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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №6
По курсу «Операционные системы»
Тема: Сокеты

Студент: Мирзоян С.А.
Группа: ИУ7-65Б.
Преподаватель: Рязанова Н.Ю.

Москва, 2020г

Сокет – это абстракция конечной точки взаимодействия. Абстракция сокетов была введена в 4.2 BSD (Berkley Software Distribution) UNIX и были созданы для организации взаимодействия процессов, причем безразлично, где эти процессы выполняются: на одной машине или на нескольких машинах.

Сокеты поддерживают множество протоколов, поэтому была определена общая структура адреса sockaddr, так как при создании коммуникационных отношений нужно указывать адрес конечной точки коммуникационного партнера.

Структура sockaddr определяется следующим образом:

typedef unsigned short sa_family_t

```
        /*
         *      1003.1g requires sa_family_t and that sa_data is char.
         */

        struct sockaddr {
                sa_family_t  sa_family;    /* address family, AF_xxx       */
                char         sa_data[14]; /* 14 bytes of protocol address  */
        };
}
```

Когда сокет создается с помощью socket (2), он существует в пространстве имен (семейство адресов), но ему не назначен адрес. Специальная функция bind () назначает адрес, указанный addr, сокету, указанному дескриптором файла sockfd. Параметр addrlen определяет размер в байтах структуры адреса, на которую указывает addr. Традиционно эта операция называется «присвоение имени сокету».

Параметр domain указывает домен связи. Домен определяет семейство протоколов, которое будет использоваться для связи. Эти семейства определены в <sys / socket.h>. Ядро Linux в настоящее время включает следующие форматы:

Структура sockaddr_in определяется следующим образом:

```
struct sockaddr_in
{
        short int sin_family; // Семейство адресов
        unsigned short int sin_port; // Номер порта
        struct in_addr sin_addr; // IP-адрес
        unsigned char sin_zero[8]; // Дополнение до размера структуры sockaddr
  };
```

Название	Назначение	Справочная страница
AF_UNIX, **AF_LOCAL**	Локальное соединение	**unix**(7)
AF_INET	Протоколы Интернет IPv4	**ip**(7)
AF_INET6	Протоколы Интернет IPv6	**ipv6**(7)
AF_IPX	Протоколы Novell IPX	
AF_NETLINK	Устройство для взаимодействия с ядром	**netlink**(7)
AF_X25	Протокол ITU-T X.25/ISO-8208	**x25**(7)
AF_AX25	Протокол любительского радио AX.25	
AF_ATMPVC	Доступ к низкоуровневым PVC в ATM	
AF_APPLETALK	AppleTalk	**ddp**(7)
AF_PACKET	Низкоуровневый пакетный интерфейс	**packet**(7)
AF_ALG	Интерфейс к ядерному крипто-API	

Второй параметр функции socket() - тип сокета (type). В настоящее время определены следующие типы:

- SOCK_STREAM Обеспечивает создание двусторонних, надёжных потоков байтов на основе установления соединения. Может также поддерживаться механизм внепоточных данных.
- SOCK_DGRAM Поддерживает дейтаграммы (ненадежные сообщения с ограниченной длиной без установки соединения).
- SOCK_SEQPACKET Обеспечивает работу последовательного двустороннего канала для передачи дейтаграмм на основе соединений; дейтаграммы имеют постоянный размер; от получателя требуется за один раз прочитать целый пакет.
- SOCK_RAW Обеспечивает прямой доступ к сетевому протоколу.
- SOCK_RDM Обеспечивает надежную доставку дейтаграмм без гарантии, что они будут расположены по порядку.
- SOCK_PACKET Устарел и не должен использоваться в новых программах; см. packet.

Часть 1

Организовать взаимодействие параллельных процессов на отдельном компьютере.

