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: Кондрашова О.П.
Группа: ИУ7-65Б
Преподаватель: Рязанова Н.Ю.

Москва, 2020г

Задание 1: Организовать взаимодействие параллельных процессов на отдельном компьютере.

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 <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/socket.h> //socket, bind

#define MSG_LEN 256
#define SOCKET_NAME “socket.soc”

int sockfd;

int main(void)
{
    char msg[MSG_LEN];
    struct sockaddr client_addr;

// создание сокета (семейство адресов – AF_UNIX,
// тип – SOCK_DGRAM – датаграммный сокет).
// протокол – 0, выбирается по умолчанию

    sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (sockfd < 0)
    {
        perror(«Error in socket»);
        return sockfd;
    }

// семейство адресов, которым мы будем пользоваться
    client_addr.sa_family = AF_UNIX;
// имя файла сокета
    strcpy(client_addr.sa_data, SOCKET_NAME);


// связывание сокета с заданным адресом
// параметры bind – дескриптор сокета, указатель на структуру, длина структуры

    if (bind(sockfd, &client_addr, sizeof(client_addr)) < 0)
    {
        printf(«Can'tbind name to socket\n»);
        close(sockfd);
        unlink(SOCKET_NAME);
        perror(«Error in bind(): «);
        return -1;
    }

// программа-сервер становится доступна для соединения по заданному адресу
// (имени файла)
```

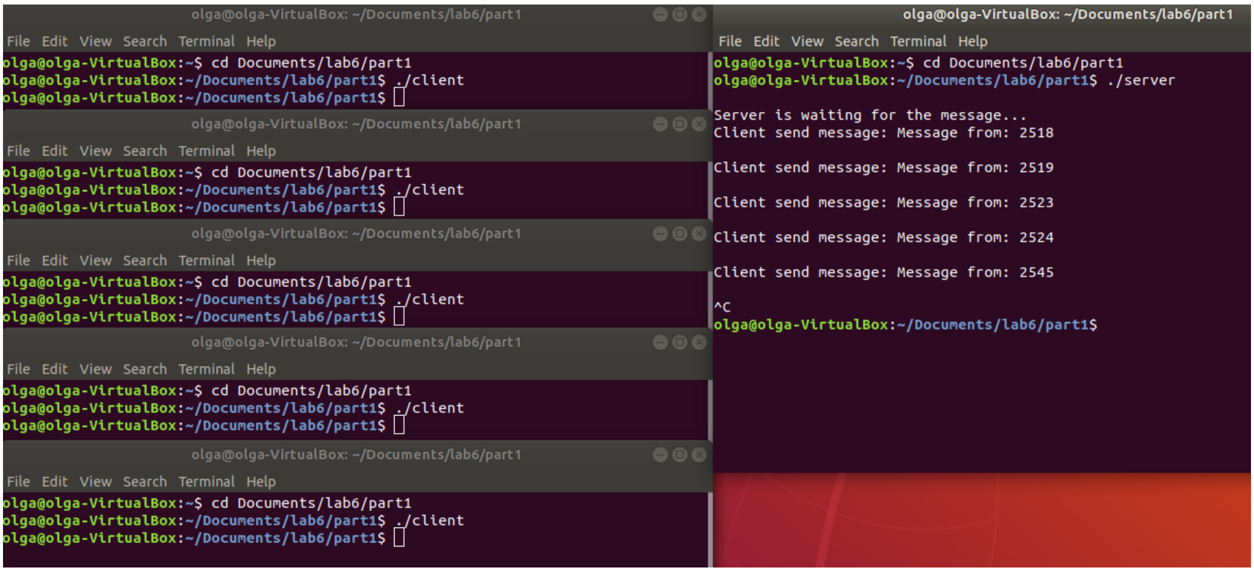
```
50.
51.     printf(«\nServer is waiting for the message…\n»);
52.
53. // сервер блокируется на функции recv() и ждет сообщения от процессов-клиентов
54.
55.     for(;;)
56.     {
57.         int recievedSize = recv(sockfd, msg, sizeof(msg), 0);
58.         if (recievedSize < 0)
59.         {
60.             close(sockfd);
61.             unlink(SOCKET_NAME);
62.             perror(«Error in recv(): «);
63.             return recievedSize;
64.         }
65.
66.         msg[recievedSize] = 0;
67.         printf(«Client send message: %s\n», msg);
68.     }
69.
70.     printf(«Closing socket\n»);
71.     close(sockfd);
72.     unlink(SOCKET_NAME);
73.     return 0;
74. }
```

Листинг 2. Код клиента

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include <unistd.h>
5.  #include <sys/types.h>
6.  #include <sys/socket.h>
7.
8.  #define MSG_LEN 256
9.  #define SOCKET_NAME "socket.soc"
10.
11. int main(void)
12. {
13.
14. // создание сокета (семейство адресов - AF_UNIX,
15. // тип - SOCK_DGRAM - датаграммный сокет).
16. // протокол - 0, выбирается по умолчанию
17.
18.     int sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
19.     if (sockfd < 0)
20.     {
21.         printf("Error in socket");
22.         return sockfd;
23.     }
24.
25.     struct sockaddr server_addr;
26.     server_addr.sa_family = AF_UNIX;
27.     strcpy(server_addr.sa_data, SOCKET_NAME);
28.
29.     char msg[MSG_LEN];
30.     sprintf(msg, "Message from: %d\n", getpid());
31.
32. // передаем сообщение серверу
33. // параметры sendto - дескриптор сокета, адрес буфера для передачи
34. // данных, длина буфера, дополнительные флаги, адрес сервера, его длина
35.
36.     sendto(sockfd, msg, strlen(msg), 0, &server_addr, sizeof(server_addr));
37.
38.     close(sockfd);
```

```
    return 0;
}
```

