



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

```

1 package main
2
3 import (
4     "bufio"
5     "encoding/json"
6     "fmt"
7     "net"
8     "os"
9     "strings"
10 )
11
12 type Message struct {
13     Command string 'json:"command" '
14     Key      string 'json:"key" '
15     Value    string 'json:"value" '
16     IP       string 'json:"ip" '
17     Port     string 'json:"port" '
18     Forward  bool   'json:"forward" '
19 }
20
21 func main() {
22     var peerAddress string
23     var conn net.Conn
24     var err error
25     scanner := bufio.NewScanner(os.Stdin)
26     for {
27         fmt.Print("Enter a command: ")
28         scanner.Scan()
29         command := scanner.Text()
30
31         if command == "exit" {
32             break
33         }
34
35         parts := strings.Fields(command)
36         if parts[0] == "PEER" && len(parts) == 2 {
37             peerAddress = parts[1]
38             conn, err = net.Dial("tcp", peerAddress)
39             if err != nil {
40                 fmt.Printf("Error: %s\n", err)
41                 continue
42             }
43             fmt.Println("Connected to peer:", peerAddress)
44         } else if parts[0] == "ADD_NEIGHBOR" && len(parts) == 3 {
45             msg := Message{
46                 Command: "ADD_NEIGHBOR",
47                 IP:      parts[1],
48                 Port:    parts[2],
49             }
50
51             msgBytes, err := json.Marshal(msg)
52             if err != nil {
53                 fmt.Printf("Error encoding message: %s\n", err)
54                 continue
55             }
56             _, err = conn.Write(append(msgBytes, '\n'))
57             if err != nil {
58                 fmt.Printf("Error sending message: %s\n", err)
59             }
60         }

```

Листинг 2 — acces.go - продолжение

```
1 else if conn != nil {
2     msg := Message{
3         Command: parts[0],
4         Key:     "",
5         Value:    "",
6     }
7
8     if len(parts) > 1 {
9         msg.Key = parts[1]
10    }
11
12    if len(parts) > 2 {
13        msg.Value = parts[2]
14    }
15
16    msgBytes, err := json.Marshal(msg)
17    if err != nil {
18        fmt.Printf("Error encoding message: %s\n", err)
19        continue
20    }
21    _, err = conn.Write(append(msgBytes, '\n'))
22    if err != nil {
23        fmt.Printf("Error sending message: %s\n", err)
24    }
25    else {
26        fmt.Println("You must first connect to a peer using 'PEER <port>'")
27    }
28 }
29
30 if conn != nil {
31     err := conn.Close()
32     if err != nil {
33         return
34     }
35 }
36 }
```

Листинг 3 — peer.go

```
1 package main
2
3 import (
4     "bufio"
5     "encoding/json"
6     "fmt"
7     "net"
8     "os"
9     "sync"
10 )
11
12 type HashTable struct {
13     data map[string]string
14     mutex sync.Mutex
15 }
16
17 type Neighbor struct {
18     IP string
19     Port string
20 }
21
22 type Message struct {
23     Command string `json:"command"`
24     Key string `json:"key"`
25     Value string `json:"value"`
26     IP string `json:"ip"`
27     Port string `json:"port"`
28     Forward bool `json:"forward"`
29 }
30
31 func (h *HashTable) Add(key, value string) {
32     h.mutex.Lock()
33     defer h.mutex.Unlock()
34     h.data[key] = value
35 }
36
37 func (h *HashTable) Delete(key string) {
38     h.mutex.Lock()
39     defer h.mutex.Unlock()
40     delete(h.data, key)
41 }
42
43 func (h *HashTable) Find(key string) string {
44     h.mutex.Lock()
45     defer h.mutex.Unlock()
46     if val, found := h.data[key]; found {
47         return val
48     }
49     return "Key not found"
50 }
```

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
12    if len(os.Args) != 2 {
13        fmt.Println("Usage: go run peer.go <port>")
14        return
15    }
16    //
17
18    port := os.Args[1]
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 *[]
46     Neighbor) {
47     defer func(conn net.Conn) {
48         err := conn.Close()
49         if err != nil {
50
51         }
52     }(conn)
53     //

```

IP

Листинг 5 — peer.go - продолжение

