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 № 3.2

по курсу «Компьютерные сети»

«Протокол одноранговой сети»

Студент группы ИУ9-32Б Лавров Р. Д.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

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

Основные требования к сетевой службе:

1. в качестве формата сообщений для протокола взаимодействия пиров нужно использовать JSON;

2. полная проверка данных, получаемых из сети;

3. устойчивость к обрыву соединения;

4. ведение подробного лога всех ошибок, а также других важных событий (установка и завершение соединения с соседним пиром, приём и передача сообщений, и т.п.).

Документация к протоколу должна быть оформлена в виде комментариев к структурам данных, описывающим сообщения, в исходном коде.

Вариант: Дерево пиров Топология: ориентированное дерево. Информация, известная пиру при запуске: уникальное имя пира, его IP-адрес и порт, а также IP-адрес и порт его родителя в дереве (родитель обязан быть заранее запущен, для корня дерева родителем является он сам). Описание службы: каждый пир через стандартный поток ввода принимает команды – печать имени родителя, печать имён всех потомков, завершение работы (при этом один из потомков должен ставиться на место завершаемого пира в дереве).

2 Результаты

Листинг 1: Кошелёк на golang (peer.go)

```
1 package peer
2
3 import (
4   "encoding/json"
5   "fmt"
6   "log"
7   "net/http"
8   "sync"
9   "time"
```

```
10
11    "github.com/gorilla/websocket"
12  )
13
14  var upgrader = websocket.Upgrader{
15    CheckOrigin: func(r *http.Request) bool { return true },
16  }
17
18  type Peer struct {
19    PeerName    string
20    IPadress    string
21    IPParent    string
22    ParentConn  *websocket.Conn
23    ChildrenIPs []string
24  }
25
26  type RequestJSON struct {
27    From      string `json:"from"`
28    Msg       string `json:"msg"`
29    NewParent string `json:"newParent"`
30  }
31
32  type ResponseParentName struct {
33    ParentIP   string
34    ParentName string
35  }
36
37  type TotalNameJSONResp struct {
38    Receiver string
39    Names    []string
40  }
41
42  func CreatePeer(name, ipAdress, ParentIP string) *Peer {
43    return &Peer{
44      PeerName:    name,
45      IPadress:    ipAdress,
46      IPParent:    ParentIP,
47      ChildrenIPs: make([]string, 0),
48    }
49  }
50
51  func StartPeer(peer *Peer) {
52    http.HandleFunc("/", peer.handleClient)
53    go func() {
54      log.Println("Starting WebSocket server...")
55      addr := fmt.Sprintf(":%s", peer.IPadress)
```

```
56      if err := http.ListenAndServe(addr, nil); err != nil {
57        log.Fatalf("ListenAndServe error: %v", err)
58      }
59    }()
60    if peer.IPParent != "" {
61      log.Println("send hello")
62      go peer.sendHelloToParent()
63    }
64  }
65
66  func (peer *Peer) sendHelloToParent() {
67    ip := fmt.Sprintf("ws://localhost:%s/ws", peer.IPParent)
68    conn, _, err := websocket.DefaultDialer.Dial(ip, nil)
69    if err != nil {
70      return
71    }
72    jsonReq := &RequestJSON{peer.PeerName, "hello", peer.IPadress}
73    req, _ := json.Marshal(&jsonReq)
74    if err := conn.WriteMessage(websocket.TextMessage, req); err != nil {
75      log.Println("Write error:", err)
76      return
77    }
78    _, reply, _ := conn.ReadMessage()
79    if string(reply) == "ok" {
80      log.Println("Succesfully found parent")
81    }
82    peer.ParentConn = conn
83  }
84
85  func (peer *Peer) handleClient(w http.ResponseWriter, r *http.Request) {
86    conn, _ := upgrader.Upgrade(w, r, nil)
87    for {
88      _, msg, err := conn.ReadMessage()
89      if err != nil {
90        break
91      }
92      var jsonReq RequestJSON
93      err = json.Unmarshal(msg, &jsonReq)
94      if err != nil {
95        log.Println("Fail with json Unmarshal")
96      }
97      log.Printf("Get message from %s: %s\n", jsonReq.From, jsonReq.Msg)
98      switch jsonReq.Msg {
