



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 Сиденко А.Г.
Группа ИУ7-63Б
Оценка (баллы)
Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

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: Код сервера

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <errno.h>
5 #include <unistd.h>
6 #include <sys/socket.h>
7 #include <sys/un.h>
8
9 #define SOCK_NAME "socket.soc"
10 #define MSG_SIZE 256
11
12 int main()
13 {
14     // Для сокетов Unix (сокетов в файловом пространстве имен)
15     // есть специализированная структура sockaddr_un
16     struct sockaddr_un server;
17     char msg[MSG_SIZE];
18     int bytes;
19     char pid[10];
20
21     // Создание сокета в файловом пространстве имен (домен AF_UNIX)
22     // Тип сокета — SOCK_DGRAM означает датаграммный сокет
23     // Протокол — 0, протокол выбирается по умолчанию
24     int sock = socket(AF_UNIX, SOCK_DGRAM, 0);
25     if (sock < 0)
26     {
27         printf("%s", strerror(errno));
28         return errno;
29     }
30
31     // Укажем семейство адресов, которыми мы будем пользоваться
32     server.sun_family = AF_UNIX;
33     // Укажем имя файла сокета
34     strcpy(server.sun_path, SOCK_NAME);
35
36     // Связывание сокета с заданным адресом
37     // bind(дескриптор сокета, указатель на структуру, длина структуры)
38     if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
39     {
40         printf("%s", strerror(errno));
41         return errno;
42     }
```

```

43
44 // Наша программа-сервер становится доступна для соединения
45 // по заданному адресу (имени файла)
46
47 // Пока клиент не отправит сообщение "break"
48 while (strcmp(msg, "break"))
49 {
50     // Для чтения данных из датаграммного сокета — recvfrom,
51     // которая блокирует программу до тех пор, пока на входе
52     // не появятся новые данные
53     // Так как нас не интересуют данные об адресе клиента
54     // передаем значения NULL в предпоследнем и последнем параметрах
55     recvfrom(sock, pid, 10, 0, NULL, NULL);
56     bytes = recvfrom(sock, msg, MSG_SIZE, 0, NULL, NULL);
57     if (bytes < 0)
58     {
59         printf("%s", strerror(errno));
60         return errno;
61     }
62     // Символ окончания строки
63     msg[bytes] = 0;
64     printf("Сообщение_от_клиента_%d:_%s\n", atoi(pid), msg);
65 }
66
67 // Закрываем сокет
68 close(sock);
69 // Удаляем файл сокета
70 unlink(SOCK_NAME);
71
72 return errno;
73 }

```

Листинг 2: Код клиента

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <errno.h>
5 #include <sys/socket.h>
6 #include <sys/un.h>
7 #include <unistd.h>
8
9 #define SOCK_NAME "socket.soc"
10 #define MSG_SIZE 256
11
12 int main(int argc, char ** argv)
13 {
14     // Создание сокета в файловом пространстве имен (домен AF_UNIX)
15     // Тип сокета — SOCK_DGRAM означает датаграммный сокет
16     // Протокол — 0, протокол выбирается по умолчанию
17     char msg[MSG_SIZE];
18     struct sockaddr_un server;

```

```

19 char id[10];
20 sprintf(id, "%d", getpid());
21 id[strlen(id)] = 0;
22
23 int sock = socket(AF_UNIX, SOCK_DGRAM, 0);
24 if (sock < 0)
25 {
26     printf("%s", strerror(errno));
27     return errno;
28 }
29
30 // Укажем семейство адресов, которыми мы будем пользоваться
31 server.sun_family = AF_UNIX;
32 // Укажем имя файла сокета
33 strcpy(server.sun_path, SOCK_NAME);
34
35 // Приглашение и ввод сообщения для сервера
36 printf("Введите сообщение:\n");
37 scanf("%s", msg);
38
39 // Передаем сообщение серверу
40 // sendto(дескриптор сокета, адрес буфера для передачи данных,
41 // его длина, дополнительные флаги, адрес сервера, его длине)
42 sendto(sock, id, strlen(id), 0, (struct sockaddr *)&server,
43                                               sizeof(server));
44 sendto(sock, msg, strlen(msg), 0, (struct sockaddr *)&server,
45                                               sizeof(server));
46
47 return errno;
48 }

```

Запускаем приложение сервера и принимаем сообщения от клиентов:

```

+ semestr2/lab6/unix ➤ semestr2_lab6 ➤ ./fsserver.out
Сообщение от клиента 78603: Hi!
Сообщение от клиента 78696: Hello
Сообщение от клиента 78789: Wow
Сообщение от клиента 78922: good
Сообщение от клиента 79029: Buy
Сообщение от клиента 79109: break

