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 Тараканов В. Д..

Преподаватель Посевин Д. П.

Москва 2024

1 Задание

N неубиваемых процессов Топология: полносвязная. Информация, известная пиру при запуске: его порт, а также порты N-1 возможных соседей. Описание службы: все пиры работают на одном узле; если какой-то пир не находит какого-то соседа, он сам его запускает. Замечание: для отладки нужно «убивать» процессы пиров по одному и смотреть, как вместо них появляются новые процессы.

2 Результаты

Исходный код программы представлен в листингах 1– ??.

Листинг 1: main.go

```
1  package main
2
3  import (
4  "encoding/json"
5  "fmt"
6  "log"
7  "net"
8  "net/http"
9  "sync"
10 "time"
11
12 "github.com/gorilla/websocket"
13 )
14
15 const numPeers = 4
16
17 var peerPorts = []int{8081, 8082, 8083, 8084}
18
19 type PeerStatus struct {
20   Port   int    `json:"port"`
21   Status string `json:"status"`
22 }
23
24 type Peer struct {
25   Port       int
26   Status     string
27   killChan   chan bool
28   statusLock sync.Mutex
```

```
29      }
30
31      var peers = make(map[int]*Peer)
32
33      var statusUpdates = make(chan []PeerStatus)
34
35      var upgrader = websocket.Upgrader{
36        CheckOrigin: func(r *http.Request) bool {
37          return true
38        },
39      }
40
41      func main() {
42        for _, port := range peerPorts {
43          peers[port] = &Peer{
44            Port:     port,
45            Status:   "alive",
46            killChan: make(chan bool),
47          }
48          go startPeer(peers[port])
49        }
50
51        go monitorPeers()
52
53        http.Handle("/", http.FileServer(http.Dir("./public")))
54        http.HandleFunc("/ws", handleWebSocket)
55        http.HandleFunc("/kill", handleKillPeer)
56
57        go func() {
58          for {
59            var statuses []PeerStatus
60            for _, peer := range peers {
61              peer.statusLock.Lock()
62              statuses = append(statuses, PeerStatus{
63                Port:   peer.Port,
64                Status: peer.Status,
65              })
66              peer.statusLock.Unlock()
67            }
68            statusUpdates <- statuses
69            time.Sleep(2 * time.Second)
70          }
71        }()
72
73        log.Println("Start server on port 8000")
74        log.Fatal(http.ListenAndServe(":8000", nil))
```

```
75      }
76
77      func startPeer(peer *Peer) {
78        peer.statusLock.Lock()
79        peer.Status = "alive"
80        peer.statusLock.Unlock()
81
82        mux := http.NewServeMux()
83        mux.HandleFunc("/status", func(w http.ResponseWriter, r *http.
      Request) {
84          peer.statusLock.Lock()
85          defer peer.statusLock.Unlock()
86          fmt.Fprintf(w, "Peer %d: %s", peer.Port, peer.Status)
87        })
88
89        server := &http.Server{
90          Addr:    fmt.Sprintf(":%d", peer.Port),
91          Handler: mux,
92        }
93
94        go func() {
95          log.Printf("Peer %d started\n", peer.Port)
96          if err := server.ListenAndServe(); err != nil && err != http.
      ErrServerClosed {
97            log.Printf("Peer %d stopped\n", peer.Port)
98          }
99        }()
100
101       <-peer.killChan
102
103       peer.statusLock.Lock()
104       peer.Status = "dead"
105       peer.statusLock.Unlock()
106       log.Printf("Peer %d was killed\n", peer.Port)
107       server.Close()
108     }
109
110     func monitorPeers() {
111       for {
112         for _, peer := range peers {
113           go func(p *Peer) {
114             if p.Status == "dead" {
115
116               log.Printf("Restart peer %d\n", p.Port)
117               go startPeer(p)
118             } else {
```

```
119
120             conn, err := net.DialTimeout("tcp", fmt.Sprintf("localhost
    :%d", p.Port), time.Second)
121             if err != nil {
122               log.Printf("Peer %d is not available, pointed as dead\n
    ", p.Port)
123               p.statusLock.Lock()
124               p.Status = "dead"
125               p.statusLock.Unlock()
126             } else {
127               conn.Close()
128             }
129           }
130         }(peer)
131       }
132       time.Sleep(5 * time.Second)
133     }
134   }
135
136   func handleWebSocket(w http.ResponseWriter, r *http.Request) {
137     conn, err := upgrader.Upgrade(w, r, nil)
138     if err != nil {
139       log.Println("Error when upgrade WebSocket:", err)
140       return
141     }
142     defer conn.Close()
143
144     for statuses := range statusUpdates {
145       if err := conn.WriteJSON(statuses); err != nil {
146         log.Println("Error when sending json to WebSocket:", err)
147         break
148       }
149     }
150   }
151
152   func handleKillPeer(w http.ResponseWriter, r *http.Request) {
153     if r.Method != http.MethodPost {
154       http.Error(w, "Method not available", http.
    StatusMethodNotAllowed)
155       return
156     }
157
158     type KillRequest struct {
159       Port int `json:"port"`
160     }
161
```

```
162         var killReq KillRequest
163         err := json.NewDecoder(r.Body).Decode(&killReq)
164         if err != nil {
165           http.Error(w, "Wrong format request", http.StatusBadRequest)
166           return
167         }
168
169         peer, exists := peers[killReq.Port]
170         if !exists {
171           http.Error(w, "Peer not found", http.StatusNotFound)
172           return
173         }
174
175         go func(p *Peer) {
176           p.killChan <- true
177         }(peer)
178
179         w.WriteHeader(http.StatusOK)
180       }
181
182
```