Запустим 5 клиентов одновременно:



Созданный файл:

В процессе-сервере с помощью вызова socket() создается сокет семейства AF_UNIX с типом SOCK_DGRAM. С помощью системного вызова bind() происходит связка сокета с локальным адресом. Сервер блокируется на функции recv () и ждет сообщения от процессов-клиентов. В процессе-клиенте создается сокет семейста AF_UNIX с типом SOCK_DGRAM с помощью системного вызова socket(). С помощью функции sendto() отправляется сообщение к процессу-серверу.

Задание 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.

Листинг 3. Код сервера

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include <errno.h>
5.  #include <unistd.h>
6.  #include <sys/types.h>
7.  #include <sys/socket.h>
8.  #include <sys/select.h>
9.  #include <arpa/inet.h>
10. #include <netdb.h>
11.
12. #define MSG_LEN 256
13. #define SOCK_ADDR "localhost"
14. #define SOCK_PORT 9999
15.
16. #define MAX_CLIENTS 10
17. int clients[MAX_CLIENTS] = { 0 };
18.
19.
20. void manageConnection(unsigned int fd)
21. {
22.     struct sockaddr_in client_addr;
23.     int addrSize = sizeof(client_addr);
24.
25. // установка соединения в ответ на запрос клиента
26. // функция accept() возвращает новый сокет, открытый
27. // для обмена данными с клиентом, запросившим соединение
28.
29.     int incom = accept(fd, (struct sockaddr*) &client_addr, (socklen_t*) &addrSize);
30.     if (incom < 0)
31.     {
32.         perror("Error in accept(): ");
33.         exit(-1);
34.     }
35.
36.     printf("\nNew connection: \nfd = %d \nip = %s:%d\n", incom,
37.                               inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port
    ));
38.
39.
40. // сохраняем дескриптор в первый свободный
41.
42.     for (int i = 0; i < MAX_CLIENTS; i++)
43.     {
44.         if (clients[i] == 0)
45.         {
```

```
46.             clients[i] = incom;
47.             printf("Managed as client #%d\n", i);
48.             break;
49.         }
50.     }
51.
52. }
53.
54. void manageClient(unsigned int fd, unsigned int client_id)
55. {
56.     char msg[MSG_LEN];
57.     memset(msg, 0, MSG_LEN);
58.
59.     struct sockaddr_in client_addr;
60.     int addrSize = sizeof(client_addr);
61.
62. // сервер блокируется на функции recv() и ждет сообщения от процессов-клиентов
63.
64.     int recvSize = recv(fd, msg, MSG_LEN, 0);
65.     if (recvSize == 0)
66.     {
67.         // возвращает адрес машины, подключившейся к сокету
68.
69.         getpeername(fd, (struct sockaddr*) &client_addr, (socklen_t*) &addrSize);
70.         printf("User %d disconnected %s:%d \n", client_id, inet_ntoa(client_addr.sin_ad
    dr), ntohs(client_addr.sin_port));
71.         close(fd);
72.         clients[client_id] = 0;
73.     }
74.     else
75.     {
76.         msg[recvSize] = '\0';
77.         printf("Message from %d client: %s\n", client_id, msg);
78.     }
79. }
80.
81. int main(void)
82. {
83.     // создание сетевого сокета (домен AF_INET,
84.     // тип сокета - SOCK_STREAM, сокет должен быть потоковым)
85.     // протокол - 0, выбирается по умолчанию
86.
87.     int my_socket = socket(AF_INET, SOCK_STREAM, 0);
88.     if (my_socket < 0)
89.     {
90.         perror("Error in sock(): ");
91.         return my_socket;
92.     }
93.
94.     struct sockaddr_in server_addr;
95. // семейство адресов, которыми мы будем пользоваться
96.     server_addr.sin_family = AF_INET;
97. // значение порта. Функция htons() переписываетзначение порта так, чтобы      // порядо
    к байтов соответствовал принятому в интернете
98.     server_addr.sin_port = htons(SOCK_PORT);
99. // адрес (наша программа-сервер зарегистрируется на всех адресах машины,
100.       // на которой она выполняется)
101.           server_addr.sin_addr.s_addr = INADDR_ANY;
102.
103.       // связывание сокета с заданным адресом
104.
105.           if (bind(my_socket, (struct sockaddr*) &server_addr, sizeof(server_addr)) <
    0)
106.           {
107.               perror("Error in bind():");
108.               return -1;
109.           }
110.           printf("Server is listening on the %d port\n", SOCK_PORT);
111.
```

