

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Кафедра компьютерных систем и программных технологий

# Технологии компьютерных сетей

Отчет по лабораторной работе  
Изучение протоколов TCP и UDP

**Работу**  
**выполнил:**  
Кузьмин Н.А.  
Группа: 43501/1  
**Преподаватель:**  
Алексюк А.О.

Санкт-Петербург  
2018

# Содержание

<b>1. Цель работы</b>	<b>3</b>
<b>2. Теоретическая информация</b>	<b>3</b>
2.1. Создание сокета	3
2.2. Организация соединения	3
2.3. Передача и приём данных по протоколу TCP	4
2.4. Передача и приём данных по протоколу UDP	4
2.5. Завершение TCP-соединения	4
2.6. Закрытие сокета	4
2.7. Привязывание сокета	5
2.8. Перевод TCP-сокета в состояние прослушивания	5
2.9. Приём входящего TCP-соединения	5
<b>3. Ход работы</b>	<b>5</b>
3.1. Клиент-серверное приложение на Linux с использованием TCP	5
3.2. Клиент-серверное приложение на Windows с использованием TCP	11
3.3. Клиент-серверное приложение на Linux с использованием UDP	16
3.4. Индивидуальное задание: платежная система	21
3.4.1. Задание	21
3.4.2. Описание реализации	22
3.4.3. Протокол запроса клиент-сервер	22
3.4.4. Протокол ответа сервер-клиент	23
3.4.5. Организация хранения данных	23
3.4.6. Организация одновременной работы нескольких клиентов	24
3.4.7. Код проекта	24
3.5. Socks5 прокси сервер	24
3.5.1. Описание протокола	24
3.5.2. Исходный код	26
<b>4. Выводы</b>	<b>31</b>

# 1. Цель работы

Изучение протоколов UDP и TCP и их использование на практике для программирования сокетов.

## 2. Теоретическая информация

С точки зрения архитектуры TCP/IP сокетом называется пара (IP-адрес, порт), однозначно идентифицирующая прикладное приложение в сети Internet.

С точки зрения операционной системы BSD-сокет (Berkley Software Distribution) или просто сокет — это выделенные операционной системой набор ресурсов, для организации сетевого взаимодействия. К таким ресурсам относятся, например, буфера для приёма/посылки данных или очереди сообщений. Технология сокетов была разработана в университете Беркли и впервые появилась в операционной системе BSD-UNIX.

В операционной системе MS Windows имеется аналогичная библиотека сетевого взаимодействия WinSock, реализованная на основе библиотеки BSD-сокетов. Существует две версии библиотеки WinSock1 и WinSock2. В подавляющем большинстве случаев функции и типы библиотеки WinSock совпадают с функциями и типами BSD-сокетов. Об имеющихся отличиях будет сказано отдельно.

### 2.1. Создание сокета

Для создания сокета в библиотеках BSD-socket и WinSock имеется системный вызов socket:

```
int socket(int domain, int type, int protocol);
```

В случае успеха результат вызова функции — дескриптор созданного сокета, в случае ошибки (-1) в библиотеке BSD-socket и INVALID\_SOCKET в библиотеке WinSock.

Параметр type определяет тип создаваемого сокета. Этот параметр может принимать значения:

- SOCK\_STREAM — для организации надёжного канала связи с установлением соединений
- SOCK\_DGRAM — для организации ненадёжного дейтаграммного канала связи
- SOCK\_RAW — для организации низкоуровневого доступа на основе «сырых» сокетов

Параметр protocol — идентификатор используемого протокола. В большинстве случаев протокол однозначно определяется типом создаваемого сокета, и передаваемое значение этого параметра — 0.

### 2.2. Организация соединения

Для установления TCP-соединения используется вызов connect:

```
int connect(int s, const struct sockaddr* serv_addr, int addr_len);
```

Результатом выполнения функции является установление TCP-соединения с TCP-сервером. Функция возвращает значение 0 в случае успеха и -1 в случае ошибки.

Параметр s — дескриптор созданного сокета.

Параметр serv\_addr — указатель на структуру, содержащую параметры удалённого узла.

Параметр addr\_len — размер в байтах структуры, на которую указывает параметр serv\_addr.

## 2.3. Передача и приём данных по протоколу TCP

Передача и приём данных в рамках установленного TCP-соединения осуществляется вызовами `send` и `recv`:

```
int send(int s, const void *msg, size_t len, int flags); int recv(int s, void *msg, size_t len, int flags);
```

Параметр `s` – дескриптор сокета, параметр `msg` – указатель на буфер, содержащий данные (вызов `send`), или указатель на буфер, предназначенный для приёма данных (вызов `recv`). Параметр `len` – длина буфера в байтах, параметр `flags` – опции отправки или приёма данных.

Возвращаемое значение – число успешно посланных или принятых байтов, в случае ошибки функция возвращает значение `-1`.

## 2.4. Передача и приём данных по протоколу UDP

В случае установленного адреса по умолчанию для протокола UDP (вызов `connect`) функции для передачи и приёма данных по протоколу UDP можно использовать вызовы `send` и `recv`.

Если адрес и порт по умолчанию для протокола UDP не установлен, то параметры удалённой стороны необходимо указывать или получать при каждом вызове операций записи или чтения. Для протокола UDP имеется два аналогичных вызова `sendto` и `recvfrom`:

```
int sendto(int s, const void *buf, size_t len, int flags, struct sockaddr *to, int* tolen);
int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, int* fromlen);
```

Параметры `s`, `buf`, `len` и `flags` имеют тот же смысл, что и в случае использования функций `send` и `recv`, параметры `to` и `tolen` – атрибуты адреса удалённого сокета при отсылке данных, параметры `from` и `fromlen` – атрибуты структуры данных в которую помещаются параметры удалённого сокета при получении данных.

## 2.5. Завершение TCP-соединения

Завершение установленного TCP-соединения осуществляется в библиотеке BSD-socket с помощью вызова `shutdown`:

```
int shutdown(int s, int how);
```

Параметр `s` – дескриптор сокета, параметр `how` – определяет способ закрытия:

- `SHUT_RD` – запрещён приём данных
- `SHUT_WR` – запрещена передача данных
- `SHUT_RDWR` – запрещены и приём и передача данных

## 2.6. Закрытие сокета

По окончании работы следует закрыть сокет, для этого в библиотеке BSD-socket предусмотрен вызов `close`:

```
int close(int s);
```

Аналогичный вызов в библиотеке WinSock имеет название `closesocket`:

```
int closesocket(SOCKET s);
```

Параметр `s` – дескриптор сокета. Возвращаемое значение – `0`, в случае успеха.

## 2.7. Привязывание сокета

Созданный сокет является объектом операционной системы, использующим её отдельные ресурсы. В то же время в большинстве случаев недостаточно просто выделить ресурсы операционной системы, а следует также связать эти ресурсы с конкретными сетевыми параметрами: сетевым адресом и номером порта. Особенно это важно для серверных сокетов, для которых такая связь – необходимое требование доступности разрабатываемого сетевого сервиса.