99      case "getName":
100       jsonResp := &ResponseParentName{peer.IPadress, peer.PeerName}
101       resp, _ := json.Marshal(jsonResp)
```

```
102         conn.WriteMessage(websocket.TextMessage, resp)
103       case "getTotalName":
104         jsonResp := &TotalNameJSONResp{peer.PeerName, make([]string, 0)}
105         jsonResp.Names = append(jsonResp.Names, peer.PeerName)
106         if peer.IPParent != "" {
107           coonection := peer.ParentConn
108           jsonReqForParent := &RequestJSON{peer.PeerName, "getTotalName", ""}
109           req, _ := json.Marshal(jsonReqForParent)
110           coonection.WriteMessage(websocket.TextMessage, req)
111           _, reply, _ := coonection.ReadMessage()
112           var jsonRespFromParent TotalNameJSONResp
113           json.Unmarshal(reply, &jsonRespFromParent)
114           jsonResp.Names = append(jsonResp.Names, jsonRespFromParent.Names...)
115         }
116         resp, _ := json.Marshal(jsonResp)
117         conn.WriteMessage(websocket.TextMessage, resp)
118       case "NewParent":
119         if peer.ParentConn != nil {
120           peer.ParentConn.Close()
121           peer.ParentConn = nil
122         }
123         peer.IPParent = jsonReq.NewParent
124         if peer.IPParent != "" {
125           go peer.sendHelloToParent()
126         }
127         conn.WriteMessage(websocket.TextMessage, []byte("ok"))
128         log.Printf("New parent %s\n", jsonReq.NewParent)
129       case "hello":
130         peer.ChildrenIPs = append(peer.ChildrenIPs, jsonReq.NewParent)
131         conn.WriteMessage(websocket.TextMessage, []byte("ok"))
132         log.Printf("Save children ip: %s", jsonReq.NewParent)
133       }
134     }
135 }
136
137 func (peer *Peer) GetNameFromParent() string {
138   conn := peer.ParentConn
139   req := &RequestJSON{peer.IPadress, "getName", ""}
140   msg, _ := json.Marshal(req)
141   if err := conn.WriteMessage(websocket.TextMessage, msg); err != nil {
142     log.Println("Write error:", err)
143   }
144   _, reply, _ := conn.ReadMessage()
145   var jsonResp ResponseParentName
146   err := json.Unmarshal(reply, &jsonResp)
147   if err != nil {
```

```
148     log.Println("Fail with json Unmarshal")
149   }
150   return jsonResp.ParentName
151 }
152
153 func (peer *Peer) GetTotalName() []string {
154   conn := peer.ParentConn
155   req := &RequestJSON{peer.PeerName, "getTotalName", ""}
156   msg, _ := json.Marshal(req)
157   if err := conn.WriteMessage(websocket.TextMessage, msg); err != nil {
158     log.Println("Write error:", err)
159   }
160   _, reply, _ := conn.ReadMessage()
161   var jsonResp TotalNameJSONResp
162   err := json.Unmarshal(reply, &jsonResp)
163   if err != nil {
164     log.Println("Fail with json Unmarshal")
165   }
166   return jsonResp.Names
167 }
168
169 func (peer *Peer) Exit() {
170   var wg sync.WaitGroup
171   for i, ip := range peer.ChildrenIPs {
172     wg.Add(1)
173     go func(i int, ip string) {
174       defer wg.Done()
175       ipAddr := fmt.Sprintf("ws://localhost:%s/ws", ip)
176       conn, _, err := websocket.DefaultDialer.Dial(ipAddr, nil)
177       if err != nil {
178         return
179       }
180       req := &RequestJSON{peer.PeerName, "NewParent", peer.IPParent}
181       msg, _ := json.Marshal(req)
182       if err := conn.WriteMessage(websocket.TextMessage, msg); err != nil {
183         log.Println("Write error:", err)
184         return
185       }
186       conn.SetReadDeadline(time.Now().Add(3 * time.Second))
187       _, reply, err := conn.ReadMessage()
188       if err != nil {
189         log.Println("Read error:", err)
190         return
191       }
192       if string(reply) == "ok" {
193         log.Printf("Succesfully changed parent for child #%d\n", i)
```

```
194         }
195       }(i, ip)
196     }
197     wg.Wait()
198     log.Println("Succesfully changed parent for all")
199 }
```

