

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 Куприй А. А.
Группа ИУ7-63Б
Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

ВВЕДЕНИЕ

Сокеты были созданы как универсальное средство взаимодействия параллельных процессов, причем безразлично, где они выполняются: на одной машине или на разных. Абстракция сокетов была введена в BSD Unix, поэтому сокет часто называют сокетом BSD (Berkley Software Distribution).

Цель: проанализировать особенности работы сетевых сокетов и сокетов в пространстве файловых имён.

Задание: Лабораторная работа делится на две части. В первой части необходимо продемонстрировать взаимодействие параллельных процессов с использованием сокетов в файловом пространстве имён, а во второй - с помощью сетевых сокетов.

1 Практическая часть

Лабораторная работа состоит из двух частей:

- 1) Организовать взаимодействие параллельных процессов на отдельном компьютере.
- 2) Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине).

1.1 Задание №1

Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство - AF_UNIX, тип - SOCK_DGRAM. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Листинг 1.1 — Текст программы клиента из первого задания

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <errno.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <unistd.h>
8 #include <stdlib.h>
9
10 #define SOCK_NAME "socket.soc"
11 #define BUF_SIZE 256
12
13 int main(int argc, char ** argv)
14 {
15     int sock = socket(AF_UNIX, SOCK_DGRAM, 0);
16     char buf[BUF_SIZE];
17     char msg[BUF_SIZE];
18     struct sockaddr srvr_name;
19
20     if (sock < 0)
21     {
22         perror("socket failed");
```

```

23     return EXIT_FAILURE;
24 }
25
26     srvr_name.sa_family = AF_UNIX;
27     strcpy(srvr_name.sa_data, SOCK_NAME);
28
29     printf("message: ");
30     scanf("%255s", msg);
31
32     sprintf(buf, "client %d: %s", getpid(), msg);
33
34     sendto(sock, buf, strlen(buf), 0, &srvr_name,
35           strlen(srvr_name.sa_data) + sizeof(srvr_name.sa_family));
36
37     return 0;
38 }

```

В процессе-клиенте создаётся сокет в файловом пространстве имён (домен AF_UNIX) с типом SOCK_DGRAM, что означает, что это датаграммный сокет, затем передаётся сообщение серверу с помощью функции sendto().

Листинг 1.2 — Текст программы сервера из первого задания

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <errno.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <unistd.h>
8
9  #define SOCK_NAME "socket.soc"
10 #define BUF_SIZE 256
11
12 int main(int argc, char ** argv)
13 {
14     struct sockaddr srvr_name;
15     char buf[BUF_SIZE];
16     int sock;
17     int namelen, bytes;
18
19     sock = socket(AF_UNIX, SOCK_DGRAM, 0);
20     if (sock < 0)
21     {
22         perror("socket failed");
23         return EXIT_FAILURE;

```

```

24     }
25     srvr_name.sa_family = AF_UNIX;
26     strcpy(srvr_name.sa_data, SOCK_NAME);
27     if (bind(sock, &srvr_name, strlen(srvr_name.sa_data) +
28         sizeof(srvr_name.sa_family)) < 0)
29     {
30         perror("bind failed");
31         return EXIT_FAILURE;
32     }
33
34     printf("Server is running.\n");
35     while (strcmp(buf, "break"))
36     {
37         bytes = recvfrom(sock, buf, sizeof(buf), 0, NULL, NULL);
38         if (bytes < 0)
39         {
40             perror("recvfrom failed");
41             close(sock);
42             unlink(SOCK_NAME);
43             return EXIT_FAILURE;
44         }
45         buf[bytes] = 0;
46         printf("Client sent: %s\n", buf);
47     }
48
49     close(sock);
50     unlink(SOCK_NAME);
51 }

```

В процессе-сервере также создаётся датаграммный сокет в файловом пространстве имён, после чего происходит связка сокета с заданным адресом с помощью функции `bind()`. Затем в цикле происходит обработка сообщений из клиентов. Для чтения используется функция `recvfrom()`, блокирующая программу до тех пор, пока на вход не придут новые данные. Сервер обрабатывает сообщения до тех пор, пока клиент не отправит сообщение "break". После чего сокет закрывается и файл сокета удаляется.