Организация привязки созданного вызовом `socket()` сокета к определённым IP-адресам и портам осуществляется с помощью функция `bind`:

```
int bind(int s, struct sockaddr *addr, socklen_t addrlen);
```

Параметр `s` – дескриптор сокета, параметр `addr` задаёт указатель на структуру, хранящую параметры адреса и порта, `addrlen` – размер структуры `addr` в байтах

## 2.8. Перевод TCP-сокета в состояние прослушивания

Для перевода сокета в состояние прослушивания служит системный вызов `listen`:

```
int listen(int s, int backlog);
```

Параметр `s` – дескриптор сокета, параметр `backlog` – задаёт максимальную длину, до которой может расти очередь ожидающих соединений. В случае успеха возвращаемое значение – 0. При ошибке возвращается -1.

## 2.9. Приём входящего TCP-соединения

В случае, когда сокет находится в состоянии прослушивания (`listen`) необходимо отслеживать поступление входящих соединений. Для этого предусмотрен системный вызов `accept`:

```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

Параметр `s` – дескриптор прослушивающего сокета, параметр `addr` – указатель на структуру, содержащую параметры сокета, инициирующего соединение, `addrlen` – размер структуры `addr` в байтах.

Возвращаемое значение – дескриптор сокета, созданного для нового соединения. Большинство параметров нового сокета соответствуют параметрам слушающего сокета. Полученный сокет в дальнейшем может использоваться для передачи и приёма данных.

В случае если входящих соединений нет, то функция `accept` ожидает поступления запроса на входящее соединение.

# 3. Ход работы

## 3.1. Клиент-серверное приложение на Linux с использованием TCP

Листинг 1: CMakeLists для сервера

```
1 cmake_minimum_required(VERSION 2.8)
2 project(server_linux)
3
4 find_package(Threads)
5
6 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
7
8 set(SOURCE_FILES main.c)
9 add_executable(server_linux ${SOURCE_FILES})
```

```
10
11 target_link_libraries (server_linux ${CMAKE_THREAD_LIBS_INIT})
```

## Листинг 2: Код сервера

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <netdb.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
7
8 #include <string.h>
9
10 #include <unistd.h>
11
12 /* function for messaging thread */
13 void *communicate_func (void *arg);
14
15 /* function for listen thread */
16 void *listen_func (void *arg);
17
18 /* variable for enable/disable logging */
19 int log_enabled = 0;
20
21 /* function for logging */
22 void logv(char* msg, int value);
23
24 void log(char* msg);
25
26 struct socket_data {
27     int sockfd;
28     unsigned int clilen;
29     struct sockaddr_in cli_addr;
30 };
31
32 int main(int argc, char *argv[]) {
33     int sockfd;
34     uint16_t portno;
35     unsigned int clilen;
36     struct sockaddr_in serv_addr, cli_addr;
37     struct socket_data data;
38
39     /* First call to socket() function */
40     sockfd = socket(AF_INET, SOCK_STREAM, 0);
41
42     if (sockfd < 0) {
43         perror("ERROR opening socket");
44         exit(1);
45     }
46
47     /* Initialize socket structure */
48     bzero((char *) &serv_addr, sizeof(serv_addr));
49     portno = 5001;
50
51     serv_addr.sin_family = AF_INET;
52     serv_addr.sin_addr.s_addr = INADDR_ANY;
53     serv_addr.sin_port = htons(portno);
54
55     if(setsockopt(sockfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), &(int)
    ↪ {1}, sizeof(int)) < 0) {
```

```

56     perror("ERROR_on_setsockopt");
57     shutdown(sockfd, 2);
58     close(sockfd);
59 }
60
61 /* Now bind the host address using bind() call.*/
62 if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
63     perror("ERROR_on_binding");
64     shutdown(sockfd, 2);
65     close(sockfd);
66     exit(1);
67 }
68
69 /* Now start listening for the clients, here process will
70 * go in sleep mode and will wait for the incoming connection */
71
72 listen(sockfd, 5);
73 clilen = sizeof(cli_addr);
74
75 /* data which will be sent in listen thread */
76 data.sockfd = sockfd;
77 data.clilen = clilen;
78 data.cli_addr = cli_addr;
79
80 pthread_t listen_thread; //thread for listening
81 int result; //result of creating thread
82 result = pthread_create(&listen_thread, NULL, listen_func, &data); //create
83 listen thread
84 if(result != 0) {
85     perror("Error_while_creating_listen_thread");
86     shutdown(sockfd, 2);
87     close(sockfd);
88     exit(1);
89 }
90
91 /* if pressed q then exit program */
92 int key = 0;
93 while(1) {
94     key = getchar();
95     if(key == 'q') break;
96     // enable/disable logging
97     if(key == 'l') {
98         log_enabled = !log_enabled;
99         log("log_enabled");
100     }
101     shutdown(sockfd, 2);
102     close(sockfd);
103     sleep(1);
104     exit(0);
105 }
106
107 /* waiting for connection */
108 void *listen_func (void *arg) {
109     struct socket_data data = * (struct socket_data *) arg;
110     int newsockfd;
111
112     /* Accept connections while server is on */
113     while(1) {
114         /* Accept actual connection from the client */
115         newsockfd = accept(data.sockfd, (struct sockaddr *) &data.cli_addr, &data.

```

```

115     ↪ clilen);
116
117     if (newsockfd < 0) {
118         perror("ERROR_on_accept");
119         pthread_exit(1);
120     }
121
122     logv("new_connection_accepted_with_fd=", newsockfd);
123
124     /* Making new thread for messaging with client */
125     pthread_t thread; //thread for messaging
126     int result; //result of thread creating
127     result = pthread_create(&thread, NULL, communicate_func, &newsockfd); //create
128     new thread
129     if(result != 0) {
130         perror("Error_while_creating_thread");
131         shutdown(newsockfd, 2);
132         close(newsockfd);
133     }
134     }
135     pthread_exit(0);
136 }
137
138 /* function for thread */
139 void *communicate_func (void *arg) {
140     int newsockfd = * (int *) arg;
141     char buffer[256];
142     char output[300]; //put message in buffer
143     ssize_t n;
144     int message_length;
145     int received_length = 0;
146     int read_length = 0;
147
148     bzero(output, 300);
149
150     strcat(output, "Here_is_the_message:_");
151
152     /* If connection is established then start communicating
153        First we need to read length of message */
154     n = read(newsockfd, (char*)&message_length, sizeof(int)); //recv on Windows
155
156     logv("length_= ", message_length);
157
158     if (n < 0) {
159         perror("ERROR_reading_length_of_the_message_from_socket");
160         shutdown(newsockfd, 2);
161         close(newsockfd);
162         pthread_exit(1);
163     }
164
165     while (received_length < message_length) {
166         bzero(buffer, 256);
167         read_length = message_length - received_length;
168         if(read_length > 255) read_length = 255;
169         n = read(newsockfd, buffer, read_length);
170         if (n < 0) {
171             perror("ERROR_reading_from_socket");
172             shutdown(newsockfd, 2);
173             close(newsockfd);
174             pthread_exit(1);

