



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический  
университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

---

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

## Лабораторная работа № 6

Дисциплина            Операционные системы.

Тема                    Сокеты

Студент               Степанов А. О.

Группа                ИУ7-63Б

Оценка (баллы)

Преподаватель      Рязанова Н.Ю.

Москва, 2020 г.

# СОКЕТ В ФАЙЛОВОМ ПРОСТРАНСТВЕ

Листинг 1: Функция error

```
1 int error()
2 {
3     fprintf(stderr, "%s\n", strerror(errno));
4     return errno;
5 }
```

Сокеты в файловом пространстве используют в качестве адреса имя файла. В данной программе сервер создает сокет при помощи функции `socket` (с параметрами `AF_UNIX`, что означает, что сокет будет в файловом пространстве, и `SOCK_DGRAM`, что означает, что сокет будет датаграммным) и связывает его с файлом `socket.soc` при помощи функции `bind`. После этого можно получать сообщения от клиентов при помощи функции `recvfrom`.

Листинг 2: Текст программы сервера

```
1 #include <string.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <signal.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #include "error.h"
8
9 #define SOCK_NAME "socket.soc"
10 #define SIZE_BUFFER 100
11 int sockfd;
12
13 void close_app()
14 {
15     if (close(sockfd) < 0)
16         exit(error());
17
18     if (unlink(SOCK_NAME) < 0)
19         exit(error());
20
21     printf("\nSocket_closed\n");
22     exit(0);
23 }
24
25 int main()
```

```

26 {
27     struct sockaddr socket_file;
28     struct sockaddr recv_name;
29     socklen_t recv_len;
30     char buffer[SIZE_BUFFER];
31     ssize_t size;
32
33     signal(SIGINT, close_app);
34
35     sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
36
37     if (sockfd < 0)
38         return error();
39
40     socket_file.sa_family = AF_UNIX;
41     strcpy(socket_file.sa_data, SOCK_NAME);
42
43     bind(
44         sockfd, &socket_file,
45         strlen(socket_file.sa_data) + sizeof(socket_file.sa_family) + 1
46     );
47
48     if (errno != 0)
49         return error();
50
51     printf("Socket_is_successfully_opened\n");
52     printf("Press_Ctrl+C_to_close_this_application\n");
53
54     while (1)
55     {
56         size = recvfrom(
57             sockfd, buffer, sizeof(buffer), 0,
58             &recv_name, &recv_len
59         );
60
61         if (size < 0)
62             return error();
63
64         buffer[size] = '\0';
65         printf("MESSAGE_RECIEVED: %s", buffer);
66     }
67

```

```

68     return 0;
69 }

```

Клиент создает сокет с такими же параметрами, как и сервер, после чего задает тип домена и имя файла в структуру `sockaddr`, с помощью которой и отправляет сообщение серверу функцией `sendto`.

### Листинг 3: Текст программы клиента

```

1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <unistd.h>
4  #include <signal.h>
5  #include <stdlib.h>
6  #include "error.h"
7
8  #define SIZE_BUFFER 100
9
10 int sockfd;
11
12 void close_app()
13 {
14     if (close(sockfd) < 0)
15         exit(error());
16
17     exit(0);
18 }
19
20 int main()
21 {
22     size_t len;
23     struct sockaddr srvr_name;
24     char *buffer;
25
26     signal(SIGINT, close_app);
27
28     sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
29
30     if (sockfd < 0)
31         return error();
32
33     srvr_name.sa_family = AF_UNIX;
34     strcpy(srvr_name.sa_data, "socket.soc");

```

```

35
36 while(1)
37 {
38     printf("Type_message:_");
39     getline(&buffer, &len, stdin);
40     sendto(
41         sockfd, buffer, strlen(buffer), 0, &srvr_name,
42         strlen(srvr_name.sa_data) + sizeof(srvr_name.sa_family) + 1
43     );
44 }
45
46 return 0;
47 }

```

```

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./server.out
Socket is successfully opened
Press Ctrl+C to close this application
MESSAGE RECIEVED: 1
MESSAGE RECIEVED: 2
MESSAGE RECIEVED: 3
MESSAGE RECIEVED: 4
MESSAGE RECIEVED: 5
MESSAGE RECIEVED: 6
MESSAGE RECIEVED: 7
MESSAGE RECIEVED: 8
MESSAGE RECIEVED: 9
MESSAGE RECIEVED: 10
MESSAGE RECIEVED: 11
MESSAGE RECIEVED: 12
MESSAGE RECIEVED: 13
MESSAGE RECIEVED: 14
MESSAGE RECIEVED: 15
MESSAGE RECIEVED: 16
MESSAGE RECIEVED: 17
MESSAGE RECIEVED: 18
MESSAGE RECIEVED: 19
MESSAGE RECIEVED: 20
[]

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./client.out
Type message: 1
Type message: 6
Type message: 11
Type message: 20
Type message: []

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./client.out
Type message: 2
Type message: 7
Type message: 12
Type message: 19
Type message: []

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./client.out
Type message: 3
Type message: 8
Type message: 13
Type message: 18
Type message: []

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./client.out
Type message: 4
Type message: 9
Type message: 14
Type message: 17
Type message: []

~/Repositories/operating-systems/sem_02/lab_06/src/part_01 feature/
lab_06_sem_02 1 2 3
$ ./client.out
Type message: 5
Type message: 10
Type message: 15
Type message: 16
Type message: []

