



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 предмету: «Операционные системы»

Тема: Сокеты

Студент: Лаврова А. А.,
Группа: ИУ7-65Б
Преподаватель: Рязанова Н. Ю.

Москва, 2020 г.

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 (сервер):

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

#define SOCK_NAME "my_sock_unix"
#define SIZE 256
int sock;

void recv_msg(int sock) {
    char buffer[SIZE];

    // блокировка на функции recv(), ожидание сообщения от клиента
    int size = recv(sock, buffer, sizeof(buffer), 0);
    if (size < 0)
    {
        perror("server: Can't receive data!\n");
        close(sock);
        remove(SOCK_NAME);
        exit(3);
    }
    buffer[size] = 0;
    printf("%s\n", buffer);
}

void sigint_handler(int signum)
{
    close(sock);
    remove(SOCK_NAME);
    exit(1);
}

int main(void) {
```

```

struct sockaddr_un addr;

// создание сокета
// AF_UNIX - домен, SOCK_DGRAM - датаграммный сокет, 0 - протокол (по
умолчанию)
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
if (sock < 0)
{
    perror("server: Can't open socket!\n");
    exit(1);
}
signal(SIGINT, sigint_handler);

// используемое семейство адресов
addr.sun_family = AF_UNIX;
// имя файла сокета
strcpy(addr.sun_path, SOCK_NAME);

// связывание сокета с заданным адресом
if (bind(sock, (struct sockaddr *)&addr, sizeof(struct sockaddr)) < 0)
{
    perror("server: Can't bind name to socket!\n");
    close(sock);
    remove(SOCK_NAME);
    exit(2);
}
printf("socket: %s\n", SOCK_NAME);

for(;;)
{
    recv_msg(sock);
}

close(sock);
remove(SOCK_NAME);
return 0;
}

```

Листинг (клиент):

```

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

#define SOCK_NAME "my_sock_unix"
#define MSG_LEN 256

int main(int argc, char *argv[])
{

```

```

int sock;

struct sockaddr_un addr;

// создание сокета
// AF_UNIX - домен, SOCK_DGRAM - датаграммный сокет, 0 - протокол (по
умолчанию)
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
if (sock < 0)
{
perror("client: Can't open socket\n");
exit(1);
}

addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, SOCK_NAME);

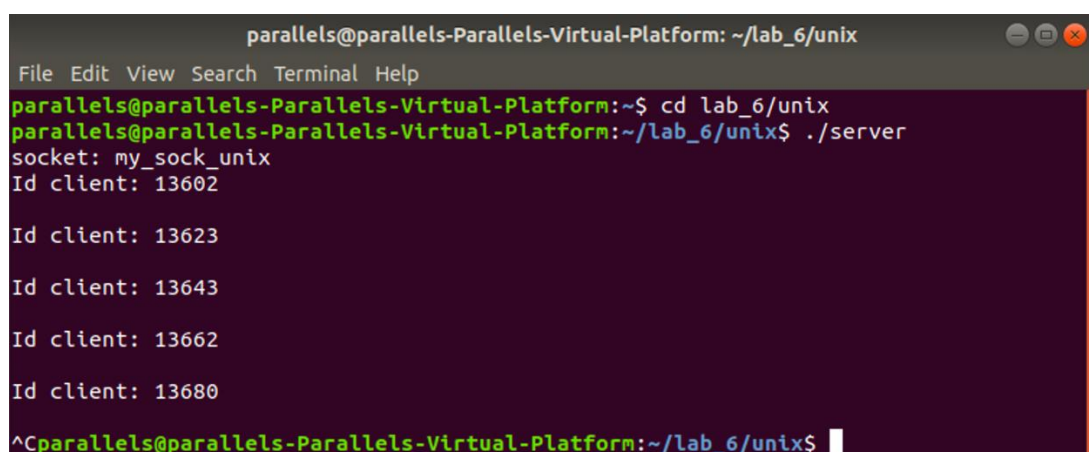
char msg[MSG_LEN];
sprintf(msg, "Id client: %d\n", getpid());

// передача сообщения серверу
sendto(sock, msg, strlen(msg), 0, (struct sockaddr *)&addr, sizeof(addr));

close(sock);
return 0;
}

```

Результат работы программы:



```

parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~$ cd lab_6/unix
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/unix$ ./server
socket: my_sock_unix
Id client: 13602

