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 № 3

по курсу «Компьютерные сети»

«Протокол одноранговой сети»

Студент группы ИУ9-31Б Горбунов А. Д.

Преподаватель Посевин Д. П.

Москва 2023

1 Задание

Адресная книга (полносвязная)

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 принимает команды – добавление записи адресной книги (вида «фамилия–e-mail»), удаление записи и вывод списка записей; пиры должны обмениваться записями, чтобы у всех была одинаковая адресная книга.

2 Результаты

Исходный код программы представлен в листинге 1, 2, 3, 4

Листинг 1 — client.go

```
package main
import (
    "encoding/json"
    "fmt"
    "log"
    "net"
    "os"
    "os/signal"
    "time"
)
type AddressBook struct {
 Records []Record `json:"records"`
}
type Record struct {
 Name  string `json:"name"`
 Email string `json:"email"`
}
type SupType struct{
    supType ForJSON `json:"suptype"`
}
type ForJSON struct{
    forJSON Record `json:"forjson"`
    oper string `json:"oper"`
    index int `json:"index"`
    len int `json:"len"`
 }
type Peer struct {
 IP   string
 Port int
}
```

Листинг 2 — client.go(продолжение 1)

```
1  func main() {
2    ip := getIPAddress()
3    port := getPort()
4    neighbors := getNeighbors()
5    addressBook := AddressBook{}
6    go startServer(ip, port, neighbors, &addressBook)
7    go startCommandServer(&addressBook, port)
8    c := make(chan os.Signal, 1)
9    signal.Notify(c, os.Interrupt)
10   <-c
11   log.Println("Exiting...")
12 }
13 func getIPAddress() string {
14   conn, err := net.Dial("udp", "8.8.8.8:80")
15   if err != nil {
16     log.Fatal(err)
17   }
18   defer conn.Close()
19   localAddr := conn.LocalAddr().(*net.UDPAddr)
20
21   return localAddr.IP.String()
22 }
23 func getPort() int {
24   port := 2222
25   fmt.Print("Port: ")
26   fmt.Scan(&port)
27   address := fmt.Sprintf("localhost:%d", port)
28   l, err := net.Listen("tcp", address)
29   if err != nil {
30     log.Fatal(err)
31   }
32   defer l.Close()
33   localAddr := l.Addr().(*net.TCPAddr)
34   return localAddr.Port
35 }
36 func getNeighbors() []Peer {
37   neighbors := []Peer{{getIPAddress(), 1111}, {getIPAddress(), 2222}, {
       getIPAddress(), 3333}}
38   return neighbors
39 }
40 func startServer(ip string, port int, neighbors []Peer, addressBook *
       AddressBook) {
41   address := fmt.Sprintf("%s:%d", ip, port)
42   listener, err := net.Listen("tcp", address)
43   if err != nil {
44     log.Fatal(err)
45   }
46   defer listener.Close()
47   log.Printf("Server started and listening on %s\n", address)
48   for {
49     conn, err := listener.Accept()
50     if err != nil {
51       log.Fatal(err)
52     }
53     go handleConnection(conn, addressBook)
54   }
55 }
```

Листинг 3 — client.go(продолжение 2)

```
func handleConnection(conn net.Conn, addressBook *AddressBook) {
  defer conn.Close()
  log.Println("New connection established")
  message := make([]byte, 4096)
  n, err := conn.Read(message)
  if err != nil {
    log.Println(err)
    return
  }
  var record AddressBook
  err = json.Unmarshal(message[:n], &record)

  if err != nil {
    log.Println(err)
    return
  }
  addressBook.Records = record.Records
}

func sendAddressBookToNeighbors(addressBook *AddressBook, port int) {
  jsonData, err := json.Marshal(addressBook)
  if err != nil {
    log.Println(err)
    return
  }
  for _, neighbor := range getNeighbors() {
    if neighbor != ([]Peer{{getIPAddress(), port}})[0] {
      go sendAddressBook(neighbor, jsonData)
    }
  }
}

func sendAddressBook(neighbor Peer, jsonData []byte) {
  address := fmt.Sprintf("%s:%d", neighbor.IP, neighbor.Port)
  conn, err := net.DialTimeout("tcp", address, time.Second*5)
  if err != nil {
    log.Printf("Failed to connect to neighbor %s: %v\n", address, err)
    return
  }
  defer conn.Close()

  _, err = conn.Write(jsonData)
  if err != nil {
    log.Println(err)
    return
  }

  log.Printf("Sent address book to neighbor %s\n", address)
}
```