```



```

175 }
176 //n == 0 means that we read all bytes or client has closed connection
177 if(n == 0) {
178     if(received_length < message_length) {
179         printf("Error: _Not_all_bytes_have_been_read\n");
180     } else {
181         printf("All_bytes_have_been_read\n");
182     }
183     break;
184 }
185 received_length += n;
186 strcat(output, buffer);
187
188 logv("received_bytes:_", n);
189 }
190 printf("%s\n", output);
191
192 /* Write a response to the client */
193 n = write(newsockfd, "I_got_your_message", 18); // send on Windows
194
195 if (n < 0) {
196     perror("ERROR_writing_to_socket");
197     shutdown(newsockfd, 2);
198     close(newsockfd);
199 pthread_exit(1);
200 }
201
202 logv("closing_connection_with_fd=_", newsockfd);
203
204 shutdown(newsockfd, 2);
205 close(newsockfd); //close socket after messaging
206 pthread_exit(0);
207 }
208
209 void logv(char* msg, int value) {
210     if(log_enabled) {
211         printf("log:_%s_%d\n", msg, value);
212     }
213 }
214
215 void log(char* msg) {
216     if(log_enabled) {
217         printf("log:_%s\n", msg);
218     }
219 }

```

Листинг 3: CMakeLists для клиента

```

1 cmake_minimum_required(VERSION 2.8)
2 project(client_linux)
3
4 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
5
6 set(SOURCE_FILES main.c)
7 add_executable(client_linux ${SOURCE_FILES})

```

Листинг 4: Код клиента

```

1 #include <stdio.h>
2 #include <stdlib.h>
3

```

```

4 #include <netdb.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
7
8 #include <string.h>
9
10 int main(int argc, char *argv[]) {
11     int sockfd, n;
12     uint16_t portno;
13     struct sockaddr_in serv_addr;
14     struct hostent *server;
15
16     char buffer[256];
17
18     if (argc < 3) {
19         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
20         exit(0);
21     }
22
23     portno = (uint16_t) atoi(argv[2]);
24
25     /* Create a socket point */
26     sockfd = socket(AF_INET, SOCK_STREAM, 0);
27
28     if (sockfd < 0) {
29         perror("ERROR_opening_socket");
30         exit(1);
31     }
32
33     server = gethostbyname(argv[1]);
34
35     if (server == NULL) {
36         fprintf(stderr, "ERROR,_no_such_host\n");
37         exit(0);
38     }
39
40     bzero((char *) &serv_addr, sizeof(serv_addr));
41     serv_addr.sin_family = AF_INET;
42     bcopy(server->h_addr, (char *) &serv_addr.sin_addr.s_addr, (size_t) server->
    ↪ h_length);
43     serv_addr.sin_port = htons(portno);
44
45     /* Now connect to the server */
46     if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    ↪ {
47         perror("ERROR_connecting");
48     shutdown(sockfd, 2);
49     close(sockfd);
50         exit(1);
51     }
52
53     /* Now ask for a message from the user, this message
54      * will be read by server
55      */
56
57     printf("Please_enter_the_message:_");
58     bzero(buffer, 256);
59     fgets(buffer, 255, stdin);
60
61     int message_length = strlen(buffer);

```

```

62
63     /* Send message length to the server */
64     n = write(sockfd, (char*)&message_length, sizeof(int));
65
66     if (n < 0) {
67         perror("ERROR_writing_length_to_socket");
68         shutdown(sockfd, 2);
69         close(sockfd);
70         exit(1);
71     }
72
73     /* Send message to the server */
74     n = write(sockfd, buffer, strlen(buffer));
75
76     if (n < 0) {
77         perror("ERROR_writing_to_socket");
78         shutdown(sockfd, 2);
79         close(sockfd);
80         exit(1);
81     }
82
83     /* Now read server response */
84     bzero(buffer, 256);
85     n = read(sockfd, buffer, 255);
86
87     if (n < 0) {
88         perror("ERROR_reading_from_socket");
89         shutdown(sockfd, 2);
90         close(sockfd);
91         exit(1);
92     }
93
94     printf("%s\n", buffer);
95     return 0;
96 }

```

### 3.2. Клиент-серверное приложение на Windows с использованием TCP

Листинг 5: Код сервера

```

1  #define WIN32_LEAN_AND_MEAN
2
3  #include <windows.h>
4  #include <winsock2.h>
5  #include <ws2tcpip.h>
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include <conio.h>
9
10 #pragma comment(lib, "Ws2_32.lib")
11 #pragma warning(disable:4996)
12
13 DWORD WINAPI listen_func(LPVOID p);
14
15 DWORD WINAPI communicate_func(LPVOID p);
16
17 int main(int argc, char *argv[]) {

```

```

18 SOCKET s;
19 WSADATA wsa;
20 unsigned int portno;
21 struct sockaddr_in serv_addr;
22
23
24 printf("\nInitialising Winsock...");
25 if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
26 {
27     printf("Failed. Error Code: %d", WSAGetLastError());
28     return 1;
29 }
30 printf("Initialised.\n");
31
32 /* First call to socket() function */
33 s = socket(AF_INET, SOCK_STREAM, 0);
34
35 if (s == INVALID_SOCKET) {
36     perror("ERROR opening socket");
37     exit(1);
38 }
39
40 /* Initialize socket structure */
41 memset((char *)&serv_addr, 0, sizeof(serv_addr));
42 portno = 5001;
43
44 serv_addr.sin_family = AF_INET;
45 serv_addr.sin_addr.s_addr = INADDR_ANY;
46 serv_addr.sin_port = htons(portno);
47
48 /* Now bind the host address using bind() call.*/
49 if (bind(s, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
50     perror("ERROR on binding");
51     closesocket(s);
52     exit(1);
53 }
54
55 /* Now start listening for the clients, here process will
56 * go in sleep mode and will wait for the incoming connection */
57 listen(s, 5);
58
59 //thread for listening
60 HANDLE listen_thread = CreateThread(NULL, 0, listen_func, &s, 0, NULL);
61
62 if (listen_thread == NULL) {
63     perror("Error while creating listen_thread");
64     closesocket(s);
65     exit(1);
66 }
67
68 /* if pressed q then exit program */
69 int key = 0;
70 while (1) {
71     key = getchar();
72     if (key == 'q') break;
73 }
74 closesocket(s);
75 _getch();
76 exit(0);
77 }

```

```

78
79 DWORD WINAPI listen_func(LPVOID p) {
80     SOCKET *s = ((SOCKET *)p)[0];
81     int c;
82     SOCKET new_socket;
83     struct sockaddr_in cli_addr;
84
85     c = sizeof(struct sockaddr_in);
86     while (1) {
87
88         new_socket = accept(s, (struct sockaddr *)&cli_addr, &c);
89         if (new_socket == INVALID_SOCKET)
90         {
91             perror("ERROR_on_accept");
92             printf("code: %d\n", WSAGetLastError());
93             ExitThread(1);
94         }
95
96         //print connection
97         HOSTENT *hst;
98         hst = gethostbyaddr((char *)&cli_addr.sin_addr.s_addr, 4, AF_INET);
99         printf("+%s[%s]_new_connect!\n", (hst) ? hst->h_name : "", inet_ntoa(
100         ↪ cli_addr.sin_addr));
101
102         /* Making new thread for messaging with client */
103         HANDLE communicate_thread = CreateThread(NULL, 0, communicate_func, &
104         ↪ new_socket, 0, NULL);
105
106         if (communicate_thread == NULL) {
107             perror("Error_while_creating_thread_for_client");
108             closesocket(new_socket);
109         }
110     }
111     printf("exit_from_listen_func\n");
112     ExitThread(0);
113 }
114
115 DWORD WINAPI communicate_func(LPVOID p) {
116     SOCKET *s = ((SOCKET *)p)[0];
117
118     char buffer[256];
119     char output[300]; //put message in buffer
120     int n;
121     int message_length = 0;
122     int received_length = 0;
123     int read_length = 0;
124
125     memset(output, 0, 300);
126
127     strcat(output, "Here_is_the_message:_");
128
129     /* If connection is established then start communicating
130     First we need to read length of message */
131     n = recv(s, (char *)&message_length, sizeof(int), 0);
132
133     if (n < 0) {
134         perror("ERROR_reading_length_of_the_message_from_socket");
135         closesocket(s);
136         ExitThread(1);
137     }