Листинг 2: Кошелёк на golang (main.go)

```
1  package main
2
3  import (
4    "bufio"
5    "fmt"
6    "os"
7    "peer-lab/peer"
8    "strconv"
9    "strings"
10 )
11
12 func main() {
13   scanner := bufio.NewReader(os.Stdin)
14   fmt.Println("Введите имя узла: ")
15   name, _ := scanner.ReadString('\n')
16   name = strings.TrimSpace(name)
17
18   fmt.Println("Введите свой порт: ")
19   ipAdress, _ := scanner.ReadString('\n')
20   ipAdress = strings.TrimSpace(ipAdress)
21
22   fmt.Println("Введите порт родителя: ")
23   ipParent, _ := scanner.ReadString('\n')
24   ipParent = strings.TrimSpace(ipParent)
25
26   MyPeer := peer.CreatePeer(name, ipAdress, ipParent)
27   go peer.StartPeer(MyPeer)
28
29   fmt.Println("1. Имя родителя\n2. Все потомки\n3. Завершение")
30   for {
31     text, err := scanner.ReadString('\n')
32     if err != nil {
33       fmt.Println(err)
34       return
35     }
36     commandId, _ := strconv.Atoi(strings.TrimSpace(text))
37     switch commandId {
```

```
38      case 1:
39          if MyPeer.IPParent == "" {
40              fmt.Println("Главный peer")
41          } else {
42              name := MyPeer.GetNameFromParent()
43              fmt.Printf("Имя родителя: %s\n", name)
44          }
45      case 2:
46          if MyPeer.IPParent == "" {
47              fmt.Println("Главный peer")
48          } else {
49              names := MyPeer.GetTotalName()
50              names = append([]string{MyPeer.PeerName}, names...)
51              for i, name := range names {
52                  fmt.Println(name)
53                  if i != len(names)-1 {
54                      fmt.Println("↑")
55
56                  }
57              }
58          }
59      case 3:
60          if MyPeer.IPParent == "" {
61              fmt.Println("Главный peer")
62          } else {
63              MyPeer.Exit()
64          }
65          return
66      }
67  }
68  }
```

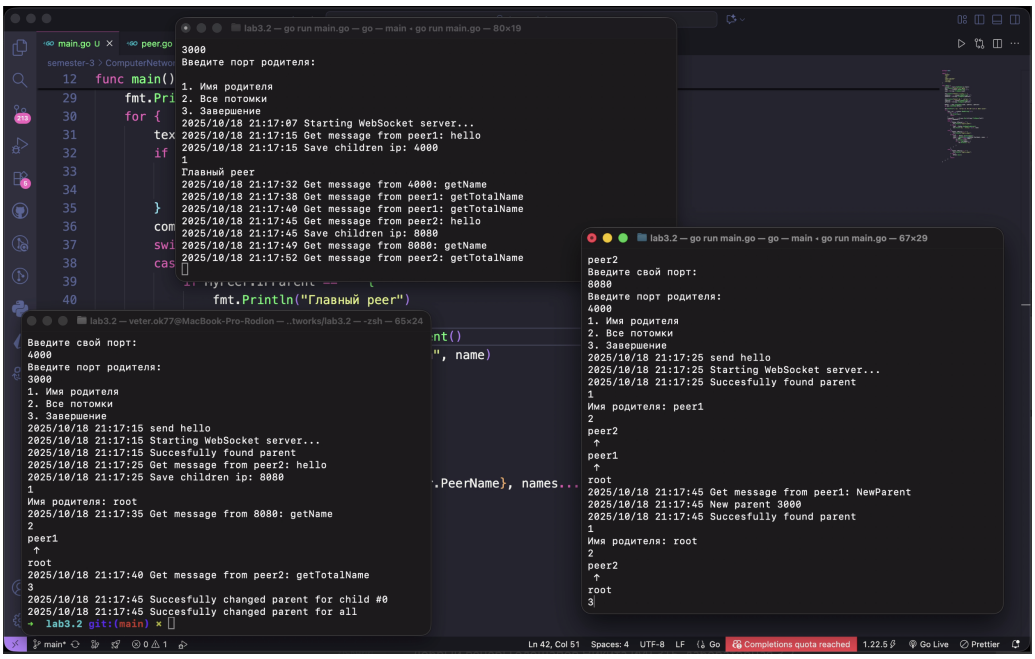

Рис. 1 — Результат работы

3 Вывод

Закрепляем навыки работы с websocket на golang после изнурительной лабораторной с кошельками