Пример работы сервера, принимающего сообщения от клиентов:

```

san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_1
./server
Server is running.
Client sent: client 4602: Hello
Client sent: client 4700: world

```

Рисунок 1.1 — Скришот работы сервера.

Пример работы клиентов:

```

san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_1 ./client
message: Hello
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_1 ./client
message: world
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_1

```

Рисунок 1.2 — Скришот работы клиента.

Примеры, показывающие сокет в файловой системе (socket.soc):

```

san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_1 ls -l
total 32
-rwxrwxr-x 1 san_sanchez san_sanchez 8648 мая 10 05:23 client
-rw-rw-r-- 1 san_sanchez san_sanchez 783 мая 10 05:23 client.c
-rwxrwxr-x 1 san_sanchez san_sanchez 8752 мая 10 05:23 server
-rw-rw-r-- 1 san_sanchez san_sanchez 1141 мая 1 20:49 server.c
srwxrwxr-x 1 san_sanchez san_sanchez 0 мая 10 05:24 socket.soc

```

Рисунок 1.3 — Скришот команды ls -l.

1.2 Задание №2

Написать приложение по модели клиент-сервер, осуществляющее взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Листинг 1.3 — Текст программы клиента

```

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

```

```

3 #include <errno.h>
4 #include <string.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <netdb.h>
9 #include <unistd.h>
10
11 #define BUF_SIZE 256
12
13 int main(int argc, char ** argv)
14 {
15     int sock, port;
16     int pid;
17     struct sockaddr_in serv_addr;
18     struct hostent *server;
19     char buf[BUF_SIZE];
20     char buf_answer[BUF_SIZE];
21
22     if (argc < 3)
23     {
24         fprintf(stderr, "usage: %s <hostname> <port_number>\n", argv[0]);
25         return EXIT_FAILURE;
26     }
27
28     pid = getpid();
29
30     sock = socket(AF_INET, SOCK_STREAM, 0);
31     if (sock < 0)
32     {
33         printf("socket() failed: %d", errno);
34         return EXIT_FAILURE;
35     }
36
37     server = gethostbyname(argv[1]);
38     if (server == NULL)
39     {
40         printf("Host not found\n");
41         return EXIT_FAILURE;
42     }
43
44     serv_addr.sin_family = AF_INET;
45     memcpy((char *)&serv_addr.sin_addr.s_addr, (char *)server->h_addr,
46           server->h_length);
47     port = atoi(argv[2]);
48     serv_addr.sin_port = htons(port);

```

```

49
50     if (connect(sock, (struct sockaddr*) &serv_addr,
51                 sizeof(serv_addr)) < 0)
52     {
53         printf("connect() failed: %d", errno);
54         return EXIT_FAILURE;
55     }
56
57     sprintf(buf, "%d", pid);
58     write(sock, buf, BUF_SIZE);
59
60     printf("Client %d:\n", pid);
61
62     while (strcmp(buf, "break\n"))
63     {
64         memset(buf, 0, BUF_SIZE);
65         printf("Message: ");
66         fgets(buf, BUF_SIZE-1, stdin);
67         write(sock, buf, strlen(buf));
68
69         memset(buf_answer, 0, BUF_SIZE);
70         read(sock, buf_answer, BUF_SIZE-1);
71         printf("Answer: %s\n", buf_answer);
72     }
73
74     close(sock);
75     return 0;
76 }

```

В процессе-клиенте создаётся сокет семейства AF_INET с типом SOCK_STREAM. С помощью функции gethostbyname() доменный адрес преобразуется в сетевой, после чего устанавливается соединение с помощью connect(). Затем используются read()/write() для чтения-записи из сокета. Клиент продолжает работу до тех пор, пока не будет послано сообщение break, после чего сокет закрывается.