```

Рис. 1: Результат работы программы

## СЕТЕВОЙ СОКЕТ

Сервер создает сокет с параметрами `AF_INET` (сетевое соединение) и `SOCK_STREAM` (поточковый сокет). С помощью функции `bind` сокет связывается с адресом. После этого используется функция `listen`, которая переводит сервер в режим ожидания

запроса на соединение. Затем создается массив дескрипторов для будущих подключаемых клиентов. Если дескриптор равен -1, то значит он свободен для новых клиентов. После сделанных действий можно ожидать подключения от клиентов. В функции **connection**, которая служит в этой программе для подключения новых клиентов, с помощью функции **accept** устанавливается соединение, затем находим свободный элемент в массиве дескрипторов и добавляем нового клиента, получив от него первое сообщение, содержащее его pid. Функция **message** принимает сообщения от подключенных клиентов и проверяет на отключение от сервера. В этой функции производится проверка дескрипторов всех подключенных клиентов. Если появляется возможность чтения или записи из дескриптора, то происходит проверка на соединение с клиентом. Если соединения нет, то освобождается дескриптор и выводится сообщение об отключении клиента. В ином случае выводится полученное сообщение и отправляется ответное.

#### Листинг 4: Текст программы сервера

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <strings.h>
6 #include <netinet/in.h>
7 #include <signal.h>
8 #include "error.h"
9
10 #define SIZE_BUFFER 100
11 #define CLIENTS 100
12
13 int sockfd;
14 int maxi, maxfd;
15 int pid_client[CLIENTS];
16
17 void close_signal()
18 {
19     if (close(sockfd) < 0)
20         exit(error());
21
22     printf("\nSocket_closed\n");
23     exit(0);
24 }
25
```

```

26 int connection(int client[FD_SETSIZE], fd_set *allset, fd_set *rset)
27 {
28     int i;
29     int connfd;
30     int message_len;
31     char buffer[SIZE_BUFFER];
32
33     if (FD_ISSET(sockfd, rset))
34     {
35         connfd = accept(sockfd, NULL, NULL);
36
37         if (connfd < 0)
38             return errno;
39
40         for (i = 0; i < FD_SETSIZE; i++)
41         {
42             if (client[i] < 0)
43             {
44                 client[i] = connfd;
45                 break;
46             }
47         }
48
49         if (i == FD_SETSIZE)
50             return errno;
51
52         FD_SET(connfd, allset);
53
54         if (connfd > maxfd)
55             maxfd = connfd;
56
57         if (i > maxi)
58             maxi = i;
59
60         message_len = read(connfd, buffer, SIZE_BUFFER);
61         pid_client[i] = atoi(buffer);
62         printf("[%d]_connected\n", pid_client[i]);
63     }
64
65     return 0;
66 }
67

```

```

68 int message(int client[FD_SETSIZE], fd_set *allset, fd_set *rset)
69 {
70     int n, i;
71     int sockfd;
72     char buffer[SIZE_BUFFER];
73
74     for (i = 0; i <= maxi; i++)
75     {
76         sockfd = client[i];
77         if (sockfd > 0)
78         {
79             if (FD_ISSET(sockfd, rset))
80             {
81                 n = read(sockfd, buffer, SIZE_BUFFER);
82
83                 if (n == 0)
84                 {
85                     close(sockfd);
86                     FD_CLR(sockfd, allset);
87                     client[i] = -1;
88                     printf("[%d]_disconnected\n", pid_client[i]);
89                 }
90                 else
91                 {
92                     write(sockfd, "good", 4);
93                     printf("[%d]_:_%s", pid_client[i], buffer);
94                 }
95             }
96         }
97     }
98
99     return 0;
100 }
101
102 int main(int argc, char **argv)
103 {
104     if (argc != 2)
105     {
106         fprintf(stderr, "Usage:_%s_<port>\n", argv[0]);
107         return -1;
108     }
109

```



```

110     int client[FD_SETSIZE];
111     fd_set rset;
112     fd_set allset;
113     struct sockaddr_in server;
114
115     signal(SIGINT, close_signal);
116
117     sockfd = socket(AF_INET, SOCK_STREAM, 0);
118
119     if (sockfd < 0)
120         return errno;
121
122     server.sin_family = AF_INET;
123     server.sin_addr.s_addr = INADDR_ANY;
124     server.sin_port = htons(atoi(argv[1]));
125
126     bind(sockfd, (struct sockaddr *) &server, sizeof(server));
127
128     if (errno != 0)
129         return errno;
130
131     listen(sockfd, CLIENTS);
132
133     maxfd = sockfd;
134     maxi = -1;
135
136     for (int i = 0; i < FD_SETSIZE; i++)
137         client[i] = -1;
138
139     FD_ZERO(&allset);
140     FD_SET(sockfd, &allset);
141
142     printf("Socket_is_successfully_opened\n");
143     printf("Press_Ctrl+C_to_close_this_application\n");
144
145     while(1)
146     {
147         rset = allset;
148         select(maxfd + 1, &rset, NULL, NULL, NULL);
149
150         connection(client, &allset, &rset);
151

```

