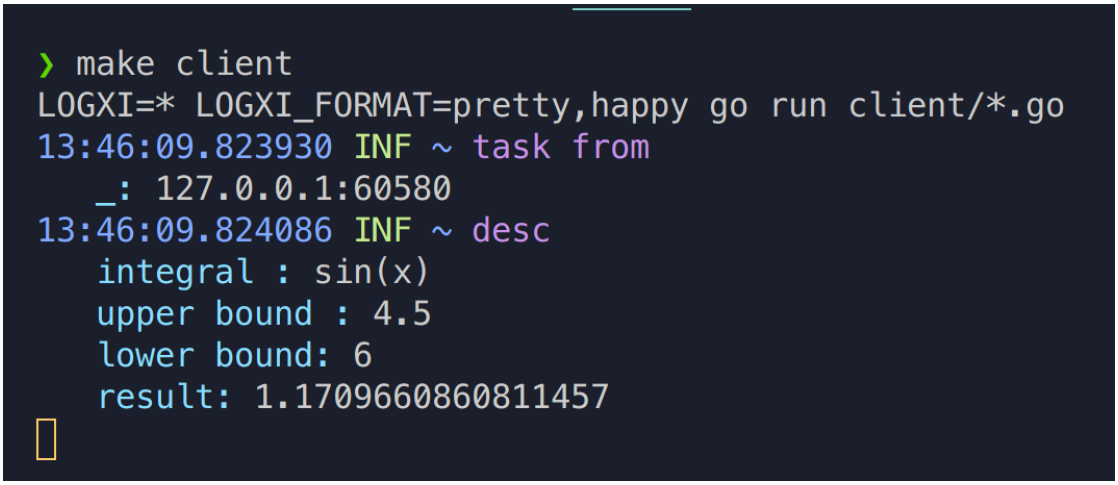

Рис. 3: решение частей интеграла другими пирами