Листинг 1.4 — Текст программы сервера

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <string.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>

```



```

8 #include <unistd.h>
9
10 #define BUF_SIZE 256
11 #define MAX_CONNECT 1024
12 #define ANSWER "OK"
13 #define SIZE_ANSWER 2
14
15 typedef struct client client_t;
16 struct client {
17     int fd;
18     int pid;
19 };
20
21 int new_connection(int sock, client_t clients[FD_SETSIZE], fd_set *all_set,
22     fd_set *reset, int *max_fd, int *max_idx);
23 int clients_handler(client_t clients[FD_SETSIZE], fd_set *all_set,
24     fd_set *reset, int *max_idx);
25
26 int main(int argc, char ** argv)
27 {
28     int sock, newsock, port;
29     client_t clients[FD_SETSIZE];
30     int max_fd, max_idx;
31     struct sockaddr_in serv_addr, cli_addr;
32     if (argc < 2)
33     {
34         fprintf(stderr, "usage: %s <port_number>\n", argv[0]);
35         return EXIT_FAILURE;
36     }
37
38     sock = socket(AF_INET, SOCK_STREAM, 0);
39     if (socket < 0)
40     {
41         printf("socket() failed: %d\n", errno);
42         return EXIT_FAILURE;
43     }
44
45     memset((char *) &serv_addr, 0, sizeof(serv_addr));
46     serv_addr.sin_family = AF_INET;
47     serv_addr.sin_addr.s_addr = INADDR_ANY;
48     port = atoi(argv[1]);
49     serv_addr.sin_port = htons(port);
50
51     if (bind(sock, (struct sockaddr *) &serv_addr,
52         sizeof(serv_addr)) < 0)
53     {

```

```

54     printf("bind() failed: %d\n", errno);
55     return EXIT_FAILURE;
56 }
57
58     printf("Server is running.\n");
59
60     listen(sock, MAX_CONNECT);
61
62     max_fd = sock;
63     max_idx = -1;
64
65     fd_set reset, all_set;
66     FD_ZERO(&all_set);
67     FD_SET(sock, &all_set);
68
69     for (int i = 0; i < FD_SETSIZE; i++)
70     {
71         clients[i].fd = -1;
72     }
73
74     while (1)
75     {
76         reset = all_set;
77
78         select(max_fd + 1, &reset, NULL, NULL, NULL);
79
80         if (new_connection(sock, clients, &all_set, &reset, &max_fd,
81                             &max_idx) != 0)
82         {
83             return EXIT_FAILURE;
84         }
85         if (clients_handler(clients, &all_set, &reset, &max_idx) != 0)
86         {
87             return EXIT_FAILURE;
88         }
89     }
90
91     close(sock);
92 }
93
94 int new_connection(int sock, client_t clients[FD_SETSIZE], fd_set *all_set,
95                   fd_set *reset, int *max_fd, int *max_idx)
96 {
97     if (FD_ISSET(sock, reset))
98     {
99         int conn_fd = accept(sock, NULL, NULL);

```

```

100     if (conn_fd < 0)
101     {
102         printf("accept() failed: %d\n", errno);
103         return EXIT_FAILURE;
104     }
105
106     int idx;
107     for (idx = 0; idx < FD_SETSIZE; idx++)
108     {
109         if (clients[idx].fd < 0)
110         {
111             clients[idx].fd = conn_fd;
112             break;
113         }
114     }
115     if (idx == FD_SETSIZE)
116     {
117         perror("Maximum number of clients reached\n");
118         return EXIT_FAILURE;
119     }
120     if (idx > *max_idx)
121     {
122         *max_idx = idx;
123     }
124     if (conn_fd > *max_fd)
125     {
126         *max_fd = conn_fd;
127     }
128
129     char pid[BUF_SIZE];
130     read(conn_fd, pid, BUF_SIZE);
131     clients[idx].pid = atoi(pid);
132
133     FD_SET(conn_fd, all_set);
134     printf("New client: %d\n", clients[idx].pid);
135 }
136
137 return 0;
138 }
139
140 int clients_handler(client_t clients[FD_SETSIZE], fd_set *all_set,
141                     fd_set *reset, int *max_idx)
142 {
143     char buf[BUF_SIZE];
144     int msg_size = 0;
145     for (int i = 0; i <= *max_idx; i++)

```

