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 № 1

по курсу «Компьютерные системы и сети»

«Простейший протокол прикладного уровня»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

10 сентября 2024 г.

Цель работы

Целью работы является знакомство с принципами разработки протоколов прикладного уровня и их реализацией на языке Go.

Задача

Выполнение лабораторной работы состоит из двух частей.

◦ Разработать вариант протокола из таблиц 1–3. Протокол должен базироваться на

текстовых сообщениях в формате JSON. Результатом разработки протокола должен

быть набор типов языка Go, представляющих сообщения, и документация к ним в

виде комментариев в исходном тексте.

◦ Написать на языке Go клиент и сервер, взаимодействующие по разработанному

протоколу. Основные требования к клиенту и серверу:

◦ полная проверка данных, получаемых из сети (необходимо учитывать, что сообщения

могут приходить в неправильном формате и в неправильном порядке, а также могут

содержать неправильные данные);

◦ устойчивость к обрыву соединения;

◦ возможность одновременного подключения нескольких клиентов к одному серверу;

◦ сервер должен вести подробный лог всех ошибок, а также других важных событий

◦ (установка и завершение соединения с клиентом, приём и передача сообщений, и

т.п.).

Мой вариант: Протокол вычисления скалярного произведения двух n-мерных векторов.

Реализация

Я создал два файла: client.go, который реализовывал серверную часть, и server.go, который реализовывал клиентускую часть.

Код

Листинг 1: Файл client.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "flag"
6    "fmt"
7    "github.com/skorobogatov/input"
8    "net"
9    "strconv"
10 )
11
12 import "main/proto"
13 type vecs struct {
14   Vec1 []string `json: "vec1"`
15   Vec2 []string `json: "vec2"`
16 }
17
18 var vectors vecs
19 type oneint struct{
20   N string `json:n`
21 }
22
23 var result oneint
24 var globalN int
25 var vec1 []string
26 var vec2 []string
27
28 // interact - функция, содержащая цикл взаимодействия с сервером.
29 func interact(conn *net.TCPConn) {
30   defer conn.Close()
31   encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
32   for {
33         // Чтение команды из стандартного потока ввода
34     fmt.Printf("command = ")
35     command := input.Gets()
36
```

```
37          // Отправка запроса.
38      switch command {
39      case "quit":
40        send_request(encoder, "quit", nil)
41        return
42      case "setN":
43        fmt.Printf("Set n = ")
44        n := input.Gets()
45        globalN , _= strconv.Atoi(n)
46        var q oneint
47        q.N = n
48        send_request(encoder, "setN", q)
49      case "setVecs":
50        fmt.Printf("Vector 1 \n")
51        vec1 = make([]string, globalN)
52        for i := 0; i < globalN; i++ {
53          fmt.Printf("Set x%d/x%d = ", i, globalN-1)
54          vec1[i] = input.Gets()
55        }
56        fmt.Printf("Vector 2 \n")
57        vec2 = make([]string, globalN)
58        for i := 0; i < globalN; i++ {
59          fmt.Printf("Set x%d/x%d = ", i, globalN-1)
60          vec2[i] = input.Gets()
61        }
62        var v vecs
63        v.Vec1 = vec1
64        v.Vec2 = vec2
65        send_request(encoder, "setVectors", &v)
66      case "calc":
67        send_request(encoder, "calc", nil)
68
69          default:
70              fmt.Printf("error: unknown command\n")
71              continue
72      }
73
74      // Получение ответа.
75      var resp proto.Response
76      if err := decoder.Decode(&resp); err != nil {
77        fmt.Printf("error: %v\n", err)
78        break
79      }
80
81      // Вывод ответа в стандартный поток вывода.
82      switch resp.Status {
```