Листинг 2: app.js

```
1   const peersContainer = document.getElementById("peers-container");
2
3
4   const socket = new WebSocket(`ws://${window.location.host}/ws`);
5
6   socket.onmessage = function(event) {
7     const data = JSON.parse(event.data);
8     renderPeers(data);
9   };
10
11  function renderPeers(peers) {
12    peersContainer.innerHTML = "";
13
14    peers.forEach(peer => {
15      const peerDiv = document.createElement("div");
16      peerDiv.className = "peer";
17
18      const statusSpan = document.createElement("span");
19      statusSpan.textContent = `Peer ${peer.port}: `;
20      statusSpan.className = peer.status === "dead" ? "alive" : "dead
    ";
21
22      const statusText = document.createElement("span");
```

```
23          statusText.textContent = peer.status;
24
25          const killButton = document.createElement("button");
26          killButton.textContent = "Kill";
27          killButton.disabled = peer.status !== "alive";
28          killButton.onclick = () => killPeer(peer.port);
29
30          peerDiv.appendChild(statusSpan);
31          peerDiv.appendChild(statusText);
32          peerDiv.appendChild(document.createTextNode(" "));
33          peerDiv.appendChild(killButton);
34
35          peersContainer.appendChild(peerDiv);
36        });
37      }
38
39      function killPeer(port) {
40        fetch("/kill", {
41          method: "POST",
42          headers: {
43            "Content-Type": "application/json"
44          },
45          body: JSON.stringify({ port: port })
46        })
47        .then(response => {
48          if (response.ok) {
49            console.log(`Peer ${port} was killed`);
50          } else {
51            console.error("Error when killing port");
52          }
53        })
54        .catch(error => {
55          console.error("Error when sending request:", error);
56        });
57      }
58
```

Листинг 3: index.html

```
1       <!DOCTYPE html>
2       <html lang="ru">
3       <head>
4       <meta charset="UTF-8">
5       <title>Peer monitor</title>
6       <style>
7       body {
8         font-family: Arial, sans-serif;
```

```
9       }
10      .peer {
11        margin-bottom: 10px;
12      }
13      .alive {
14        color: green;
15      }
16      .dead {
17        color: red;
18      }
19      </style>
20      </head>
21      <body>
22      <h1>Peer state</h1>
23      <div id="peers-container"></div>
24
25      <script src="/app.js"></script>
26      </body>
27      </html>
28
```

Результат запуска представлен на рисунке 1– 2.

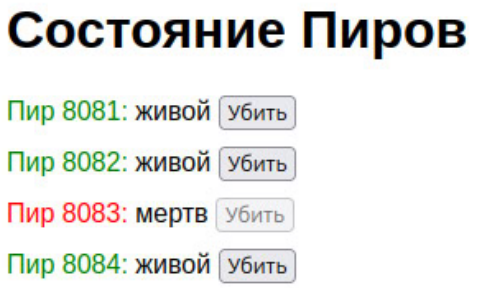

Рис. 1 — Результат

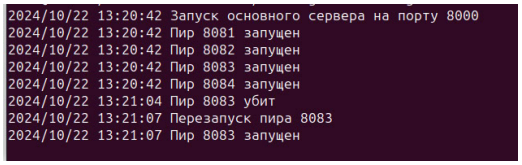

Рис. 2 — Результат