Id client: 13623

Id client: 13643

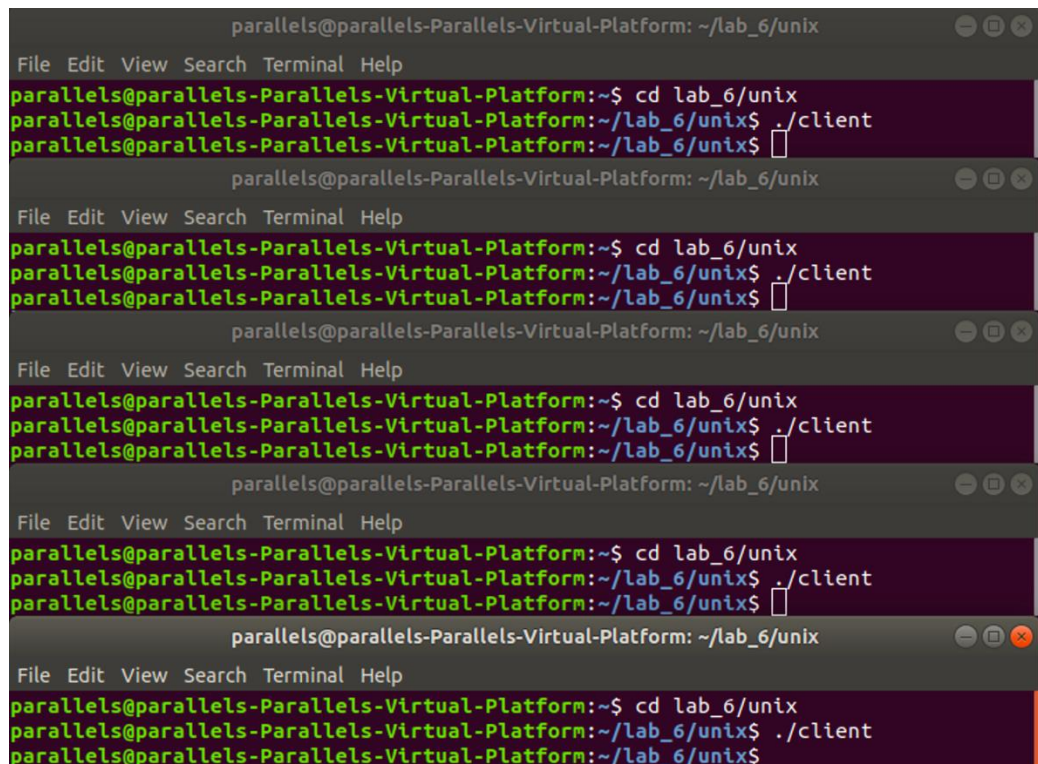
Id client: 13662

Id client: 13680

^Cparallels@parallels-Parallels-Virtual-Platform:~/lab_6/unix$

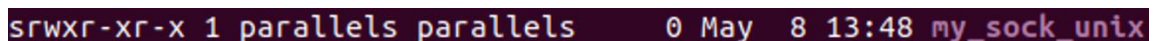
```

Рис. 1: Результат работы сервера



The image shows four identical terminal windows stacked vertically. Each window has a title bar that reads 'parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/unix'. The terminal content in each window is as follows:
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~\$ cd lab_6/unix
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/unix\$./client
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/unix\$

Рис. 2: Результат работы клиента



```
srwxr-xr-x 1 parallels parallels 0 May  8 13:48 my_sock_unix
```

Рис.3: Сокет в файловой системе

Часть 2

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

Задание:

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 (сервер):

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SOCKET_PORT 9797
#define BUFFER_SIZE 256
#define LISTEN_QUEUE_SIZE 256

#define CLIENTS 5

void recieve_data(int clients[], int i)
{
    char buffer[BUFFER_SIZE];

    // блокировка на функции recv(), ожидание сообщения от клиента
    int read = recv(clients[i], buffer, BUFFER_SIZE, 0);
    if (read <= 0)
    {
        close(clients[i]);
        clients[i] = 0;
    }
    else
    {
        buffer[read] = 0;
        printf("Client: %d\nMessage: %s\n\n", i, buffer);
    }
}

int max(int listener, int arr[], int count)
{
    int res = listener;
    for (int i = 0; i < count; i++)
        if (arr[i] > res)
            res = arr[i];

    return res;
}

int insert(int s, int clients[], int count)
{