Задание: 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

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

#define SOCK NAME "my socket"
    #define SIZE 256

    void reciever(int sock);
    void sig_handler(int sig);

    int mysocket;

    int main(void)
    {
        //int mysocket;
        struct sockaddr addr;

        mysocket = socket(AF_LOCAL, SOCK_DGRAM, 0); //socket - системный вызов, возвращает дескриптор сокета
                            //AF_LOCAL - для связи процессов в одной машине
                            //SOCK_DGRAM - датаграммные сокеты (без установления соединения, ненадежная передача сообщений; сообщения фиксированной максимальной длины).
        if (mysocket < 0)
        {
```

```
27.             perror("Сообщение сервера: Невозможно открыть сокет!\n");
28.             exit(1);
29.         }
30.
31.         addr.sa_family = AF_LOCAL; //sa_family - Семейство адресов, AF_xxx
32.         strcpy(addr.sa_data, SOCK_NAME) ;//sa_data - 14 байтов для хранения адреса
33.
34.         if (bind(mysocket, &addr, sizeof(struct sockaddr)) < 0) //bind(2) привязывает
    сокет к локальному адресу,
35.         {
36.             printf("Закрыть сокет\n");
37.             perror("Сообщение сервера: невозможно привязать имя к сокету!\n");
38.             close(mysocket) ;
39.             unlink(SOCK_NAME) ;//unlink - системная функция, удаляет созданный файл
40.             exit(2);
41.         }
42.
43.         printf ("Сокет: %s\n", SOCK_NAME) ;
44.         signal(SIGINT, sig_handler); // cntl+c
45.
46.         while (1)
47.             reciever(mysocket); //прием сообщений
48.
49.         printf("Закрыть сокет\n");
50.         close(mysocket) ;
51.         unlink(SOCK_NAME) ;
52.
53.         return 0;
54.     }
55.
56.     void reciever(int sock)
57.     {
58.         char buffer[SIZE];
59.         int size = recv(sock, buffer, sizeof(buffer), 0); //recv - прием данных из
    сокета
60.         if (size < 0)
61.         {
62.             perror("Сообщение сервера: невозможно получить данные!\n");
63.             close(sock) ;
64.             unlink(SOCK_NAME) ;
65.             exit(3);
```

```
66.         }
67.         buffer[size] = 0;
68.         printf("%s\n", buffer);
69.     }
70.
71.     void sig_handler(int sig) //удаление файла
72.     {
73.         puts("Завершение приема по сигналу (ctrl + c)\n");
74.         close(mysocket) ;
75.         unlink(SOCK_NAME) ;
76.         exit(1);
77.     }
```

Листинг 2: Код клиента

```
1.#include <sys/types.h>
2.#include <sys/socket.h>
3.#include <sys/un.h>
4.#include <stdio.h>
5.#include <stdlib.h>
6.#include <unistd.h>
7.
8.#define SOCK NAME "my socket"
9.#define SIZE 256
10.
11.     int main(int argc, char *argv[])
12.     {
13.         int sock;
14.
15.         struct sockaddr addr;
16.
17.         sock = socket(PF_LOCAL, SOCK_DGRAM, 0);
18.         if (sock < 0)
19.         {
20.             perror("Сообщение клиента: невозможно открыть сокет!\n");
21.             exit(1);
22.         }
23.
24.         addr.sa_family = AF_LOCAL;
25.         strcpy(addr.sa_data, SOCK_NAME);
26.
27.         int pid = getpid();
```

```
28.         char mypid[6];   // ex. 34567
29.         sprintf(mypid, "%d", pid);
30.
31.         strcat(argv[1], " ");
32.         strcat(argv[1], mypid);
33.
34.         sendto(sock, argv[1], sizeof(argv[1]), 0, &addr, sizeof(addr));  //sendto
   -  отправляет данные в сокет,
35.
36.         close(sock) ;
37.         return 0;
38.     }
```

Результат работы:

```
srwxr-xr-x 1 sergey sergey         0 мая  1 05:09  my_socket
```
- созданный сокет в ФС (ls –l).