```

Запускаем приложения клиентов, отправляем сообщения, при отправке сообщения break сервер заканчивает работу:

```
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
Hi!
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
Hello
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
Wow
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
good
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
Buy
+ semestr2/lab6/unix semestr2_lab6± ./fsclient.out
Введите сообщение:
break
```

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 3: Код сервера

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <strings.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>
8  #include <unistd.h>
9
10 #define MSG_SIZE 256
11 #define LISTENQ 1024
12
13 int maxi, maxfd;
14
15 // Соединение с новым клиентом
16 int newClient(int listensock, int client[FD_SETSIZE], int id[FD_SETSIZE], fd_
17 {
18     // Индекс
19     int i;
20     int connfd;
21     char pid[MSG_SIZE];
22
23     if (FD_ISSET(listensock, reset))
24     {
25         // Установка соединения в ответ на запрос клиента
26         // Так как нас не интересуют данные об адресе клиента
27         // передаем значения NULL в предпоследнем и последнем параметрах
28         int connfd = accept(listensock, NULL, NULL);
29         if (connfd < 0)
30         {
31             printf("%s", strerror(errno));
32             return errno;
33         }
34         // Функция accept() возвращает новый сокет, открытый
35         // для обмена данными с клиентом, запросившим соединение
36
37         // Сохраняем дескриптор в первый свободный
38         for (i = 0; i < FD_SETSIZE; i++)
39         {
40             if (client[i] < 0)
41             {
42                 client[i] = connfd;

```

```

43         break;
44     }
45 }
46
47 if (i == FD_SETSIZE)
48 {
49     printf("Достигнуто_максимальное_число_клиентов");
50     return errno;
51 }
52
53 // Добавление нового дескриптора
54 FD_SET(connfd, allset);
55
56 // Максимальный для функции select
57 if (connfd > maxfd)
58     maxfd = connfd;
59
60 // Максимальный индекс в массиве клиентов
61 if (i > maxi)
62     maxi = i;
63
64 read(connfd, pid, MSG_SIZE);
65 id[i] = atoi(pid);
66 printf("Клиент_%d_подключился\n", id[i]);
67 }
68 return errno;
69 }
70
71 int readMsg(int client[FD_SETSIZE], int id[FD_SETSIZE], fd_set *allset,
72             fd_set *reset)
73 {
74     int n, i;
75     int sockfd;
76     char msg[MSG_SIZE];
77     // Проверяем все клиенты на наличие данных, пока не дошли до конца
78     // или не закончились дескрипторы готовые для чтения
79     for (i = 0; i <= maxi; i++)
80     {
81         // Если не пустой
82         if ((sockfd = client[i]) > 0)
83         {
84             // Установлен ли бит?
85             if (FD_ISSET(sockfd, reset))
86             {
87                 // Соединение закрыто клиентом
88                 if ((n = read(sockfd, msg, MSG_SIZE)) == 0)
89                 {
90                     // Закрываем сокет
91                     close(sockfd);
92                     // Сброс бита
93                     FD_CLR(sockfd, allset);
94                     // Освобождаем ячейку в массиве клиентов

```

```

95         client[i] = -1;
96         printf("Клиент_%d_отключился\n", id[i]);
97     }
98     else
99     {
100         // Сообщение клиенту о доставке сообщения
101         write(sockfd, "OK", 2);
102         // Установка символа конца строки и вывод сообщения на экран
103         msg[n] = 0;
104         printf("Сообщение_от_клиента_%d:_%s", id[i], msg);
105     }
106 }
107 }
108 }
109 return errno;
110 }
111
112 int main(int argc, char ** argv)
113 {
114     int listensock;
115     int client[FD_SETSIZE];
116     int id[FD_SETSIZE];
117     fd_set reset, allset;
118     // Структура предназначен для хранения адресов в формате Интернета
119     struct sockaddr_in server;
120
121     if (argc < 2)
122     {
123         fprintf(stderr, "Использование:_%s_<port_number>\n", argv[0]);
124         return 1;
125     }
126
127     // Создание сетевого сокета (домен AF_INET)
128     // Тип сокета — SOCK_STREAM, сокет должен быть потоковым
129     // Протокол — 0, протокол выбирается по умолчанию
130     listensock = socket(AF_INET, SOCK_STREAM, 0);
131     if (listensock < 0)
132     {
133         printf("%s", strerror(errno));
134         return errno;
135     }
136
137     // Укажем семейство адресов, которыми мы будем пользоваться
138     server.sin_family = AF_INET;
139     // Укажем адрес (наша программа-сервер регистрируется на всех адресах
140     // машины, на которой она выполняется)
141     server.sin_addr.s_addr = INADDR_ANY;
142     // Укажем значение порта. Функция htons() переписывает
143     // двухбайтовое значение порта так, чтобы порядок байтов
144     // соответствовал принятому в Интернете
145     server.sin_port = htons(atoi(argv[1]));
146

```