```
112.        // переводим сервер в режим ожидания запроса на соединение
113.        // (второй параметр - максимальное количество соединений, обрабатываемых
114.        // одновременно)
115.
116.            if (listen(my_socket, 3) < 0)
117.            {
118.                perror("Error in listen(): ");
119.                return -1;
120.            }
121.            printf("Waiting for connections\n");
122.
123.            for (;;)
124.            {
125.                fd_set readfds;
126.                int max_fd;
127.                int active_clients;
128.
129.                FD_ZERO(&readfds);
130.                FD_SET(my_socket, &readfds);
131.                max_fd = my_socket;
132.
133.                for (int i = 0; i < MAX_CLIENTS; i++)
134.                {
135.                    int fd = clients[i];
136.
137.                    if (fd > 0)
138.                    {
139.                        FD_SET(fd, &readfds);
140.                    }
141.
142.                    max_fd = (fd > max_fd) ? (fd) : (max_fd);
143.                }
144.
145.        // сервер блокируется на вызове функции select(), она возвращает управление,
146.        // если хотя бы  один из проверяемых сокетов готов к выполнению
147.        // соответствующей операции
148.
149.                struct timeval timeout = {20, 0};
150.                active_clients = select(max_fd + 1, &readfds, NULL, NULL, &timeout);
151.
152.                if (active_clients == 0)
153.                {
154.                    printf("\nServer closed connection by timeout\n\n");
155.                    return 0;
156.                }
157.
158.                if (active_clients < 0 && (errno != EINTR))
159.                {
160.                    perror("Error in select():");
161.                    return active_clients_count;
162.                }
163.
164.        // если есть новые соединения, вызывается функция manageConnection()
165.
166.                if (FD_ISSET(my_socket, &readfds))
167.                {
168.                    manageConnection(my_socket);
169.                }
170.
171.        // обход массива дескрипторов
172.        // если дескриптор находится в наборе, запускается функция manageClient()
173.
174.                for (int i = 0; i < MAX_CLIENTS; i++)
175.                {
176.                    int fd = clients[i];
177.                    if ((fd > 0) && FD_ISSET(fd, &readfds))
178.                    {
179.                        manageClient(fd, i);
180.                    }
```

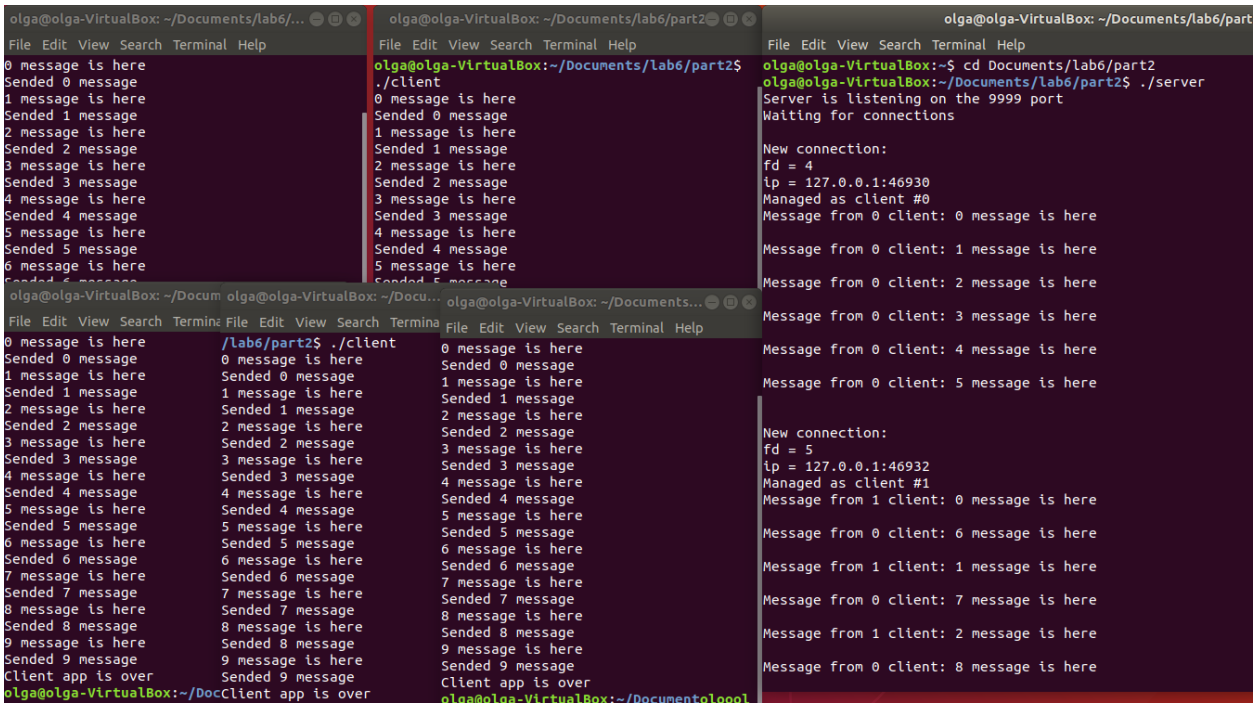
```
            }
        }

