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 ИУ9-32Б Волохов А. В.

Преподаватель Посевин Д. П.

Москва 2023

1 Задание

Целью данной работы является разработка одноранговой сетевой службы. Распределённая хеш-таблица (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 – добавить пару «ключ–значение», удалить пару по ключу, найти значение по ключу. Замечание: все словарные пары доступны всем пирам; позже добавленные пары должны замещать ранее добавленные пары с тем же ключом Исходный код программы представлен в листингах 1– 2– 3– 4– 5– 6.

Листинг 1 — acces.go

```go
package main

import (
  "bufio"
  "encoding/json"
  "fmt"
  "net"
  "os"
  "strings"
)

type Message struct {
  Command string `json:"command"`
  Key     string `json:"key"`
  Value   string `json:"value"`
  IP      string `json:"ip"`
  Port    string `json:"port"`
  Forward bool   `json:"forward"`
}

func main() {
  var peerAddress string
  var conn net.Conn
  var err error
  scanner := bufio.NewScanner(os.Stdin)
  for {
    fmt.Print("Enter a command: ")
    scanner.Scan()
    command := scanner.Text()

    if command == "exit" {
      break
    }

    parts := strings.Fields(command)
    if parts[0] == "PEER" && len(parts) == 2 {
      peerAddress = parts[1]
      conn, err = net.Dial("tcp", peerAddress)
      if err != nil {
        fmt.Printf("Error: %s\n", err)
        continue
      }
      fmt.Println("Connected to peer:", peerAddress)
    } else if parts[0] == "ADD_NEIGHBOR" && len(parts) == 3 {
      msg := Message{
        Command: "ADD_NEIGHBOR",
        IP:      parts[1],
        Port:    parts[2],
      }

      msgBytes, err := json.Marshal(msg)
      if err != nil {
        fmt.Printf("Error encoding message: %s\n", err)
        continue
      }
      _, err = conn.Write(append(msgBytes, '\n'))
      if err != nil {
        fmt.Printf("Error sending message: %s\n", err)
      }
    }
```

Листинг 2 — acces.go - продолжение

```
else if conn != nil {
        msg := Message{
          Command: parts[0],
          Key:     "",
          Value:   "",
        }

        if len(parts) > 1 {
          msg.Key = parts[1]
        }

        if len(parts) > 2 {
          msg.Value = parts[2]
        }

        msgBytes, err := json.Marshal(msg)
        if err != nil {
          fmt.Printf("Error encoding message: %s\n", err)
          continue
        }
        _, err = conn.Write(append(msgBytes, '\n'))
        if err != nil {
          fmt.Printf("Error sending message: %s\n", err)
        }
      } else {
        fmt.Println("You must first connect to a peer using 'PEER <port>'"
    )
      }
    }

    if conn != nil {
      err := conn.Close()
      if err != nil {
        return
      }
    }
}
```

Листинг 3 — peer.go

```go
package main

import (
  "bufio"
  "encoding/json"
  "fmt"
  "net"
  "os"
  "sync"
)

type HashTable struct {
  data  map[string]string
  mutex sync.Mutex
}

type Neighbor struct {
  IP   string
  Port string
}

type Message struct {
  Command string `json:"command"`
  Key     string `json:"key"`
  Value   string `json:"value"`
  IP      string `json:"ip"`
  Port    string `json:"port"`
  Forward bool   `json:"forward"`
}

func (h *HashTable) Add(key, value string) {
  h.mutex.Lock()
  defer h.mutex.Unlock()
  h.data[key] = value
}

func (h *HashTable) Delete(key string) {
  h.mutex.Lock()
  defer h.mutex.Unlock()
  delete(h.data, key)
}

func (h *HashTable) Find(key string) string {
  h.mutex.Lock()
  defer h.mutex.Unlock()
  if val, found := h.data[key]; found {
    return val
  }
  return "Key not found"
}
```