```

147 // Связывание сокета с заданным адресом
148 // bind(дескриптор сокета, указатель на структуру, длина структуры)
149 if (bind(listensock, (struct sockaddr *)&server, sizeof(server)) < 0)
150 {
151     printf("%s", strerror(errno));
152     return errno;
153 }
154
155 // Переводим сервер в режим ожидания запроса на соединение
156 // Второй параметр – максимальное число
157 // обрабатываемых одновременно соединений
158 listen(listensock, LISTENQ);
159
160 // Инициализация значения
161 maxfd = listensock;
162 // Индекс в массиве клиентов (наибольший используемый)
163 maxi = -1;
164 // Массив дескрипторов присоединенного сокета для каждого клиента
165 for (int i = 0; i < FD_SETSIZE; i++)
166     client[i] = -1; // -1 означает, что элемент свободен
167
168 // Сбрасываем все биты в allset
169 FD_ZERO(&allset);
170 // Устанавливаем бит для listensock в allset
171 FD_SET(listensock, &allset);
172
173 while(1)
174 {
175     // Присваивание значения структуре
176     reset = allset;
177     // select() ждет пока не будет установлено новое клиентское
178     // соединение или на существующем не придут данные
179     // select(количество проверяемых дескрипторов, 2–4 наборы
180     // дескрипторов, которые следует проверять, интервал времени,
181     // по прошествии которого она вернет управление в любом случае)
182     select(maxfd + 1, &reset, NULL, NULL, NULL);
183
184     if (newClient(listensock, client, id, &allset, &reset) ||
185         readMsg(client, id, &allset, &reset))
186         return errno;
187 }
188
189 // Закрываем сокет
190 close(listensock);
191
192 return errno;
193 }

```

Листинг 4: Код клиента

```

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

```

```

3 #include <errno.h>
4 #include <strings.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 MSG_SIZE 256
12
13 int main(int argc, char ** argv)
14 {
15     struct sockaddr_in server;
16     struct hostent *host;
17     char msg_client[MSG_SIZE], msg_server[MSG_SIZE];
18     char id[10];
19     sprintf(id, "%d", getpid());
20     id[strlen(id)] = 0;
21
22     if (argc < 3)
23     {
24         fprintf(stderr, "Использование:_%s_<hostname>_<port_number>\n",
25                                     argv[0]);
26         return 1;
27     }
28
29     // Создание сетевого сокета (домен AF_INET)
30     // Тип сокета — SOCK_STREAM, сокет должен быть потоковым
31     // Протокол — 0, протокол выбирается по умолчанию
32     int sock = socket(AF_INET, SOCK_STREAM, 0);
33     if (sock < 0)
34     {
35         printf("%s", strerror(errno));
36         return errno;
37     }
38
39     // Преобразование доменного имени сервера в его сетевой адрес
40     host = gethostbyname(argv[1]);
41     if (host == NULL)
42     {
43         printf("%s", strerror(errno));
44         return errno;
45     }
46
47     // Укажем семейство адресов, которыми мы будем пользоваться
48     server.sin_family = AF_INET;
49     // Укажем адрес (наша программа-сервер регистрируется на всех адресах
50     // машины, на которой она выполняется)
51     memcpy(&server.sin_addr, host->h_addr_list[0], host->h_length);
52     // Укажем значение порта. Функция htons() переписывает
53     // двухбайтовое значение порта так, чтобы порядок байтов
54     // соответствовал принятому в Интернете

```

```

55 server.sin_port = htons(atoi(argv[2]));
56
57 // Установка соединения
58 if (connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
59 {
60     printf("%s", strerror(errno));
61     return errno;
62 }
63 write(sock, id, strlen(id));
64
65 // Пока не сообщение "break"
66 while (strcmp(msg_client, "break\n"))
67 {
68     memset(msg_client, 0, MSG_SIZE);
69     printf("Введите сообщение:\n");
70     fgets(msg_client, MSG_SIZE, stdin);
71     // Для записи данных — write
72     write(sock, msg_client, strlen(msg_client));
73
74     // Заполнение массива нулями
75     memset(msg_server, 0, MSG_SIZE);
76     // Для чтения данных из сокета — read
77     read(sock, msg_server, MSG_SIZE);
78     printf("%s\n", msg_server);
79 }
80 // Закрываем сокет
81 close(sock);
82
83 return 0;
84 }

```

Запускаем приложение сервера и принимаем сообщения от клиентов:

```
x + semestr2/lab6/server semestr2_lab6± ./netserver.out 9877
Клиент 58632 подключился
Клиент 58733 подключился
Сообщение от клиента 58733: Hi!
Сообщение от клиента 58632: Hello!
Клиент 59234 подключился
Сообщение от клиента 59234: Good evening!
Сообщение от клиента 58733: Hello, new client!
Сообщение от клиента 58632: break
Клиент 58632 отключился
Клиент 60176 подключился
Сообщение от клиента 60176: Nice to meet you!
Сообщение от клиента 59234: Fantastic!
Сообщение от клиента 58733: Wow!
Сообщение от клиента 58733: Buy
Сообщение от клиента 58733: break
Клиент 58733 отключился
Сообщение от клиента 59234: Goodbye!
Сообщение от клиента 60176: I am here!
Сообщение от клиента 59234: I see
Клиент 62310 подключился
Сообщение от клиента 62310: Go friends!
Сообщение от клиента 59234: Go!
Сообщение от клиента 59234: break
Клиент 59234 отключился
Сообщение от клиента 62310: break
Клиент 62310 отключился
Сообщение от клиента 60176: break
Клиент 60176 отключился
```

Запускаем приложения клиентов, при каждом новом подключении сервер выводит сообщение об этом. Далее несколько клиентов отправляют сообщения, могут завершиться, затем вновь возобновить работу.

При этом новому клиенту присваивается первый свободный номер.

```
x + semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Hello!
OK
Введите сообщение:
break
OK
+ semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Nice to meet you!
OK
Введите сообщение:
I am here!
OK
Введите сообщение:
break
OK
x + semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Hi!
OK
Введите сообщение:
Hello, new client!
OK
Введите сообщение:
Wow!
OK
Введите сообщение:
Buy
OK
Введите сообщение:
break
OK
+ semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Go friends!
OK
Введите сообщение:
break
OK
```

```

+ semestr2/lab6/server semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Good evening!
OK
Введите сообщение:
Fantastic!
OK
Введите сообщение:
Goodbye!
OK
Введите сообщение:
I see
OK
Введите сообщение:
Go!
OK
Введите сообщение:
break
OK

```

Максимально одновременно можно запустить $FD_SETSIZE = 256$ клиентов. Запустим 5 клиентов одновременно проверим работу.

Сервер:

```

x + semestr2/lab6/server semestr2_lab6± ./netserver.out 9878
Клиент 68046 подключился
Клиент 68187 подключился
Клиент 68319 подключился
Клиент 68549 подключился
Клиент 68690 подключился
Сообщение от клиента 68046: Hi!
Сообщение от клиента 68319: Hello!
Сообщение от клиента 68690: Nice to meet you!
Сообщение от клиента 68187: Good evening!
Сообщение от клиента 68549: Wow!
Сообщение от клиента 68046: break
Клиент 68046 отключился
Сообщение от клиента 68187: break
Клиент 68187 отключился
Сообщение от клиента 68549: break
Клиент 68549 отключился
Сообщение от клиента 68690: break
Клиент 68690 отключился
Сообщение от клиента 68319: break
Клиент 68319 отключился

```

Клиенты:

```
x + semestr2/lab6/server | semestr2_lab6± ./netclient.out localhost 9878
Введите сообщение:
Hi!
OK
Введите сообщение:
break
OK

x + semestr2/lab6/server | semestr2_lab6± ./netclient.out localhost 9878
Введите сообщение:
Good evening!
OK
Введите сообщение:
break
OK

x + semestr2/lab6/server | semestr2_lab6± ./netclient.out localhost 9878
Введите сообщение:
Hello!
OK
Введите сообщение:
break
OK

+ semestr2/lab6/server | semestr2_lab6± ./netclient.out localhost 9878
Введите сообщение:
Nice to meet you!
OK
Введите сообщение:
break
OK

+ semestr2/lab6/server | semestr2_lab6± ./netclient.out localhost 9878
Введите сообщение:
Wow!
OK
Введите сообщение:
break
OK
```