        return 0;
    }
```

Листинг 4. Код клиента

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>

#define MSG_LEN 256
#define SOCK_ADDR "localhost"
#define SOCK_PORT 9999

int main(void)
{
    srand(time(NULL));

// создание сетевого сокета (домен AF_INET,
// тип сокета - SOCK_STREAM, сокет должен быть потоковым)
// протокол - 0, выбирается по умолчанию

    int sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        perror("Error in sock(): ");
        return sock;
    }

// преобразование доменного имени сервера в его сетевой адрес

    struct hostent* host = gethostbyname(SOCK_ADDR);    // /etc/hosts
    if (!host)
    {
        perror("Error in gethostbyname(): ");
        return -1;
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = PF_INET;
    server_addr.sin_port = htons(SOCK_PORT);
    server_addr.sin_addr = *((struct in_addr*) host->h_addr_list[0]);

// установка соединения

    if (connect(sock, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0)
    {
        perror("Error in connect():");
        return -1;
    }

    char msg[MSG_LEN];
    for (int i = 0; i < 10; i++)
    {
        memset(msg, 0, MSG_LEN);
        sprintf(msg, "%d message is here\n", i);
        printf("%s", msg);
```

```
59.
60.         // запись данных в сокет
61.         if (send(sock, msg, strlen(msg), 0) < 0)
62.         {
63.             perror("Error in send(): ");
64.             return -1;
65.         }
66.
67.         printf("Sended %d message\n", i);
68.
69.         int wait_time = 1 + rand() % 3;
70.         sleep(wait_time);
71.     }
72.
73.     printf("Client app is over\n");
74.     return 0;
75. }
```

Запустим 5 клиентов одновременно:



Отправив все сообщения, клиенты завершаются:

```
User 2 disconnected 127.0.0.1:60284
Message from 4 client: 5 message is here

User 1 disconnected 127.0.0.1:60282
```

Порт:

```
olga@olga-VirtualBox:~$ lsof -i | grep -E LISTEN
server  2316 olga    3u  IPv4  40088      0t0  TCP *:9999 (LISTEN)
```

В процессе-сервере с помощью вызова socket() создается сокет семейства AF_INET с типом SOCK_STREAM. С помощью системного вызова bind() происходит связка сокета с адресом, прописанным в SOCKET_ADDRESS. С помощью вызова listen() сокету сообщается, что должны приниматься новые соединения. На каждой итерации цикла создается новый набор дескрипторов set. В него заносятся сокет сервера и сокеты клиентов с помощью функции FD_SET. После этого сервер блокируется на вызове функции select(), она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции. После выхода из блокировки, проверяется наличие новых соединений. При наличии таковых вызывается функция connectHandler. В этой функции с помощью accept() принимается новое соединение, а также создается сокет, который записывается в массив файловых дескрипторов. Затем происходит обход массива дескрипторов, и, если дескриптор находится в наборе дескрипторов, то запускается функция clientHandeler(). В ней осуществляется считывание с помощью recv() и вывод сообщения от клиента. Если recv() возвращает нулевое значение, значит соединение было сброшено. В таком случае выводится сообщение о закрытии сокета. В процессе-клиенте создается сокет семейста AF_INETс типом SOCK_STREAM с помощью системного вызова socket(). С помощью функции gethostbyname() доменный адрес преобразуется в сетевой и с его помощью можно установить соединение, используя функцию connect(). Затем происходит отправка сообщений серверу.