```
83      case "ok":
84        fmt.Printf("ok\n")
85      case "failed":
86        if resp.Data == nil {
87          fmt.Printf("error: data field is absent in response\n")
88        } else {
89          var errorMsg string
90          if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
91            fmt.Printf("error: malformed data field in response\n")
92          } else {
93            fmt.Printf("failed: %s\n", errorMsg)
94          }
95        }
96      case "result":
97        if resp.Data == nil {
98          fmt.Printf("error: data field is absent in response\n")
99        } else {
100         var out oneint
101         if err := json.Unmarshal(*resp.Data, &out); err != nil {
102           fmt.Printf("error: malformed data field in response\n")
103         } else {
104           fmt.Printf("result: %s\n", out.N)
105         }
106       }
107     default:
108       fmt.Printf("error: server reports unknown status %q\n", resp.
      Status)
109     }
110   }
111 }
112
113 // send_request - вспомогательная функция для передачи запроса с указанн
      ой командой
114 // и данными. Данные могут быть пустыми (data == nil).
115 func send_request(encoder *json.Encoder, command string, data interface
      {}) {
116   var raw json.RawMessage
117   raw, _ = json.Marshal(data)
118   encoder.Encode(&proto.Request{command, &raw})
119 }
120
121 func main() {
122   // Работа с командной строкой, в которой может указываться необязатель
      ный ключ -addr.
123   var addrStr string
```

```
124   flag.StringVar(&addrStr, "addr", "185.102.139.169:1572", "specify ip
      address and port")
125   flag.Parse()
126
127   // Разбор адреса, установка соединения с сервером и
128   // запуск цикла взаимодействия с сервером.
129   if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
130     fmt.Printf("error: %v\n", err)
131   } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
132     fmt.Printf("error: %v\n", err)
133   } else {
134     interact(conn)
135   }
136 }
```

Листинг 2: Файл server.go

```
 1 package main
 2
 3 import (
 4   "encoding/json"
 5   "flag"
 6   "fmt"
 7   "github.com/mgutz/logxi/v1"
 8   "math/big"
 9   "net"
10   "strconv"
11
12 )
13
14 import "main/proto"
15
16 // Client - состояние клиента.
17 type Client struct {
18   logger log.Logger     // Объект для печати логов
19   conn   *net.TCPConn   // Объект TCP-соединения
20   enc    *json.Encoder // Объект для кодирования и отправки сообщений
21   sum    *big.Rat      // Текущая сумма полученных от клиента дробей
22   count  int64         // Количество полученных от клиента дробей
23 }
24
25 // NewClient - конструктор клиента, принимает в качестве параметра
26 // объект TCP-соединения.
27 func NewClient(conn *net.TCPConn) *Client {
28   return &Client{
29     logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())
       ),
```

```
30      conn:    conn,
31      enc:     json.NewEncoder(conn),
32      sum:     big.NewRat(0, 1),
33      count:   0,
34    }
35  }
36
37  // serve - метод, в котором реализован цикл взаимодействия с клиентом.
38  // Подразумевается, что метод serve будет вызываться в отдельной go-прог
        рамме.
39  func (client *Client) serve() {
40    defer client.conn.Close()
41    decoder := json.NewDecoder(client.conn)
42    for {
43      var req proto.Request
44      if err := decoder.Decode(&req); err != nil {
45        client.logger.Error("cannot decode message", "reason", err)
46        break
47      } else {
48        client.logger.Info("received command", "command", req.Command)
49        if client.handleRequest(&req) {
50          client.logger.Info("shutting down connection")
51          break
52        }
53      }
54    }
55  }
56
57  var globalN int
58
59  func calcScalar(vec1 []string, vec2 []string) string {
60    sum := 0
61    for i := 0; i < globalN; i++ {
62      v1, _ := strconv.Atoi(vec1[i])
63      v2, _ := strconv.Atoi(vec2[i])
64      sum += v1*v2
65    }
66    return strconv.Itoa(sum)
67  }
68
69  type vecs struct {
70    Vec1 []string `json: "vec1"`
71    Vec2 []string `json: "vec2"`
72  }
73
74  var vectors vecs
```