Листинг 4 — client.go

```
1  func startCommandServer(addressBook *AddressBook, port int) {
2    for {
3      var command string
4      fmt.Print("Enter command (add, remove, list): ")
5      fmt.Scanln(&command)
6
7      switch command {
8      case "add":
9        addRecord(addressBook, port)
10     case "remove":
11       removeRecord(addressBook, port)
12     case "list":
13       listRecords(addressBook)
14     default:
15       log.Println("Invalid command")
16     }
17   }
18 }
19 func addRecord(addressBook *AddressBook, port int) {
20   var name, email string
21   fmt.Print("Enter name: ")
22   fmt.Scanln(&name)
23   fmt.Print("Enter email: ")
24   fmt.Scanln(&email)
25
26   record := Record{Name: name, Email: email}
27   addressBook.Records = append(addressBook.Records, record)
28
29   sendAddressBookToNeighbors(addressBook, port)
30 }
31
32 func removeRecord(addressBook *AddressBook, port int) {
33   var index int
34   fmt.Print("Enter the index of the record to remove: ")
35   fmt.Scanln(&index)
36
37   if index >= 0 && index < len(addressBook.Records) {
38     addressBook.Records = append(addressBook.Records[:index],
     addressBook.Records[index+1:]...)
39
40     sendAddressBookToNeighbors(addressBook, port)
41   } else {
42     log.Println("Invalid index")
43   }
44 }
45
46 func listRecords(addressBook *AddressBook) {
47   for i, record := range addressBook.Records {
48     fmt.Printf("%d. %s - %s\n", i, record.Name, record.Email)
49   }
50 }
```

Результат запуска представлен на рисунке 1, 2

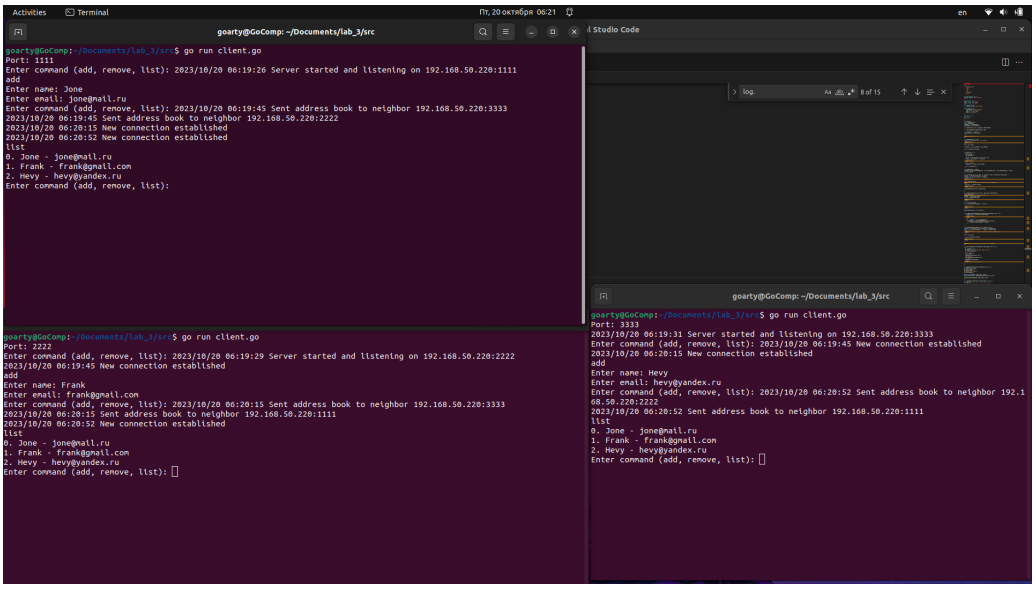

Рис. 1 — Пример ввода контактов

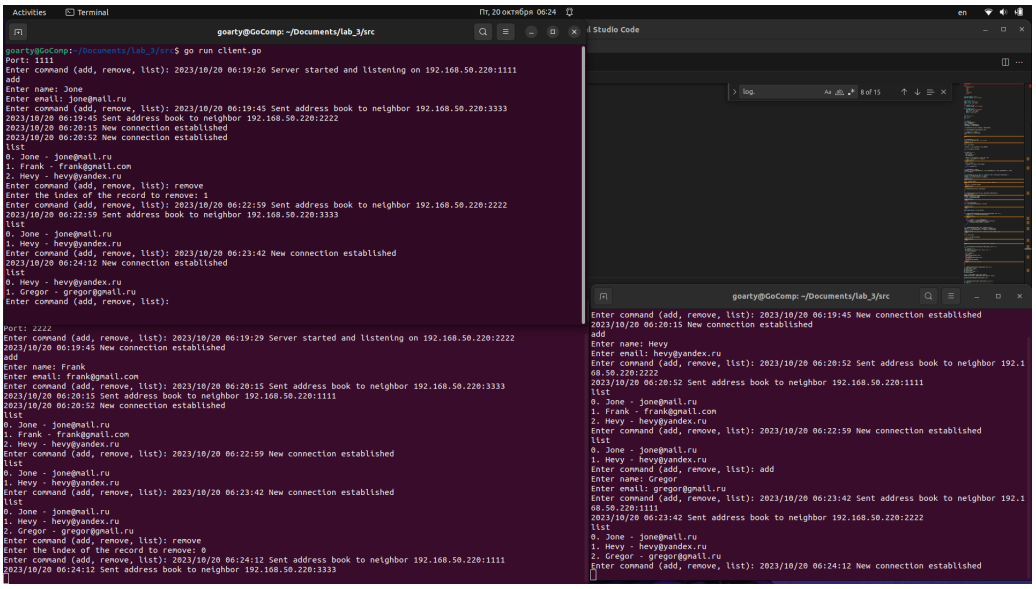

Рис. 2 — Пример удаления контактов