```
sergey@sergey-VirtualBox:~/Рабочий стол$ ./s
Сокет: my_socket
1 9580
2 9606
3 9622
4 9639
5 9653
^CЗавершение приема по сигналу (ctrl + c)
```

```
sergey@sergey-VirtualBox:~/Рабочий стол$ ./c 1
sergey@sergey-VirtualBox: ~/Рабочий стол
sergey@sergey-VirtualBox:~/Рабочий стол$ ./c 2
sergey@sergey-VirtualBox: ~/Рабочий стол
sergey@sergey-VirtualBox:~/Рабочий стол$ ./c 3
sergey@sergey-VirtualBox: ~/Рабочий стол
sergey@sergey-VirtualBox:~/Рабочий стол$ ./c 4
sergey@sergey-VirtualBox: ~/Рабочий стол
sergey@sergey-VirtualBox:~/Рабочий стол$ ./c 5
```

После завершения my_socket удаляется.

Часть 2

Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине).

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.

TCP-порт **31337** использует протокол управления передачей данных (TCP), который является одним из основных протоколов в сетях TCP/IP. TCP является протоколом с установлением соединения и требует квитирования для установки сквозной связи. Только после установления соединения пользовательские данные могут пересылаться в обоих направлениях.
TCP гарантирует доставку пакетов данных через порт **31337** в том же порядке, в котором они были отправлены. Гарантированная связь через TCP-порт **31337** является основным отличием TCP от UDP.

Листинг 3: Код сервера

```
1.#include <arpa/inet.h>
2.#include <sys/types.h>
3.#include <sys/socket.h>
4.#include <sys/time.h>
5.#include <netinet/in.h>
6.#include <unistd.h>
7.#include <string.h>
8.#include <stdlib.h>
9.#include <stdio.h>
10.     #include <fcntl.h>
11.
12.     #define SOCKET PORT 31337
13.     #define BUFFER SIZE 256
14.     #define LISTEN QUEUE SIZE 256
15.
16.     #define CLIENTS 10
17.
18.
19.     void recieve_data(int clients[], int i);
```

```
20.     int max(int listener, int arr[], int count);
21.     int insert(int s, int clients[], int count);
22.
23.     int main(void)
24.     {
25.         int sockL;
26.         struct sockaddr_in addr;
27.         int clients[CLIENTS] = {0};
28.         fd_set rset;
29.
30.         sockL = socket(PF_INET, SOCK_STREAM, 0);
31.         if (sockL < 0)
32.         {
33.             perror("Сообщение сервера: невозможно открыть сокет!\n");
34.             return sockL;
35.         }
36.         addr.sin_family = PF_INET;
37.         addr.sin_port = htons(SOCKET_PORT) ;
38.         addr.sin_addr.s_addr = INADDR_ANY;
39.         if (bind(sockL, (struct sockaddr *)&addr, sizeof(addr)) < 0)
40.         {
41.             perror("Сообщение сервера: невозможно привязать адрес!\n");
42.             exit(2);
43.         }
44.
45.         printf("Порт %d\n", SOCKET_PORT) ;
46.
47.         listen(sockL, LISTEN_QUEUE_SIZE);
48.
49.         while(1)
50.         {
51.             FD_ZERO(&rset) ;
52.             FD_SET(sockL, &rset);
53.                 for (int i = 0; i < CLIENTS; i++)
54.                     if (clients[i] != 0)
55.                         FD_SET(clients[i], &rset);
56.
57.             int mx = max(sockL, clients, CLIENTS);
58.
59.             if (select(mx+1, &rset, NULL, NULL, NULL) <= 0)
60.             {
```

