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

«Протокол одноранговой сети»

по курсу: «Компьютерные сети»

Выполнила:

Студентка группы ИУ9-31Б

Яровикова А. С.

Проверил:

Посевин Д. П.

1. **Условия задачи.**

Краткое описание вариантов одноранговых сетевых служб, один из которых нужно раз-работать в ходе выполнения лабораторной работы, приведено в таблицах 1–10.

36.Социальная сеть (полносвязная)

Топология: полносвязная.

Информация, известная пиру при запуске: его имя, IP-адрес и порт,

а также IP-адреса и порты возможных соседей.

Описание службы: каждый пир через стандартный поток ввода

принимает команды – зафрендить другого пира по имени,

отфрендить другого пира, распечатать список френдов.

Замечание: два пира – френды, если они друг друга зафрендили.

1. **Код решения**

package main

import (

"fmt"

"net"

"encoding/json"

"github.com/skorobogatov/input"

"github.com/mgutz/logxi/v1"

)

// Response -- ответ от соседнего пира.

type Response struct {

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

}

// Request -- сообщение от соседнего пира

type Request struct {

// В поле Data должен лежать элемент

// в виде структуры Elem.

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

// Поле Command может принимать три значения:

// \* "friend" - зафрендить другого пира по имени;

// \* "unfriend" - отфрендить другого пира;

// \* "quit" - завершить сввязь с данным пиром;

// \* "list" - распечатать список френдов.

Command string `json:"command"`

}

// Elem -- структура элемента.

type Elem struct {

// IP-адрес пира

IP string

// имя пира тк мы френдим по имени

Name string

// имя пира от которого запрос запрос

Str string

}

var MyAddr string = ""

var name string = ""

var ppl [5]string

var localhost = "127.0.0.1"

var encoder \*json.Encoder = nil

var decoder \*json.Decoder = nil

var friends [5][5]int

/\*func giveNames() {

ppl[0] = "John"

ppl[1] = "Alice"

ppl[2] = "Bob"

ppl[3] = "Peter"

ppl[4] = "Ann"

}\*/

var board map[string]int

var board1 map[int]string

// Peer - соединение с другим пиром

type Peer struct {

logger log.Logger // Объект для печати логов

conn \*net.TCPConn // Объект TCP-соединения

enc \*json.Encoder // Объект для кодирования и отправки сообщений

dec \*json.Decoder // Объект для декодирования сообщений

}

// NewPeer - конструктор объекта пира, принимает в качестве параметра

// объект TCP-соединения.

func NewPeer(conn \*net.TCPConn) \*Peer {

return &Peer{

logger: log.New(fmt.Sprintf("peer %s", conn.RemoteAddr().String())),

conn: conn,

enc: json.NewEncoder(conn),

dec: json.NewDecoder(conn),

}

}

// send\_request - вспомогательная функция для передачи запроса с указанной командой

// и данными. Данные могут быть пустыми (data == nil).

func send\_request(encoder \*json.Encoder, com string, data interface{}) {

var raw json.RawMessage

raw, \_ = json.Marshal(data)

encoder.Encode(&Request{&raw, com})

}

// interact - функция, содержащая цикл взаимодействия с пользователем.

func interact(conns []conStruct, curName string) {

for {

var command string

command = input.Gets()

switch command {

case "friend":

var str string

fmt.Print("enter name: ")

str = input.Gets()

name = str

send\_request(encoder, "friend", Elem{MyAddr, name, curName}) //name - с кем, curName - ты

case "unfriend":

var str string

fmt.Print("enter name: ")

str = input.Gets()

name = str

send\_request(encoder, "unfriend", Elem{MyAddr, name, curName})

case "list":

name = ""

for \_, e := range conns {

send\_request(e.encoder, "list", Elem{MyAddr, name, curName})

var resp Response

if err := e.decoder.Decode(&resp); err != nil {

fmt.Printf("error: %v\n", err)

break

}

var elem Elem

if err := json.Unmarshal(\*resp.Data, &elem); err != nil {

fmt.Printf("error: malformed data field in response\n")

} else {

fmt.Printf("friends of "+ elem.Str +"are " + elem.Name + ", ")

}

}

default:

fmt.Printf("error: unknown command\n")

//continue

}

}

}

// handleRequest - метод обработки запроса от пира. //???

func (peer \*Peer) handleRequest(req \*Request) {

errorMsg := ""

switch req.Command {

case "friend":

fmt.Printf("this peer wants to be friends with you\n")

///fmt.Printf("accept friendship? y/n")

//var ans string

//fmt.Scan(&ans)

//if ans == "y" {

elem := new(Elem)

if err := json.Unmarshal(\*req.Data, elem); err != nil {

errorMsg = "malformed data field"