```

```

    for (int i = 0; i < count; i++)
        if (clients[i] == 0)
        {
            clients[i] = s;
            return 0;
        }
    return -1;
}

int main(void)
{
    int sockL;
    struct sockaddr_in addr;
    int clients[CLIENTS] = {0};
    fd_set rset;

    int stop_flag = 0;

    // создание сокета
    // AF_INET - домен, SOCK_STREAM - тип сокета, 0 - тип протокола (по умолчанию)
    sockL = socket(AF_INET, SOCK_STREAM, 0);
    if (sockL < 0)
    {
        perror("server: Can't open socket!\n");
        exit(1);
    }
    fcntl(sockL, F_SETFL, O_NONBLOCK);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(SOCKET_PORT);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);

    // связываем сокет с заданным адресом
    if (bind(sockL, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("server: Can't bind to address!\n");
        exit(2);
    }

    // перевод сервера в режим ожидания
    listen(sockL, LISTEN_QUEUE_SIZE);

    //timeout
    struct timeval timeout;
    timeout.tv_sec = 120;
    timeout.tv_usec = 0;

    while(1)
    {
        FD_ZERO(&rset);
        FD_SET(sockL, &rset);
        for (int i = 0; i < CLIENTS; i++)
            if (clients[i] != 0)
                FD_SET(clients[i], &rset);

        int mx = max(sockL, clients, CLIENTS);

```

```

// ожидаем поступление данных
if (select(mx+1, &rset, NULL, NULL, &timeout) <= 0)
{
    perror("server: Timeout\n");
    exit(3);
}

// проверяем на наличие новых соединений
if (FD_ISSET(sockL, &rset))
{
    // установка соединения, accept() возвращает новый сокет
    int sock = accept(sockL, NULL, NULL);
    if (sock < 0)
    {
        perror("server:Can't accept \n");
        exit(4);
    }

    printf("New connection:\nfd = %d\nip = %s:%d\n\n", sock,
inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));

    // создание неблокирующего ввода/вывода
    fcntl(sock, F_SETFL, O_NONBLOCK);
    if (insert(sock, clients, CLIENTS) < 0)
    {
        perror("server: Too many clients!\n");
        exit(5);
    }
}

// обход массива дескрипторов
for (int i = 0; i < CLIENTS; i++)
{
    if (FD_ISSET(clients[i], &rset))
        recieve_data(clients, i);
}
}
return 0;
}

```

Листинг (клиент):

```

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>

#define SOCKET_PORT 9797
#define BUFFER_SIZE 256
#define SOCK_ADDR "localhost"

```



```

int main(int argc, char** argv)
{
    int sock;
    struct sockaddr_in addr;

    // создание сетевого сокета
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        perror("client: Can't open socket!\n");
        exit(1);
    }

    // преобразование доменного имени в сетевой адрес
    struct hostent* host = gethostbyname(SOCK_ADDR);
    if(!host)
    {
        perror("Error in gethostbyname!");
        exit(2);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(SOCKET_PORT);
    addr.sin_addr = *((struct in_addr*) host->h_addr_list[0]);

    // установка соединения
    if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("Can't connect!\n");
        exit(3);
    }

    char buf[BUFFER_SIZE];
    char nbuf[BUFFER_SIZE];
    printf("Input message: ");
    scanf("%s", buf);
    sprintf(buf, "%s (pid=%d)", buf, getpid());

    // запись данных в сокет
    send(sock, buf, strlen(buf), 0);

    close(sock);
    exit(0);
}

```

Результат работы программы:

```

server 13019 parallels 3u IPv4 231937 0t0 TCP *:9797 (LISTEN)

```

Рис. 4: Порт

```
parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~$ cd lab_6/inet
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./server
New connection:
fd = 4
ip = 0.0.0.0:9797

Client: 0
Message: 1 (pid=26234)

New connection:
fd = 4
ip = 0.0.0.0:9797

Client: 0
Message: 2 (pid=26252)

New connection:
fd = 4
ip = 0.0.0.0:9797

Client: 0
Message: 3 (pid=26269)

Client: 0
Message: 3 (pid=26269)

New connection:
fd = 4
ip = 0.0.0.0:9797

Client: 0
Message: 4 (pid=26287)

New connection:
fd = 4
ip = 0.0.0.0:9797

Client: 0
Message: 5 (pid=26306)

server: Timeout
: Success
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$
```

Рис. 5-6: Результат работы сервера

```
parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./client
Input message: 1
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$

parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./client
Input message: 2
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$

parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./client
Input message: 3
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$

parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./client
Input message: 4
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$

parallels@parallels-Parallels-Virtual-Platform: ~/lab_6/inet
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$ ./client
Input message: 5
parallels@parallels-Parallels-Virtual-Platform:~/lab_6/inet$
```

Рис. 7: Результат работы клиента