```
61.                 perror("Выбор невозможен\n");
62.                 exit(3);
63.             }
64.
65.             if (FD_ISSET(sockL, &rset))
66.             {
67.                 struct sockaddr_in client_addr;
68.                 int addrSize = sizeof(client_addr);
69.
70.                 int sock = accept(sockL, (struct sockaddr*) &client_addr, (socklen_t*)
    &addrSize) ;
71.                 if (sock < 0)
72.                 {
73.                     perror("Сообщение сервера: прием невозможен \n");
74.                     exit(4);
75.                 }
76.
77.                 printf("\nНовое соединение: \nfd = %d \nip = %s:%d\n",
    sock,inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
78.                 if (insert(sock, clients, CLIENTS) < 0)
79.                 {
80.                     perror("Сообщение сервера: слишком много клиентов!\n");
81.                     exit(5);
82.                 }
83.             }
84.             for (int i = 0; i < CLIENTS; i++)
85.                 if (FD_ISSET(clients[i], &rset))
86.                     recieve_data(clients, i);
87.         }
88.         return 0;
89.     }
90.
91.     void recieve_data(int clients[], int i)
92.     {
93.         char buffer[BUFFER_SIZE];
94.         int read = recv(clients[i], buffer, BUFFER_SIZE, 0);
95.         if (read <= 0)
96.         {
97.             close(clients[i]);
98.             clients[i] = 0;
99.         }
```

```
100.        else
101.        {
102.        buffer[read] = 0;
103.        printf("Сообщение от %d клиенту - %s\n", i, buffer);
104.        }
105.    }
106.
107.
108.    int max(int listener, int arr[], int count)
109.    {
110.        int res = listener;
111.        for (int i = 0; i < count; i++)
112.            if (arr[i] > res)
113.                res = arr[i];
114.        return res;
115.    }
116.
117.
118.    int insert(int s, int clients[], int count)
119.    {
120.        for (int i= 0; i < count; i++)
121.            if (clients[i] == 0)
122.            {
123.                clients[i] = s;
124.                return 0;
125.            }
126.        return -1;
127.    }
```

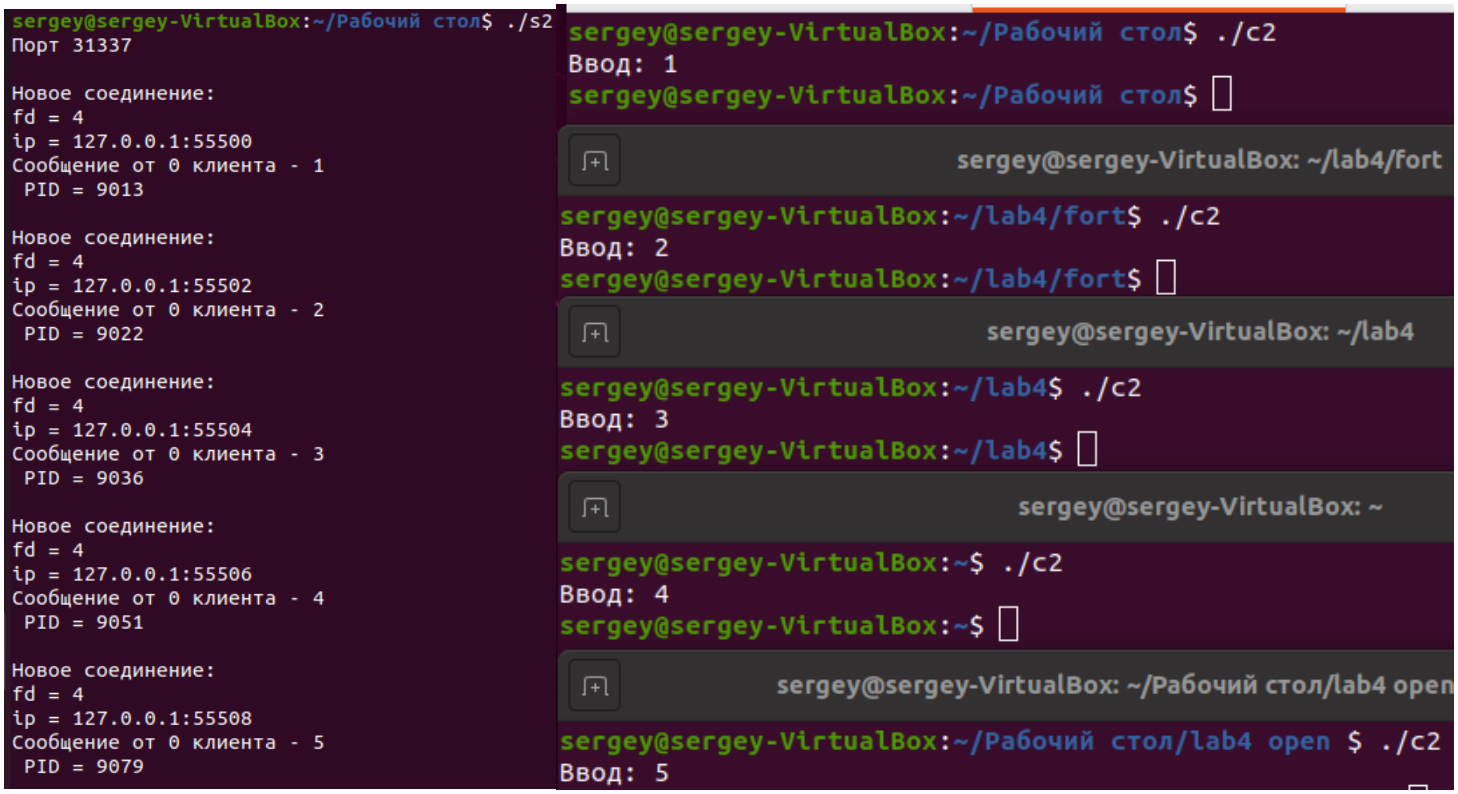
Листинг 4: Код клиента

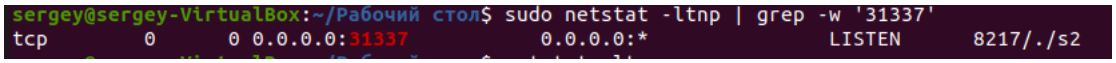
```
1.#include <sys/types.h>
2.#include <sys/socket.h>
3.#include <netinet/in.h>
4.#include <unistd.h>
5.#include <string.h>
6.#include <stdlib.h>
7.#include <stdio.h>
8.
9.#define SOCKET PORT 31337
10.     #define BUFFER SIZE 512
11.
12.     int main(int argc, char** argv)
```