```

146     {
147         if (clients[i].fd == -1)
148         {
149             continue;
150         }
151
152         if (FD_ISSET(clients[i].fd, reset))
153         {
154             msg_size = read(clients[i].fd, buf, BUF_SIZE);
155             if (msg_size == 0)
156             {
157                 close(clients[i].fd);
158                 FD_CLR(clients[i].fd, all_set);
159                 clients[i].fd = -1;
160                 printf("Client %d disconnected\n", clients[i].pid);
161             }
162             else
163             {
164                 write(clients[i].fd, ANSWER, SIZE_ANSWER);
165                 buf[msg_size] = '\0';
166                 printf("Message from %d client: %s", clients[i].pid, buf);
167             }
168         }
169     }
170
171     return 0;
172 }

```

В процессе-сервере также создаётся сокет семейства AF_INET с типом SOCK_STREAM. Затем происходит связывание сокета с заданным адресом. Далее ожидаются запросы на соединение с помощью listen(). После чего инициализируются вспомогательные значения для работы сервера и устанавливается новое значение в набор дескрипторов с помощью функции FD_SET.

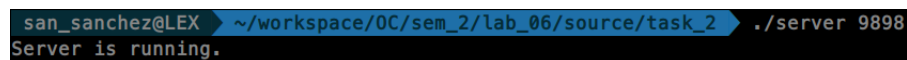
Затем сервер блокируется на функции select() до тех пор, пока не будет установлено новое клиентское соединение или на существующее не придут новые данные. После выхода из блокировки происходит обработка соответствующих событий.

В функции new_connection() происходит соединение в ответ на запрос клиента с помощью функции accept(), которая возвращает новый сокет для

связи с клиентом. Также мы добавляем новый дескриптор файла (сокеты) в набор дескрипторов.

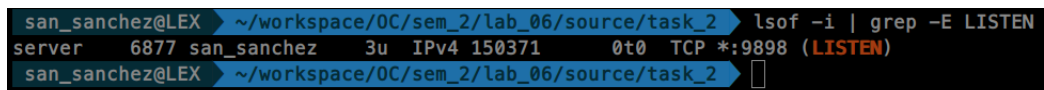
В функции `clients_handler()` происходит проверка клиентов на наличие данных, после чего происходит чтение сообщения и ответ клиенту, либо закрытие сокета `close()` и удаление из набора дескрипторов `FD_CLR()`, если клиент отключился.

В примерах работы приложения используется 9898 порт:



```
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_2$ ./server 9898
Server is running.
```

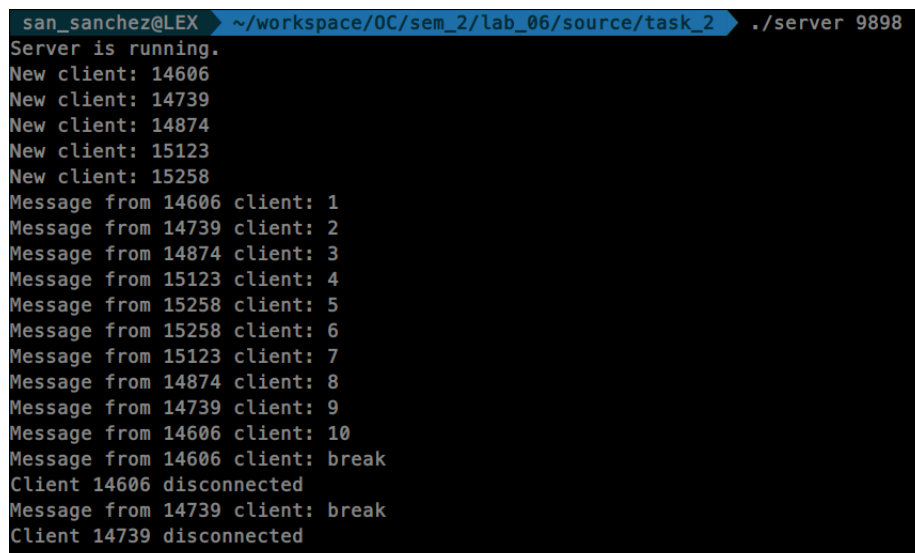
Рисунок 1.4 — Скришот запущенного сервера.



```
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_2$ lsof -i | grep -E LISTEN
server  6877 san_sanchez  3u  IPv4 150371      0t0  TCP *:9898 (LISTEN)
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_2$
```

Рисунок 1.5 — Скришот работы команды `lsof -i | grep -E LISTEN`.

Примеры работы второго задания:



```
san_sanchez@LEX ~/workspace/0C/sem_2/lab_06/source/task_2$ ./server 9898
Server is running.
New client: 14606
New client: 14739
New client: 14874
New client: 15123
New client: 15258
Message from 14606 client: 1
Message from 14739 client: 2
Message from 14874 client: 3
Message from 15123 client: 4
Message from 15258 client: 5
Message from 15258 client: 6
Message from 15123 client: 7
Message from 14874 client: 8
Message from 14739 client: 9
Message from 14606 client: 10
Message from 14606 client: break
Client 14606 disconnected
Message from 14739 client: break
Client 14739 disconnected
```

Рисунок 1.6 — Скришот работы сервера.

```
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14606:
Message: 1
Answer: OK
Message: 10
Answer: OK
Message: break
Answer: OK
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14739:
Message: 2
Answer: OK
Message: 9
Answer: OK
Message: break
Answer: OK
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14874:
Message: 3
Answer: OK
Message: 8
Answer: OK
Message: 
Message: 4
Answer: OK
Message: 7
Answer: OK
Message:

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 15258:
Message: 5
Answer: OK
Message: 6
Answer: OK
Message:
```

Рисунок 1.7 — Скришот работы клиентов.

```
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ls
client client.c server server.c
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./server
usage: ./server <port_number>
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./server 9898
Server is running.
New client: 8249
Message from 8249 client: asdws
Message from 8249 client: dasdwscc
New client: 9848
Message from 9848 client: 123
New client: 10716
Message from 10716 client: Hello
New client: 10922
Message from 10922 client: world
New client: 11286
Message from 11286 client: !
Message from 11286 client: break
Client 11286 disconnected
Message from 9848 client: break
Client 9848 disconnected
Message from 8249 client: Bye
Message from 8249 client: break
Client 8249 disconnected
```

Рисунок 1.8 — Скришот работы сервера.

```
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14606:
Message: 1
Answer: OK
Message: 10
Answer: OK
Message: break
Answer: OK
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14739:
Message: 2
Answer: OK
Message: 9
Answer: OK
Message: break
Answer: OK
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 14874:
Message: 3
Answer: OK
Message: 8
Answer: OK
Message: 
Message: 4
Answer: OK
Message: 7
Answer: OK
Message:

san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 >
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_06/source/task_2 > ./client localhost 9898
Client 15258:
Message: 5
Answer: OK
Message: 6
Answer: OK
Message:
```

Рисунок 1.9 — Скришот работы клиентов.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе была проанализированна работа сетевых сокетов и сокетов в пространстве файловых имён.