```

1 remoteAddr := conn.RemoteAddr().String()
2 fmt.Printf("Connected to peer at %s\n", remoteAddr)
3
4 scanner := bufio.NewScanner(conn)
5 for scanner.Scan() {
6     command := scanner.Text()
7
8     // JSON-
9     var msg Message
10    err := json.Unmarshal([]byte(command), &msg)
11    if err != nil {
12        fmt.Printf("Error decoding message: %s\n", err)
13        continue
14    }
15
16    switch msg.Command {
17    case "ADD":
18        if msg.Key != "" && msg.Value != "" {
19            hashTable.Add(msg.Key, msg.Value)
20            fmt.Printf("Added key-value pair: %s-%s\n", msg.Key, msg.Value)
21            if !msg.Forward {
22                //
23
24                for _, neighbor := range *neighbors {
25                    if neighbor.IP != msg.IP || neighbor.Port != msg.Port {
26                        sendRequest(neighbor.IP, neighbor.Port, msg)
27                    }
28                }
29            }
30        case "DELETE":
31            if msg.Key != "" {
32                hashTable.Delete(msg.Key)
33                fmt.Printf("Deleted key: %s\n", msg.Key)
34                if !msg.Forward {
35                    //
36
37                    for _, neighbor := range *neighbors {
38                        if neighbor.IP != msg.IP || neighbor.Port != msg.Port {
39                            sendRequest(neighbor.IP, neighbor.Port, msg)
40                        }
41                    }
42                }
43            case "FIND":
44                if msg.Key != "" {
45                    value := hashTable.Find(msg.Key)
46                    fmt.Printf("Found value: %s\n", value)
47                }
48            case "LIST_NEIGHBORS":
49                //
50
51                fmt.Println("List of neighbors:")
52                for _, neighbor := range *neighbors {
53                    fmt.Printf("IP: %s, Port: %s\n", neighbor.IP, neighbor.Port)

```

Листинг 6 — peer.go - продолжение

```
1 case "ADD_NEIGHBOR":
2     //
3     if msg.IP != "" && msg.Port != "" {
4         *neighbors = append(*neighbors, Neighbor{IP: msg.IP, Port: msg.
5             Port})
6         fmt.Printf("Added neighbor: %s:%s\n", msg.IP, msg.Port)
7     }
8 }
9 }
10
11 func sendRequest(ip, port string, msg Message) {
12     msg.Forward = true
13     msg.IP = ""
14     msg.Port = ""
15     neighborAddr := ip + ":" + port
16     conn, err := net.Dial("tcp", neighborAddr)
17     if err != nil {
18         fmt.Printf("Error connecting to neighbor %s: %s\n", neighborAddr,
19             err)
20         return
21     }
22     defer func(conn net.Conn) {
23         err := conn.Close()
24         if err != nil {
25             }
26     }(conn)
27     msgBytes, err := json.Marshal(msg)
28     if err != nil {
29         fmt.Printf("Error encoding message: %s\n", err)
30         return
31     }
32     _, err = conn.Write(append(msgBytes, '\n'))
33     if err != nil {
34         fmt.Printf("Error sending message to neighbor %s: %s\n",
35             neighborAddr, err)
36         return
37     }
38 }
```



```

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

```

Peer listening on port 0303
Connected to peer at 195.19.42.179:42882
Added neighbor: 185.104.249.105:0303
List of neighbors:
IP: 185.104.249.105, Port: 0303
Added neighbor: 185.255.133.113:0303
Connected to peer at 195.19.42.179:47828
List of neighbors:
IP: 185.104.249.105, Port: 0303
IP: 185.255.133.113, Port: 0303
Connected to peer at 195.19.42.179:48736
Added key-value pair: 1-A
Deleted key: 1
Added key-value pair: 2-B
Connected to peer at 185.255.133.113:37652
Added key-value pair: 3-ABC
Connected to peer at 195.19.42.179:46636
Found value: ABC
Deleted key: 3

root@letuchkal:~/IU9-328/Volokhov/lab3# go run peer.go 0303
Peer listening on port 0303
Connected to peer at 195.19.42.179:48164
Added neighbor: 185.139.70.64:0303
Added neighbor: 185.255.133.113:0303
Connected to peer at 195.19.42.179:56012
List of neighbors:
IP: 185.139.70.64, Port: 0303
IP: 185.255.133.113, Port: 0303
Connected to peer at 185.139.70.64:53240
Added key-value pair: 1-A
Connected to peer at 185.139.70.64:54586
Deleted key: 1
Connected to peer at 185.139.70.64:38318
Added key-value pair: 2-B
Connected to peer at 185.255.133.113:56654
Added key-value pair: 3-ABC
Connected to peer at 185.139.70.64:60752
Deleted key: 3

Added neighbor: 185.139.70.64:0303
Added neighbor: 185.104.249.105:0303
Connected to peer at 195.19.42.179:54028
List of neighbors:
IP: 185.139.70.64, Port: 0303
IP: 185.104.249.105, Port: 0303
Connected to peer at 185.139.70.64:40642
Added key-value pair: 1-A
Connected to peer at 185.139.70.64:54114
Deleted key: 1
Connected to peer at 185.139.70.64:51192
Added key-value pair: 2-B
Connected to peer at 195.19.42.179:52854
Found value: B
Added key-value pair: 3-ABC
Connected to peer at 185.139.70.64:47640
Deleted key: 3

```

Рис. 2 — Сервер