```
75 type oneint struct{
76   N string `json:n`
77 }
78
79
80 // handleRequest - метод обработки запроса от клиента. Он возвращает
       true,
81 // если клиент передал команду "quit" и хочет завершить общение.
82 func (client *Client) handleRequest(req *proto.Request) bool {
83   switch req.Command {
84   case "quit":
85     client.respond("ok", nil)
86     return true
87   case "setN":
88     errorMsg := ""
89     if req.Data == nil {
90       errorMsg = "data field is absent"
91     } else {
92       var s oneint
93             if err := json.Unmarshal(*req.Data, &s); err != nil {
94                 errorMsg = "malformed data field"
95             } else {
96                 globalN, _ = strconv.Atoi(s.N)
97                 fmt.Printf("N is set to %d\n", globalN)
98             }
99         }
100
101    if errorMsg == "" {
102      client.respond("ok", nil)
103    } else {
104      client.logger.Error("addition failed", "reason", errorMsg)
105      client.respond("failed", errorMsg)
106    }
107  case "setVectors":
108    errorMsg := ""
109    if req.Data == nil {
110      errorMsg = "data field is absent"
111    } else {
112      if err := json.Unmarshal(*req.Data, &vectors); err != nil {
113        errorMsg = "malformed data field"
114      } else {
115        // fmt.Printf("N is set to %d\n", globalN)
116        fmt.Println("Vec1 is set to", vectors.Vec1)
117        fmt.Println("Vec2 is set to", vectors.Vec2)
118      }
119    }
```

```
120     if errorMsg == "" {
121       client.respond("ok", nil)
122     } else {
123       client.logger.Error("addition failed", "reason", errorMsg)
124       client.respond("failed", errorMsg)
125     }
126   case "calc":
127     var result oneint
128     result.N = calcScalar(vectors.Vec1, vectors.Vec2)
129         fmt.Printf("Result: %s\n", result.N)
130     client.respond("result", &result)
131   default:
132     client.logger.Error("unknown command")
133     client.respond("failed", "unknown command")
134   }
135   return false
136 }
137
138 // respond - вспомогательный метод для передачи ответа с указанным стату
      сом
139 // и данными. Данные могут быть пустыми (data == nil).
140 func (client *Client) respond(status string, data interface{}) {
141   var raw json.RawMessage
142   raw, _ = json.Marshal(data)
143   client.enc.Encode(&proto.Response{status, &raw})
144 }
145
146 func main() {
147     // Работа с командной строкой, в которой может указываться необязате
      льный ключ -addr.
148   var addrStr string
149   flag.StringVar(&addrStr, "addr", "185.102.139.169:1572", "specify ip
       address and port")
150   flag.Parse()
151
152     // Разбор адреса, строковое представление которого находится в перем
      енной addrStr.
153   if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
154     log.Error("address resolution failed", "address", addrStr)
155   } else {
156     log.Info("resolved TCP address", "address", addr.String())
157
158         // Инициация слушания сети на заданном адресе.
159     if listener, err := net.ListenTCP("tcp", addr); err != nil {
160       log.Error("listening failed", "reason", err)
161     } else {
```

```
162             // Цикл приёма входящих соединений.
163         for {
164           if conn, err := listener.AcceptTCP(); err != nil {
165             log.Error("cannot accept connection", "reason", err)
166           } else {
167             log.Info("accepted connection", "address", conn.RemoteAddr().
      String())
168
169                       // Запуск go-программы для обслуживания клиентов.
170             go NewClient(conn).serve()
171           }
172         }
173       }
174     }
175 }
```

Вывод программы

После загрузки программы на серверной части 185.102.139.169 и ее запуска, я подключился при помощи клиента. По мере ввода команд со стороны клиента на сервер также прилетали данные, они обрабатывалась и отправлялись назад в соответствии с заданием.

Листинг 3: Ввод и вывод со стороны клиента

```
1  root@net3:~/Fedukov/lab1/sample/src/client# go run client.go
2  command = setN
3  Set n = 3
4  ok
5  command = setVecs
6  Vector 1
7  Set x0/x2 = 1
8  Set x1/x2 = 0
9  Set x2/x2 = 1
10 Vector 2
11 Set x0/x2 = 0
12 Set x1/x2 = 0
13 Set x2/x2 = 1
14 ok
15 command = calc
16 result: 1
17 command =
```

Листинг 4: Вывод со стороны сервера

```
root@net4:~/Fedukov/lab1/sample/src/server# go run server.go
N is set to 3
Vec1 is set to [1 0 1]
Vec2 is set to [0 0 1]
Result: 1
```

Вывод

Я научился применять протокол websocket и на его основе реализовал свой собственный. Кроме того, я потренировался устанавливать Go и поработал с удаленными машинами по ssh.