```

```

136
137 while (received_length <= message_length) {
138     memset(buffer, 0, 256);
139     read_length = message_length - received_length;
140     if (read_length > 255) read_length = 255;
141     n = recv(s, buffer, read_length, 0);
142     if (n < 0) {
143         printf("ERROR_reading_from_socket");
144         closesocket(s);
145         ExitThread(1);
146     }
147     received_length += n;
148     strcat(output, buffer);
149 }
150 printf("%s\n", output);
151
152 n = send(s, "I_got_your_message", 18, 0);
153
154 if (n < 0) {
155     perror("ERROR_writing_to_socket");
156     closesocket(s);
157     ExitThread(1);
158 }
159
160 closesocket(s); //close socket after messaging
161 ExitThread(0);
162 }

```

Листинг 6: Код клиента

```

1 #define WIN32_LEAN_AND_MEAN
2 #define _WINSOCK_DEPRECATED_NO_WARNINGS
3
4 #include <windows.h>
5 #include <winsock2.h>
6 #include <ws2tcpip.h>
7 #include <stdlib.h>
8 #include <stdio.h>
9 #include <conio.h>
10
11 #pragma comment (lib, "Ws2_32.lib")
12
13 int main(int argc, char *argv[]) {
14     int n;
15     SOCKET sockfd;
16     unsigned int portno;
17     struct sockaddr_in serv_addr;
18     struct hostent *server;
19     WSADATA wsa;
20
21     char buffer[256];
22
23     if (argc < 3) {
24         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
25         exit(0);
26     }
27
28     portno = (unsigned int)atoi(argv[2]);
29
30     printf("\nInitialising_Winsock...");
31     if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)

```

```

32 {
33     printf("Failed. _Error_Code_: _%d", WSAGetLastError());
34     exit(1);
35 }
36 printf("Initialised.\n");
37
38 /* Create a socket point */
39 sockfd = socket(AF_INET, SOCK_STREAM, 0);
40
41 if (sockfd == NULL) {
42     perror("ERROR_opening_socket");
43     exit(1);
44 }
45
46 server = gethostbyname(argv[1]);
47
48 if (server == NULL) {
49     fprintf(stderr, "ERROR, _no_such_host\n");
50     exit(0);
51 }
52
53 memset((char *)&serv_addr, 0, sizeof(serv_addr));
54 serv_addr.sin_family = AF_INET;
55 memmove((char *)&serv_addr.sin_addr.s_addr, server->h_addr, (size_t)server->
    ↪ h_length);
56 serv_addr.sin_port = htons(portno);
57
58 /* Now connect to the server */
59 if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) ==
    ↪ SOCKET_ERROR) {
60     perror("ERROR_connecting");
61     closesocket(sockfd);
62     exit(1);
63 }
64
65 /* Now ask for a message from the user, this message
66 * will be read by server
67 */
68 printf("Please_enter_the_message:_");
69 memset(buffer, 0, 256);
70 fgets(buffer, 255, stdin);
71
72 int message_length = strlen(buffer);
73
74 /* Send message length to the server */
75 n = send(sockfd, (char *)&message_length, sizeof(int), 0);
76
77 if (n < 0) {
78     perror("ERROR_writing_length_to_socket");
79     exit(1);
80 }
81
82 /* Send message to the server */
83 n = send(sockfd, buffer, strlen(buffer), 0);
84
85 if (n < 0) {
86     perror("ERROR_writing_to_socket");
87     exit(1);
88 }
89

```

```

90  /* Now read server response */
91  memset(buffer, 0, 256);
92  n = recv(sockfd, buffer, 255, 0);
93
94  if (n < 0) {
95      perror("ERROR reading from socket");
96      exit(1);
97  }
98
99  printf("%s\n", buffer);
100  exit(0);
101 }

```

### 3.3. Клиент-серверное приложение на Linux с использованием UDP

Листинг 7: CMakeLists для сервера

```

1  cmake_minimum_required(VERSION 2.8)
2  project(server_linux)
3
4  find_package (Threads)
5
6  set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
7
8  set(SOURCE_FILES main.c)
9  add_executable(server_linux ${SOURCE_FILES})
10
11 target_link_libraries (server_linux ${CMAKE_THREAD_LIBS_INIT})

```

Листинг 8: Код сервера

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <netdb.h>
5  #include <netinet/in.h>
6  #include <unistd.h>
7
8  #include <string.h>
9
10 #include <unistd.h>
11
12 /* function for listen thread */
13 void *listen_func (void *arg);
14
15 /* variable for enable/disable logging */
16 int log_enabled = 0;
17
18 /* function for logging */
19 void logv(char* msg, int value);
20
21 void log(char* msg);
22
23 struct socket_data {
24     int sockfd;
25     unsigned int cli_len;
26     struct sockaddr_in cli_addr;
27 };
28

```



```

29 int main(int argc, char *argv[]) {
30     int sockfd;
31     uint16_t portno;
32     unsigned int clilen;
33     struct sockaddr_in serv_addr, cli_addr;
34     struct socket_data data;
35
36     /* First call to socket() function */
37     sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
38
39     if (sockfd < 0) {
40         perror("ERROR opening socket");
41         exit(1);
42     }
43
44     /* Initialize socket structure */
45     bzero((char *) &serv_addr, sizeof(serv_addr));
46     portno = 5001;
47
48     serv_addr.sin_family = AF_INET;
49     serv_addr.sin_addr.s_addr = INADDR_ANY;
50     serv_addr.sin_port = htons(portno);
51
52     if(setsockopt(sockfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), &(int)
↪ {1}, sizeof(int)) < 0) {
53         perror("ERROR on setsockopt");
54         shutdown(sockfd, 2);
55         close(sockfd);
56     }
57
58     /* Now bind the host address using bind() call.*/
59     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
60         perror("ERROR on binding");
61         shutdown(sockfd, 2);
62         close(sockfd);
63         exit(1);
64     }
65
66     clilen = sizeof(cli_addr);
67
68     /* data which will be sent in listen thread */
69     data.sockfd = sockfd;
70     data.clilen = clilen;
71     data.cli_addr = cli_addr;
72
73     pthread_t listen_thread; //thread for listening
74     int result; //result of creating thread
75     result = pthread_create(&listen_thread, NULL, listen_func, &data); //create
listen thread
76     if(result != 0) {
77         perror("Error while creating listen_thread");
78         shutdown(sockfd, 2);
79         close(sockfd);
80         exit(1);
81     }
82
83     /* if pressed q then exit program */
84     int key = 0;
85     while(1) {
86         key = getchar();
87         if(key == 'q') break;

```