Листинг 4 — peer.go - продолжение

```
1  func main() {
2    //                                                    -
3    hashTable := &HashTable{
4      data: make(map[string]string),
5    }
6
7    //
8    neighbors := make([]Neighbor, 0)
9
10   //                                                              -
                ,
11   if len(os.Args) != 2 {
12     fmt.Println("Usage: go run peer.go <port>")
13     return
14   }
15
16   //
17   port := os.Args[1]
18
19   //
20   listener, err := net.Listen("tcp", ":"+port)
21   if err != nil {
22     fmt.Printf("Error: %s\n", err)
23     return
24   }
25   defer func(listener net.Listener) {
26     err := listener.Close()
27     if err != nil {
28
29     }
30   }(listener)
31
32   fmt.Printf("Peer listening on port %s\n", port)
33
34   for {
35     conn, err := listener.Accept()
36     if err != nil {
37       fmt.Printf("Error: %s\n", err)
38       continue
39     }
40
41     go handleConnection(conn, hashTable, &neighbors)
42   }
43 }
44
45 func handleConnection(conn net.Conn, hashTable *HashTable, neighbors *[]
       Neighbor) {
46   defer func(conn net.Conn) {
47     err := conn.Close()
48     if err != nil {
49
50     }
51   }(conn)
52
53   //                      IP
```

Листинг 5 — peer.go - продолжение

```
remoteAddr := conn.RemoteAddr().String()
  fmt.Printf("Connected to peer at %s\n", remoteAddr)

  scanner := bufio.NewScanner(conn)
  for scanner.Scan() {
    command := scanner.Text()

    //                          JSON-
    var msg Message
    err := json.Unmarshal([]byte(command), &msg)
    if err != nil {
      fmt.Printf("Error decoding message: %s\n", err)
      continue
    }

    switch msg.Command {
    case "ADD":
      if msg.Key != "" && msg.Value != "" {
        hashTable.Add(msg.Key, msg.Value)
        fmt.Printf("Added key-value pair: %s-%s\n", msg.Key, msg.Value)
        if !msg.Forward {
          //                                                    ,

          for _, neighbor := range *neighbors {
            if neighbor.IP != msg.IP || neighbor.Port != msg.Port {
              sendRequest(neighbor.IP, neighbor.Port, msg)
            }
          }
        }
      }
    case "DELETE":
      if msg.Key != "" {
        hashTable.Delete(msg.Key)
        fmt.Printf("Deleted key: %s\n", msg.Key)
        if !msg.Forward {
          //                                                    ,

          for _, neighbor := range *neighbors {
            if neighbor.IP != msg.IP || neighbor.Port != msg.Port {
              sendRequest(neighbor.IP, neighbor.Port, msg)
            }
          }
        }
      }
    case "FIND":
      if msg.Key != "" {
        value := hashTable.Find(msg.Key)
        fmt.Printf("Found value: %s\n", value)
      }
    case "LIST_NEIGHBORS":
      //

      fmt.Println("List of neighbors:")
      for _, neighbor := range *neighbors {
        fmt.Printf("IP: %s, Port: %s\n", neighbor.IP, neighbor.Port)
      }
```

Листинг 6 — peer.go - продолжение

```
case "ADD_NEIGHBOR":
        //
        if msg.IP != "" && msg.Port != "" {
          *neighbors = append(*neighbors, Neighbor{IP: msg.IP, Port: msg.
    Port})
          fmt.Printf("Added neighbor: %s:%s\n", msg.IP, msg.Port)
        }
     }
   }
}

func sendRequest(ip, port string, msg Message) {
  msg.Forward = true
  msg.IP = ""
  msg.Port = ""
  neighborAddr := ip + ":" + port
  conn, err := net.Dial("tcp", neighborAddr)
  if err != nil {
    fmt.Printf("Error connecting to neighbor %s: %s\n", neighborAddr,
    err)
    return
  }
  defer func(conn net.Conn) {
    err := conn.Close()
    if err != nil {

    }
  }(conn)
  msgBytes, err := json.Marshal(msg)
  if err != nil {
    fmt.Printf("Error encoding message: %s\n", err)
    return
  }
  _, err = conn.Write(append(msgBytes, '\n'))
  if err != nil {
    fmt.Printf("Error sending message to neighbor %s: %s\n",
    neighborAddr, err)
    return
  }
}
```

```
Enter a command: PEER 185.139.70.64:0303
Connected to peer: 185.139.70.64:0303
Enter a command: ADD 1 A
Enter a command: DELETE 1
Enter a command: ADD 2 B
Enter a command: PEER 185.255.133.113:0303
Connected to peer: 185.255.133.113:0303
Enter a command: FIND 2
Enter a command: ADD 3 ABC
Enter a command: PEER 185.139.70.64:0303
Connected to peer: 185.139.70.64:0303
Enter a command: FIND 3
Enter a command: DELETE 3
Enter a command:
```

Рис. 1 — Клиент

Рис. 2 — Сервер