```
13.     {
14.         int sock;
15.         struct sockaddr_in addr;
16.
17.         sock = socket(PF_INET, SOCK_STREAM, 0);//SOCK_STREAM - обеспечивает
    последовательное, надежное, двустороннее соединение
18.                                                 //PF_INET - IPv6 Internet protocols
19.
20.         if (sock < 0)
21.         {
22.             perror("Сообщение клиента: невозможно открыть сокет!\n");
23.             exit(1);
24.         }
25.
26.         addr.sin_family = PF_INET;
27.         addr.sin_port = htons(SOCKET_PORT); //Функция htons() преобразует узловой
    порядок расположения байтов положительного короткого
28.                                             //целого hostshort в сетевой порядок
    расположения байтов.
29.         addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK); //Функция htonl() преобразует
    узловой порядок расположения байтов положительного
30.                                                     //целого hostlong в сетевой
    порядок расположения байтов.
31.
32.         if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) //connect -
    системный вызов соединение сокета с другим сокетом
33.         {
34.             perror("Соединение невозможно!\n");
35.             exit (2);
36.
37.         }
38.
39.         char buf[BUFFER_SIZE];
40.         char nbuf[BUFFER_SIZE] ;
41.         printf("Ввод: ");
42.         scanf("%s", buf);
43.
44.         int pid = getpid();
45.         char mypid[6];   // ex. 34567
46.         sprintf(mypid, "%d", pid);
47.
```

```
48.          strcat(buf, "\n PID = ");
49.          strcat(buf, mypid);
50.          send(sock, buf, strlen(buf), 0);
51.
52.          close(sock) ;
53.          exit (0);
54.      }
```

Результат работы программы:





Порт 31337 в режиме «прослушивания», используется процессом «./s2».