```

152         if (errno != 0)
153             return error();
154
155         message(client , &allset , &rset);
156
157         if (errno != 0)
158             return error();
159     }
160
161     return 0;
162 }

```

Клиент создает сокет с теми же параметрами, как и сервер. После этого, клиенту необходимо подключиться по адресу. Это делается с помощью структуры `sockaddr_in`, поля которой необходимо заполнить информацией о сервере, к которому мы хотим подключиться. Функцией `connect` производится подключение. После успешного подключения, клиент отправляет свой pid сообщением и начинает отправлять сообщения, введенные пользователем. После каждого отправленного сообщения клиент ждет ответного сообщения от сервера. Отправка сообщения производится функцией `write`, получение – `read`.

#### Листинг 5: Текст программы клиента

```

1  #include <stdint.h>
2  #include <stdio.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <unistd.h>
6  #include <netinet/in.h>
7  #include <netdb.h>
8  #include <stdlib.h>
9  #include <signal.h>
10 #include "error.h"
11
12 #define SIZE_BUFFER 100
13
14 int sockfd;
15
16 void close_signal()
17 {
18     if (close(sockfd) < 0)
19         exit(error());

```

```

20
21     printf("\nSocket_closed\n");
22     exit(0);
23 }
24
25 int main(int argc, char *argv[])
26 {
27     if (argc != 3)
28     {
29         printf("Usage: %s <servername> <port>", argv[0]);
30         return -1;
31     }
32
33     signal(SIGINT, close_signal);
34
35     struct hostent *server;
36     struct sockaddr_in serv_addr;
37     char *buffer = NULL;
38     size_t len;
39     char buffer_server[SIZE_BUFFER];
40
41     sockfd = socket(AF_INET, SOCK_STREAM, 0);
42
43     if (sockfd < 0)
44         return error();
45
46     server = gethostbyname(argv[1]);
47
48     if (server == NULL)
49         return error();
50
51     serv_addr.sin_family = AF_INET;
52     strncpy(
53         (char *)&serv_addr.sin_addr.s_addr,
54         (char *)server->h_addr, server->h_length
55     );
56     serv_addr.sin_port = htons(atoi(argv[2]));
57
58     connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
59
60     if (errno != 0)
61         return error();

```

```

62
63     buffer = calloc(10, 1);
64     sprintf(buffer, "%d", getpid());
65     write(sockfd, buffer, strlen(buffer));
66
67     free(buffer);
68     buffer = NULL;
69
70     printf("Type_message:_");
71     getline(&buffer, &len, stdin);
72     buffer[len] = '\0';
73
74     while (strcmp(buffer, "exit\n"))
75     {
76         write(sockfd, buffer, len);
77
78         memset(buffer_server, 0, SIZE_BUFFER);
79         read(sockfd, buffer_server, SIZE_BUFFER);
80         printf("Message_received:_%s\n", buffer_server);
81
82         printf("Type_message:_");
83         getline(&buffer, &len, stdin);
84         buffer[len] = '\0';
85     }
86
87     return 0;
88 }

```

```
I II III IV V VI alex@MacBook-Pro-Aleksandr: ~/Repositories/operating-systems/sem_02/lab_06/src/part_02 98% 07:21
~/Repositories/operating-systems/sem_02/lab_06/src/part_02 feature/
lab_06_sem_02 1 2 3
$ ./server.out 1
Socket is successfully opened
Press Ctrl+C to close this application
[39943] connected
[39948] connected
[39960] connected
[39943] : before all connections
[39968] connected
[39968] disconnected
[39971] connected
[39985] connected
[39943] : 1
[39948] : 2
[39960] : 3
[39971] : 4
[39985] : 5
[39985] disconnected
[39960] disconnected
[39948] : 6
[39943] : 7
[39943] disconnected
[39948] disconnected
[39971] : hello
[39971] disconnected
[]

$ ./client.out 1 1
Type message: before all connections
Message received: good
Type message: 1
Message received: good
Type message: 7
Message received: good
Type message: ^C

lab_06_sem_02 1 2 3
$ ./client.out 1 1
Type message: 2
Message received: good
Type message: 6
Message received: good
Type message: ^C

$ ./client.out 1 1
Type message: 3
Message received: good
Type message: ^C

~/Repositories/operating-systems/sem_02/lab_06/src/part_02 feature/
lab_06_sem_02 1 2 3
$ []

lab_06_sem_02 1 2 3
$ ./client.out 1 1
Type message: 4
Message received: good
Type message: hello
Message received: good
Type message: ^C

$ ./client.out 1 1
Type message: 5
Message received: good
Type message: ^C

~/Repositories/operating-systems/sem_02/lab_06/src/part_02 feature/
lab_06_sem_02 1 2 3
$ []
```

Рис. 2: Результат работы программы