```

88 // enable/disable logging
89 if(key == 'l') {
90     log_enabled = !log_enabled;
91     log("log_enabled");
92 }
93 }
94 shutdown(sockfd, 2);
95 close(sockfd);
96 exit(0);
97 }
98
99 /* waiting for connection */
100 void *listen_func (void *arg) {
101     struct socket_data data = * (struct socket_data *) arg;
102     char buffer[256];
103     int n;
104     while(1) {
105         n = recvfrom(data.sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *) &data
↪ .cli_addr, &data.clilen);
106         if(n == -1) {
107             perror("ERROR_on_receive_data");
108             shutdown(data.sockfd, 2);
109             close(data.sockfd);
110             exit(1);
111         } else {
112             printf("Here_is_the_message:%s\n", buffer);
113             n = sendto(data.sockfd, "I_got_your_message", 18, 0, (struct sockaddr *) &
↪ data.cli_addr, data.clilen);
114             if(n == -1) {
115                 perror("ERROR_on_send_data");
116                 shutdown(data.sockfd, 2);
117                 close(data.sockfd);
118                 exit(1);
119             }
120             log("Message_sent\n");
121         }
122     }
123     pthread_exit(0);
124 }
125
126 void logv(char* msg, int value) {
127     if(log_enabled) {
128         printf("log:_%s_%d\n", msg, value);
129     }
130 }
131
132 void log(char* msg) {
133     if(log_enabled) {
134         printf("log:_%s\n", msg);
135     }
136 }
137 }

```

Листинг 9: CMakeLists для клиента

```

1 cmake_minimum_required(VERSION 2.8)
2 project(client_linux)
3
4 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
5
6 set(SOURCE_FILES main.c)

```

```
7 add_executable(client_linux ${SOURCE_FILES})
```

## Листинг 10: Код клиента

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <netdb.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
7
8 #include <string.h>
9
10 #include <unistd.h>
11
12 /* function for listen thread */
13 void *listen_func (void *arg);
14
15 /* variable for enable/disable logging */
16 int log_enabled = 0;
17
18 /* function for logging */
19 void log(char* msg, int value);
20
21 void log(char* msg);
22
23 struct socket_data {
24     int sockfd;
25     unsigned int clilen;
26     struct sockaddr_in cli_addr;
27 };
28
29 int main(int argc, char *argv[]) {
30     int sockfd;
31     uint16_t portno;
32     unsigned int clilen;
33     struct sockaddr_in serv_addr, cli_addr;
34     struct socket_data data;
35
36     /* First call to socket() function */
37     sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
38
39     if (sockfd < 0) {
40         perror("ERROR_opening_socket");
41         exit(1);
42     }
43
44     /* Initialize socket structure */
45     bzero((char *) &serv_addr, sizeof(serv_addr));
46     portno = 5001;
47
48     serv_addr.sin_family = AF_INET;
49     serv_addr.sin_addr.s_addr = INADDR_ANY;
50     serv_addr.sin_port = htons(portno);
51
52     if(setsockopt(sockfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), &(int)
↪ {1}, sizeof(int)) < 0) {
53         perror("ERROR_on_setsockopt");
54         shutdown(sockfd, 2);
55         close(sockfd);
56     }
```

```

57
58  /* Now bind the host address using bind() call.*/
59  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
60      perror("ERROR_on_binding");
61      shutdown(sockfd, 2);
62      close(sockfd);
63      exit(1);
64  }
65
66  clilen = sizeof(cli_addr);
67
68  /* data which will be sent in listen thread */
69  data.sockfd = sockfd;
70  data.clilen = clilen;
71  data.cli_addr = cli_addr;
72
73  pthread_t listen_thread; //thread for listening
74  int result; //result of creating thread
75  result = pthread_create(&listen_thread, NULL, listen_func, &data); //create
listen thread
76  if(result != 0) {
77      perror("Error_while_creating_listen_thread");
78      shutdown(sockfd, 2);
79      close(sockfd);
80      exit(1);
81  }
82
83  /* if pressed q then exit program */
84  int key = 0;
85  while(1) {
86      key = getchar();
87      if(key == 'q') break;
88      // enable/disable logging
89      if(key == 'l') {
90          log_enabled = !log_enabled;
91          log("log_enabled");
92      }
93  }
94  shutdown(sockfd, 2);
95  close(sockfd);
96  exit(0);
97 }
98
99 /* waiting for connection */
100 void *listen_func (void *arg) {
101     struct socket_data data = * (struct socket_data *) arg;
102     char buffer[256];
103     int n;
104     while(1) {
105         n = recvfrom(data.sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *) &data
➔ .cli_addr, &data.clilen);
106         if(n == -1) {
107             perror("ERROR_on_receive_data");
108             shutdown(data.sockfd, 2);
109             close(data.sockfd);
110             exit(1);
111         } else {
112             printf("Here_is_the_message:%s\n", buffer);
113             n = sendto(data.sockfd, "I_got_your_message", 18, 0, (struct sockaddr *) &
➔ data.cli_addr, data.clilen);
114             if(n == -1) {

```

```

115     perror("ERROR_on_send_data");
116     shutdown(data.sockfd, 2);
117     close(data.sockfd);
118     exit(1);
119     }
120     log("Message_sent\n");
121 }
122 }
123
124 pthread_exit(0);
125 }
126
127 void logv(char* msg, int value) {
128     if(log_enabled) {
129         printf("log:_%s_%d\n", msg, value);
130     }
131 }
132
133 void log(char* msg) {
134     if(log_enabled) {
135         printf("log:_%s\n", msg);
136     }
137 }

```

### 3.4. Индивидуальное задание: платежная система

#### 3.4.1. Задание

Разработать приложение–клиент и приложение–сервер платежной системы. Участники платежной системы имеют электронные кошельки. Электронный кошелек имеет уникальный номер. При регистрации пользователя в платежной системе на его счет зачисляется определенная сумма. Пользователя платежной системы могут осуществлять платежи друг другу через приложение–сервер.

Основные возможности. Серверное приложение должно реализовывать следующие функции:

1. Прослушивание определенного порта
2. Обработка запросов на подключение по этому порту от клиентов платежной системы
3. Поддержка одновременной работы нескольких клиентов платежной системы через механизм нитей
4. Прием запросов от клиента на регистрацию пользователя, передачу списка электронных кошельков пользователей платежной системы, осуществление платежей одного пользователя другому, проверка состояния счета кошелька
5. Осуществление добавления пользователя в платежную систему, хранение и изменение состояния электронных кошельков в зависимости от платежей пользователей
6. Передача запросов на платежи от одного пользователя другому, подтверждений платежей, номера нового кошелька при регистрации пользователя, списка электронных кошельков
7. Обработка запроса на отключение клиента

## 8. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Установление соединения с сервером
2. Передача запросов на передачу списка электронных кошельков пользователей платежной системы, платежи одного пользователя другому, проверку состояния счета кошелька
3. Получение от сервера запросов на платеж от другого пользователя, результатов платежа
4. Разрыв соединения
5. Обработка ситуации отключения клиента сервером

### 3.4.2. Описание реализации

Было написано два приложения - клиент и сервер. Сервер принимает запросы на порту 5001 и создает новые потоки для работы с каждым отдельным клиентом. Формат команды для запуска сервера: `./server_linux`.