} else {

friends[board[elem.Name]][board[elem.Str]] = 1

friends[board[elem.Str]][board[elem.Name]] = 1

fmt.Printf("we`ve become friends\n")

}

//}

if errorMsg == "" {

peer.logger.Info("friend peer friends succesfully")

} else {

peer.logger.Error("friend peer failed", "reason", errorMsg)

}

case "unfriend":

fmt.Printf("this peer wants to ruin friendship\n")

//fmt.Printf("agree? y/n")

//var answ string

//fmt.Scan(&answ)

//if answ == "y" {

elem := new(Elem)

if err := json.Unmarshal(\*req.Data, elem); err != nil {

errorMsg = "malformed data field"

} else {

friends[board[elem.Name]][board[elem.Str]] = 0

friends[board[elem.Str]][board[elem.Name]] = 0

fmt.Printf("we`ve become strangers\n")

}

//}

if errorMsg == "" {

peer.logger.Info("unfirend peer friends succesfully")

} else {

peer.logger.Error("unfriend peer failed", "reason", errorMsg)

}

case "list":

errorMsg := ""

if req.Data == nil {

errorMsg = "data field is absent"

} else {

el := new(Elem)

if err := json.Unmarshal(\*req.Data, el); err != nil {

errorMsg = "malformed data field"

}

if name == "" {

fmt.Printf("lets write friends\n")

for i := 0; i < 5; i ++ {

if friends[board[el.Str]][i] == 1 {

peer.respond(Elem{el.IP, fmt.Sprintf("%s:%s:\n", board1[i], MyAddr,), el.Str})}

}

}

}

if errorMsg == "" {

peer.logger.Info("list peer friends succesfully")

} else {

peer.logger.Error("list peer failed", "reason", errorMsg)

}

default:

peer.logger.Error("unknown command")

}

}

// respond - вспомогательный метод для передачи ответа с данными.

// Данные могут быть пустыми (data == nil).

func (peer \*Peer) respond(data interface{}) {

var raw json.RawMessage

raw, \_ = json.Marshal(data)

peer.enc.Encode(&Response{&raw})

}

// serve - метод, в котором реализован цикл взаимодействия с пиром.

// будет вызаваться в отдельной go-программе.

func (peer \*Peer) serve() {

defer peer.conn.Close()

for {

var req Request

if err := peer.dec.Decode(&req); err != nil {

peer.logger.Error("cannot decode message", "reason", err)

break

} else {

peer.logger.Info("shutting down connection")

peer.handleRequest(&req)

}

}

}

func listen(addrStr string) {

var listener \*net.TCPListener

defer listener.Close() //чтобы прослушать много запросов

// Разбор адреса, строковое представление которого находится в переменной addr.

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 go-программы для обслуживания клиентов.

go NewPeer(conn).serve()

}

}

}

}

}

// conStruct - структура для каждой связи между двумя пирами

type conStruct struct{

conn \*net.TCPConn

encoder \*json.Encoder

decoder \*json.Decoder

}

func main() {

//var ToAddr string

var ports [5]string

ports[0] = "127.0.0.1:6000"

ports[1] = "127.0.0.1:6061"

ports[2] = "127.0.0.1:6062"

ports[3] = "127.0.0.1:6063"

ports[4] = "127.0.0.1:6064"

//giveNames()

board = make(map[string]int)

board["John"] = 0

board["Alice"] = 1

board["Bob"] = 2

board["Peter"] = 3

board["Ann"] = 4

board1 = make(map[int]string)

board1[0] = "John"

board1[1] = "Alice"

board1[2] = "Bob"

board1[3] = "Peter"

board1[4] = "Ann"

var conns []conStruct //массив связей между пирами

var port string

fmt.Scan(&port)

MyAddr = fmt.Sprintf("%s:%s", localhost, port)

var strname string

fmt.Printf("Enter your name")

strname = input.Gets()

go listen(MyAddr)

//var i int

//fmt.Printf("Your peer number is: ")

//fmt.Scan(&i)

//fmt.Printf("Your name is: %s\n", ppl[i])

//MyAddr = ports[i]

var str string

for {

fmt.Println("Do yo want to continue work with this peer? yes/no")

fmt.Scan(&str)

if str == "yes" {

break

}

}

for {

for \_, ToAddr := range ports {

if ToAddr != MyAddr{

if addr, err := net.ResolveTCPAddr("tcp", ToAddr); err != nil {

fmt.Printf("error: %v\n", err)

} else if conn, err := net.DialTCP("tcp", nil, addr); err == nil {

encoder = json.NewEncoder(conn)

decoder = json.NewDecoder(conn)

conns = append(conns, conStruct{conn, encoder, decoder})

}

}

}

interact(conns, strname)

}

}