Клиент подключается к серверу, используя его ip-адрес и порт. Формат команды для запуска клиента: `./client_linux <ip-адрес> 5001`.

Обмен данными происходит следующим образом: сервер ожидает запроса от клиента. Клиент посылает серверу команду и необходимые аргументы. Сервер выполняет запрос и отправляет ответ клиенту. Клиент принимает ответ и выводит необходимые данные на консоль, после чего цикл повторяется. Все протоколы и команды описаны далее.

### 3.4.3. Протокол запроса клиент-сервер

Клиент посылает на сервер запрос по следующему протоколу:

1. Длина запроса - 1 байт
2. Длина команды N - 1 байт
3. Команда - N байт
4. Длина первого аргумента M - 1 байт
5. Первый аргумент запроса - M байт
6. Длина второго аргумента K - 1 байт
7. Второй аргумент запроса - K байт

Команды, доступные на клиенте:

- GET - получить состояние счета клиента. Доступно только после входа в аккаунт.
- REG <login> <password> - зарегистрировать нового клиента. После регистрации сервер автоматически залогинит пользователя в созданном аккаунте.
- LOGIN <login> <password> - войти в аккаунт.

- SEND <amount> <login> - переслать заданную сумму пользователю с указанным логином.
- DEL - удалить текущий аккаунт.
- QUIT - выйти из приложения.
- HELP - вывести справку по командам.

#### 3.4.4. Протокол ответа сервер-клиент

Сервер отвечает на запросы клиента по следующему протоколу:

1. Длина ответа - 1 байт
2. Длина типа ответа N - 1 байт
3. Тип ответа - N байт
4. Длина тела ответа (payload) M - 1 байт
5. Тело ответа (payload) - M байт

Типы ответов сервера:

- RESPONSE\_OK - запрос успешно обработан, payload содержит запрашиваемую информацию, например, состояние счета клиента.
- RESPONSE\_ERROR - произошла ошибка. Например, клиент не вошел систему и хочет выполнить действие, требующее авторизации. Payload содержит сообщение об ошибке
- RESPONSE\_TOKEN - запрос на вход систему успешно обработан, payload содержит токен для сессии клиента.
- RESPONSE\_DELETED - запрос на выход из системы успешно обработан, на сервере удален токен для сессии. Payload содержит сообщение об успешности операции.

#### 3.4.5. Организация хранения данных

Данные хранятся в виде файлов в каталоге data, автоматически создаваемой при запуске сервера в той же папке. Каталог data содержит два подкаталога: clients и session.

В clients хранится информация о клиентах: их логин, пароль и состояние счета. Формат следующий: имя файла - логин пользователя. В файле в первой строке записано количество средств у пользователя, а во второй строке - пароль.

В session хранится информация о текущих сессиях. Формат следующий: имя файла - токен сессии. В файле записан логин пользователя, которому принадлежит этот токен.

### 3.4.6. Организация одновременной работы нескольких клиентов

Для одновременной работы нескольких клиентов на сервере был реализован механизм нитей. Есть главный поток, он принимает команды пользователя. Список команд на сервере:

- q - выход из программы, выключение сервера, удаление всех текущих сессий.
- l - выключение/выключение логирования в консоль.
- list - выводит список текущих сессий и токенов для них.
- del <token> - принудительно завершить сессию по ее токену.

В главном потоке создается слушающий поток, который принимает новые соединения с помощью функции `accept()`. Для каждого нового клиента создается отдельный клиентский поток, благодаря чему все клиенты работают независимо друг от друга.

### 3.4.7. Код проекта

Код проекта доступен по ссылке [https://github.com/K1ta/NetworksLab2018/tree/individual\\_task/i](https://github.com/K1ta/NetworksLab2018/tree/individual_task/i)

## 3.5. Socks5 прокси сервер

В качестве дополнительного проекта был реализован прокси сервер, работающий по протоколу Socks5. В качестве языка была использована Java8. Проект реализован с использованием неблокирующих сокетов из Java nio и может обеспечивать работу нескольких клиентов сразу.

### 3.5.1. Описание протокола

SOCKS 5 расширяет модель SOCKS 4, добавляя к ней поддержку UDP, обеспечение универсальных схем строгой аутентификации и расширяет методы адресации, добавляя поддержку доменных имен и адресов IPv6. Начальная установка связи теперь состоит из следующего:

- Клиент подключается, и посылает приветствие, которое включает перечень поддерживаемых методов аутентификации.
- Сервер выбирает из них один (или посылает ответ о неудаче запроса, если ни один из предложенных методов не приемлем).
- В зависимости от выбранного метода, между клиентом и сервером может пройти некоторое количество сообщений.
- Клиент посылает запрос на соединение, аналогично SOCKS 4.
- Сервер отвечает, аналогично SOCKS 4.

Методы аутентификации пронумерованы следующим образом:



0x00	Аутентификация не требуется
0x01	GSSAPI
0x02	Имя пользователя / пароль
0x03-0x7F	Зарезервировано IANA
0x80-0xFE	Зарезервировано для методов частного использования

Начальное приветствие от клиента:

Размер	Описание
1 байт	Номер версии SOCKS (должен быть 0x05 для этой версии)
1 байт	Количество поддерживаемых методов аутентификации
n байт	Номера методов аутентификации, переменная длина, 1 байт для каждого поддерживаемого метода

Сервер сообщает о своём выборе:

Размер	Описание
1 байт	Номер версии SOCKS (должен быть 0x05 для этой версии)
1 байт	Выбранный метод аутентификации или 0xFF, если не было предложено приемлемого метода

Последующая идентификация зависит от выбранного метода. Запрос клиента:

Размер	Описание
1 байт	Номер версии SOCKS (должен быть 0x05 для этой версии)
1 байт	Код команды: <ul style="list-style-type: none"> <li>0x01 = установка TCP/IP соединения</li> <li>0x02 = назначение TCP/IP порта (binding)</li> <li>0x03 = ассоциирование UDP-порта</li> </ul>
1 байт	Зарезервированный байт, должен быть 0x00
1 байт	Тип адреса: <ul style="list-style-type: none"> <li>0x01 = адрес IPv4</li> <li>0x03 = имя домена</li> <li>0x04 = адрес IPv6</li> </ul>
Зависит от типа адреса	Назначение адреса: <ul style="list-style-type: none"> <li>4 байта для адреса IPv4</li> <li>Первый байт — длина имени, затем следует имя домена без завершающего нуля на конце</li> <li>16 байт для адреса IPv6</li> </ul>
2 байта	Номер порта, в порядке от старшего к младшему (big-endian)

Ответ сервера:

Размер	Описание
1 байт	Номер версии SOCKS (0x05 для этой версии)
1 байт	Код ответа: <ul style="list-style-type: none"> <li>• 0x00 = запрос предоставлен</li> <li>• 0x01 = ошибка SOCKS-сервера</li> <li>• 0x02 = соединение запрещено набором правил</li> <li>• 0x03 = сеть недоступна</li> <li>• 0x04 = хост недоступен</li> <li>• 0x05 = отказ в соединении</li> <li>• 0x06 = истечение <b>TTL</b></li> <li>• 0x07 = команда не поддерживается / ошибка протокола</li> <li>• 0x08 = тип адреса не поддерживается</li> </ul>
1 байт	Байт зарезервирован, должен быть 0x00
1 байт	Тип последующего адреса: <ul style="list-style-type: none"> <li>• 0x01 = адрес IPv4</li> <li>• 0x03 = имя домена</li> <li>• 0x04 = адрес IPv6</li> </ul>
Зависит от типа адреса	Назначение адреса: <ul style="list-style-type: none"> <li>• 4 байта для адреса IPv4</li> <li>• Первый байт — длина имени, затем следует имя домена без завершающего нуля на конце</li> <li>• 16 байт для адреса IPv6</li> </ul>
2 байта	Номер порта, в порядке от старшего к младшему ( <b>big-endian</b> )

### 3.5.2. Исходный код

Листинг 11: Socks5Server.java

```

1 package socks5;
2
3 import java.io.IOException;
4 import java.net.InetAddress;
5 import java.net.InetSocketAddress;
6 import java.nio.channels.SelectionKey;
7 import java.nio.channels.Selector;
8 import java.nio.channels.ServerSocketChannel;
9 import java.nio.channels.SocketChannel;
10 import java.util.Arrays;
11 import java.util.Iterator;
12 import java.util.Set;
13 import java.util.StringTokenizer;
14 import java.util.logging.Logger;
15
16 public class Socks5Server {
17     private static int port = 1080;
18     private static Logger logger = Logger.getLogger(Socks5Server.class.getName()
19 ↪ );
20     public static void main(String[] args) {
21         try {
22             logger.info("Server_starting..");
23             Selector selector = Selector.open();
24             ServerSocketChannel serverSocketChannel = ServerSocketChannel.open()
25 ↪ ;
26             logger.info("ServerSocketChannel_is_opened");

```

```

26     serverSocketChannel.bind(new InetSocketAddress(port));
27     logger.info("ServerSocketChannel_is_bind");
28     serverSocketChannel.configureBlocking(false);
29     logger.info("ServerSocketChannel_is_configured_non-blocking");
30     serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
31     while (true) {
32         selector.select();
33         Set<SelectionKey> selectedKeys = selector.selectedKeys();
34         Iterator<SelectionKey> it = selectedKeys.iterator();
35         while (it.hasNext()) {
36             SelectionKey key = it.next();
37             if (key.isAcceptable()) {
38                 accept(key);
39             } else if (key.isConnectable()) {
40                 connect(key);
41             } else if (key.isReadable()) {
42                 read(key);
43             } else if (key.isWritable()) {
44                 write(key);
45             }
46             it.remove();
47         }
48     }
49     } catch (IOException e) {
50         e.printStackTrace();
51     }
52 }
53
54 /*
55 принять запрос на подключение от клиента
56 */
57 private static void accept(SelectionKey key) throws IOException {
58     //принимаем запрос
59     SocketChannel newChannel = ((ServerSocketChannel) key.channel()).accept
60     ↪ ();
61     //делаем неблокирующим
62     newChannel.configureBlocking(false);
63     //регистрируем в селекторе
64     newChannel.register(key.selector(), SelectionKey.OP_READ);
65 }
66
67 /*
68 соединение с другим сервером завершено
69 */
70 private static void connect(SelectionKey key) throws IOException {
71     SocketChannel channel = (SocketChannel) key.channel();
72     Attachment attachment = (Attachment) key.attachment();
73     //завершаем соединение
74     channel.finishConnect();
75
76     //набираем сообщение, которое нужно отправить клиенту
77     //нам нужен адрес, по которому мы будем связываться с удаленным сервером
78     StringBuildersb = new StringBuildersb(channel.getLocalAddress().toString
79     ↪ ());
80     sb = sb.replace(0, 1, "");
81     StringTokenizer st = new StringTokenizer(sb.toString(), ":");
82     InetAddress ip = InetAddress.getByAddress(st.nextToken());
83     int intPort = Integer.parseInt(st.nextToken());
84     byte[] bytesIp = ip.getAddress();

```

```

83     byte[] bytesPort = new byte[] {(byte) ((intPort & 0xFF00) >> 8), (byte) (
↪ intPort & 0xFF)};
84
85     Attachment peerAttachment = (Attachment) attachment.peer.attachment();
86     peerAttachment.writeBuffer.clear();
87     //сообщаем, что все ок
88     peerAttachment.writeBuffer.put(new byte[] {0x05, 0x00, 0x00, 0x01});
89     //добавляем в сообщение адрес и порт
90     peerAttachment.writeBuffer.put(bytesIp);
91     peerAttachment.writeBuffer.put(bytesPort);
92
93     //добавляем интерес на запись клиенту
94     attachment.peer.interestOps(SelectionKey.OP_WRITE);
95     System.out.println("Connected");
96 }
97
98 /*
99 прочитать запрос от клиента
100 */
101 private static void read(SelectionKey key) throws IOException {
102     SocketChannel clientChannel = (SocketChannel) key.channel();
103     Attachment clientAttachment = (Attachment) key.attachment();
104     if (clientAttachment == null) {
105         //если у клиента нет attachment, то значит, он у нас в первый раз
106         //нужно проверить запрос на соответствие шаблону и отправить номер версии и
тип аутентификации
107         clientAttachment = new Attachment();
108         //читаем запрос от клиента
109         int code = clientChannel.read(clientAttachment.readBuffer);
110         if (code < 1) {
111             //если -1 - то разрыв соединения, 0 - нет места в буфере
112             close(key);
113             return;
114         }
115         //печатаем прочитанное в консоль
116         System.out.println("Read1_from_" + clientChannel.getRemoteAddress()
↪ + " :_" + Arrays.toString(clientAttachment.readBuffer.array()));
117         //проверяем сообщение на соответствие приветствию
118         if (clientAttachment.readBuffer.get(0) == 0x05 && clientAttachment.
↪ readBuffer.get(1) == 0x01) {
119             //отвечаем версией протокола и типом аутентификации
120             clientAttachment.writeBuffer.put(((byte) 0x05));
121             clientAttachment.writeBuffer.put(((byte) 0x00));
122             key.interestOps(SelectionKey.OP_WRITE);
123         } else {
124             throw new IllegalStateException("Bad_Request");
125         }
126     } else if (clientAttachment.peer == null) {
127         //если у клиента есть attachment, но нет peer, то это запрос с адресом для
подключения
128         //нужно прочитать адрес и порт и создать соединение с полученным адресом
129         //также нужно зарегистрировать это соединение, создать SelectionKey и настроить
peer клиента и нового key
130         clientAttachment.readBuffer.clear();
131         //читаем запрос от клиента
132         int code = clientChannel.read(clientAttachment.readBuffer);
133         if (code < 1) {
134             //если -1 - то разрыв соединения, 0 - нет места в буфере
135             close(key);
136             return;
137         }

```

```

138         //печатаем прочитанное в консоль
139         System.out.println("Read2_from_" + clientChannel.getRemoteAddress()
    ↪ + ":_" + Arrays.toString(clientAttachment.readBuffer.array()));
140         //проверяем сообщение на соответствие
141         byte[] rb = clientAttachment.readBuffer.array();
142         if (!(rb[0] == 0x05 && rb[1] == 0x01 && rb[2] == 0x00 && rb[3] == 0
    ↪ x01)) {
143             System.out.println("array:_" + Arrays.toString(rb));
144             throw new IllegalStateException("Bad_Request");
145         }
146         //читаем адрес и порт
147         byte[] addr = new byte[]{rb[4], rb[5], rb[6], rb[7]};
148         int p = ((rb[8] & 0xFF) << 8) | (rb[9] & 0xFF);
149         //создаем новый сокет
150         SocketChannel newSocketChannel = SocketChannel.open();
151         //делаем его неблокирующим
152         newSocketChannel.configureBlocking(false);
153         //коннектимся по прочитанному от клиента адресу к серверу телеграма
154         newSocketChannel.connect(new InetSocketAddress(InetAddress.
    ↪ getByAddress(addr), p));
155         //регистрируем сокет в селекторе
156         SelectionKey newSelectionKey = newSocketChannel.register(key.
    ↪ selector(), SelectionKey.OP_CONNECT);
157         //теперь настраиваем peer клиента на новый key и peer newSelectionKey на клиента
158         //writeBuffer клиента - это readBuffer peer'a и наоборот
159         Attachment newAttachment = new Attachment();
160         newAttachment.peer = key;
161         newAttachment.writeBuffer = clientAttachment.readBuffer;
162         newAttachment.readBuffer = clientAttachment.writeBuffer;
163         clientAttachment.peer = newSelectionKey;
164         newSelectionKey.attach(newAttachment);
165     } else {
166         //если у клиента уже все есть, то обмениваемся сообщениями
167         //читаем из attachment клиента readBuffer
168         //и передаем его в attachment пира в writeBuffer
169         clientAttachment.readBuffer.clear();
170         //читаем запрос от клиента
171         int code = clientChannel.read(clientAttachment.readBuffer);
172         if (code < 1) {
173             //если -1 - то разрыв соединения, 0 - нет места в буфере
174             close(key);
175             return;
176         }
177         //печатаем прочитанное в консоль
178         System.out.println("Read3_from_" + clientChannel.getRemoteAddress()
    ↪ + ":_" + Arrays.toString(clientAttachment.readBuffer.array()));
179         key.interestOps(key.interestOps() ^ SelectionKey.OP_READ);
180         //ставим интерес прочитать пиру клиента
181         clientAttachment.peer.interestOps(SelectionKey.OP_WRITE);
182     }
183     key.attach(clientAttachment);
184 }
185
186 /*
187 отправить ответ клиенту
188 */
189 private static void write(SelectionKey key) throws IOException {
190     SocketChannel clientChannel = (SocketChannel) key.channel();
191     Attachment clientAttachment = (Attachment) key.attachment();
192     //этот метод отправляет серверу или клиенту данные из буфера writeBuffer

```

```

193     System.out.println("Write_to_" + clientChannel.getRemoteAddress() + ":\n"
194     ↪ + Arrays.toString(clientAttachment.writeBuffer.array()));
195     clientAttachment.writeBuffer.flip();
196     if (clientChannel.write(clientAttachment.writeBuffer) == -1) {
197         //если -1 - то разрыв соединения, 0 - нет места в буфере
198         close(key);
199         return;
200     }
201     if (clientAttachment.writeBuffer.remaining() == 0) {
202         if (clientAttachment.peer == null) {
203             //если пира еще нет, то готовимся принимать ответ
204             key.interestOps(SelectionKey.OP_READ);
205         } else {
206             //если всё записано, чистим буфер
207             clientAttachment.writeBuffer.clear();
208             //добавляем пиру интерес на чтение
209             clientAttachment.peer.interestOps(SelectionKey.OP_READ);
210             //убираем интерес на запись
211             key.interestOps(SelectionKey.OP_READ);
212         }
213     }
214
215     private static void close(SelectionKey key) throws IOException {
216         key.cancel();
217         key.channel().close();
218         Attachment attachment = ((Attachment) key.attachment());
219         if (attachment != null && attachment.peer != null) {
220             Attachment peerKey = ((Attachment) attachment.peer.attachment());
221             if (peerKey != null && peerKey.peer != null) {
222                 peerKey.peer.cancel();
223             }
224             attachment.peer.cancel();
225         }
226     }
227 }
228 }

```

ЛИСТИНГ 12: Attachment.java

```

1 package socks5;
2
3 import java.nio.ByteBuffer;
4 import java.nio.channels.SelectionKey;
5
6 public class Attachment {
7
8     SelectionKey peer;
9
10    ByteBuffer writeBuffer;
11    ByteBuffer readBuffer;
12
13    public Attachment() {
14        writeBuffer = ByteBuffer.allocate(8*1024);
15        readBuffer = ByteBuffer.allocate(8*1024);
16    }
17
18 }

```

## 4. Выводы

В ходе выполнения работ были изучены и использованы на практике протоколы TCP и UDP. Также были рассмотрены особенности программирования сокетов на ОС Linux и Windows.

Для реализации первого индивидуального задания - платежной системы - был выбран протокол TCP, так как в этой работе нам нужно обеспечить целостность данных, что с TCP сделать значительно проще, чем с UDP.

Для организации работы нескольких клиентов был реализован механизм на основе потоков: слушающий поток принимает входящие соединения и создает отдельный поток для каждого клиента.

Для хранения данных использовались обычные файлы, разделенные на две группы: данные о текущих сессиях (data/session) и данные о клиентах (data/clients). Данные хранятся в незашифрованном виде, что является потенциальной уязвимостью, однако это уже выходит за рамки задания.

Для логирования была написана собственная библиотека, поддерживающая форматированный вывод. Использование похоже на функцию printf, но написанная библиотека выводит данные на консоль и в файл server.log. Также выводится дополнительная информация: время, в которое было совершено логирование, имя функции и идентификатор потока, вызвавшего функцию логирования.

Тестирование производилось вручную с запуском сервера и нескольких клиентов, после чего анализировались логи сервера. В ходе проверки была использована утилита gdb для тестирования в консоли, и Wireshark для просмотра пакетов, передаваемых между приложениями. Также были использованы автоматические тесты с использованием ПО, предоставленного преподавателем.

В качестве второго индивидуального задания был реализован прокси сервер на протоколе Socks5. Это сетевой протокол, который позволяет пересылать пакеты от клиента к серверу через прокси-сервер прозрачно (незаметно для них) и таким образом использовать сервисы за межсетевыми экранами (фаерволами).

Для реализации использовалась Java 8 с неблокирующими сокетами. Для тестирования был использован сервис AWS EC2. С помощью ssh был настроен удаленный сервер, после чего на нем был запущен прокси сервер. Далее сервер использовался для получения и отправки данных в Telegram.