

**«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
СПбГЭТУ «ЛЭТИ»**

Кафедра вычислительной техники

Отчет по по лабораторной работе №11 по дисциплине
«Организация процессов и программирование в среде Linux»
Тема: «Взаимодействие процессов через сокеты»

Студент гр.8306

Преподаватель

Слепов А.Э.

Разумовский Г.В.

Санкт-Петербург

2021

Цель работы

Знакомство с организацией сокетов, системными функциями, обеспечивающими управление сокетами, и их использованием для решения задач межпроцессного взаимодействия.

Задание

1. Написать две программы (сервер и клиент) , которые обмениваются сообщениями через потоковые сокеты. Клиенты проверяют возможность соединения с сервером и в случае отсутствия соединения или истечения времени ожидания отправки сообщения завершают работу. После соединения с сервером они генерируют случайную последовательность чисел и выводят ее на экран, а затем отсылают серверу. Сервер в течение определенного времени ждет запросы от клиентов и в случае их отсутствия завершает работу. При поступлении запроса от клиента сервер порождает обслуживающий процесс, который принимает последовательность чисел, упорядочивает ее и выводит на экран, а затем отправляет обратно клиенту и завершают работу. Клиент полученную последовательность выводит на экран и заканчивает свою работу.

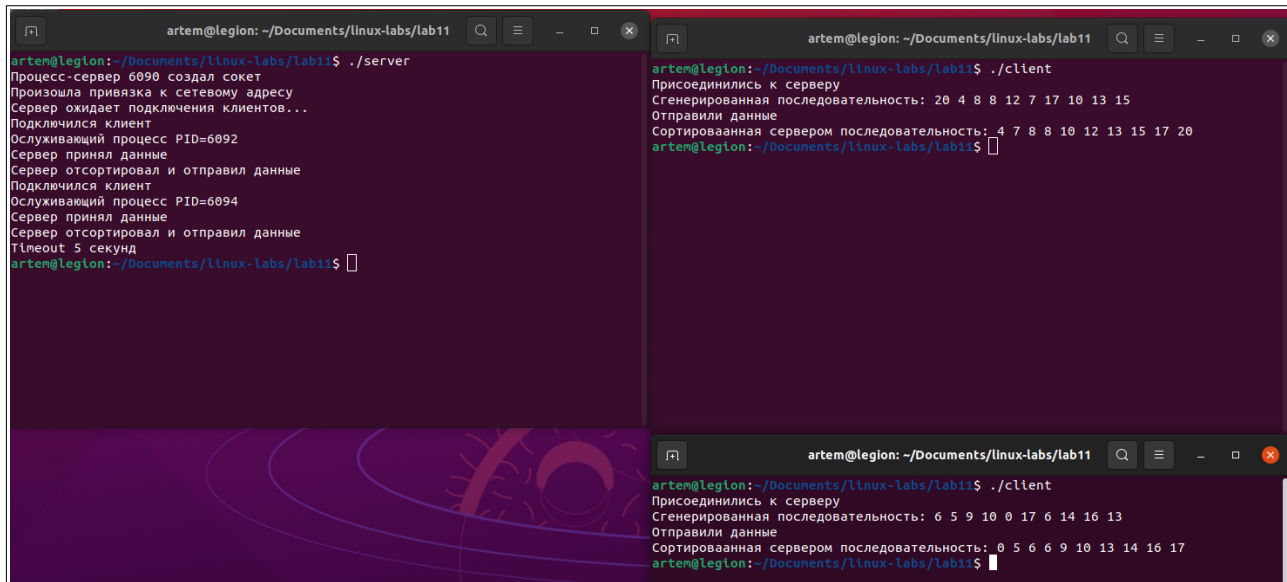
2. Откомпилировать программы и запустить их.

Порядок выполнения работы

Разработано 2 программы: сервер и клиент. Сервер открывает сокет, связывает его к своему сетевому адресу и ждет запросы от клиентов в течение 5 секунд (с помощью функции select). Если запрос клиента поступил, сервер порождает обслуживающий процесс, который принимает от клиента последовательность чисел, сортирует ее и возвращает клиенту. Если после обслуживания последнего клиента, сервер не получает запрос в течение 5 секунд, он завершает свою работу.

При запуске клиент пытается присоединиться к серверу в течение 5 секунд. Если соединение установлено, клиент генерирует случайную последовательность чисел и отправляет ее серверу. Сервер присылает в ответ отсортированный массив чисел, клиент выводит его на экран.

Результаты запуска сервера и двух клиентов приведены на рисунке 1.



```
artem@legion: ~/Documents/linux-labs/lab11$ ./server
Процесс-сервер 6090 создал сокет
Произошла привязка к сетевому адресу
Сервер ожидает подключения клиентов...
Подключился клиент
Ослуживающий процесс PID=6092
Сервер принял данные
Сервер отсортировал и отправил данные
Подключился клиент
Ослуживающий процесс PID=6094
Сервер принял данные
Сервер отсортировал и отправил данные
Timeout 5 секунд
artem@legion: ~/Documents/linux-labs/lab11$
```

```
artem@legion: ~/Documents/linux-labs/lab11$ ./client
Присоединились к серверу
Сгенерированная последовательность: 20 4 8 8 12 7 17 10 13 15
Отправили данные
Сортированная сервером последовательность: 4 7 8 8 10 12 13 15 17 20
artem@legion: ~/Documents/linux-labs/lab11$
```

```
artem@legion: ~/Documents/linux-labs/lab11$ ./client
Присоединились к серверу
Сгенерированная последовательность: 6 5 9 10 0 17 6 14 16 13
Отправили данные
Сортированная сервером последовательность: 0 5 6 6 9 10 13 14 16 17
artem@legion: ~/Documents/linux-labs/lab11$
```

Рисунок 1. Результаты запуска программ

Выводы

В ходе работы были изучены механизмы организации сокетов, системными функциями, обеспечивающими управление сокетами, и их использованием для решения задач межпроцессного взаимодействия в операционной системе Ubuntu.

ПРИЛОЖЕНИЕ А

Текст программы сервера

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h> //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include <fcntl.h>
#include <unistd.h>

int sock, attached_socket;
//char *message;
int reply[1024];

#define MSG_LEN 10

int cmp(const void *a, const void *b) {
    return *(int*)a - *(int*)b;
}

void service_process(){
    //pid_t pid;
    switch(fork()) {
        case 0:
            printf("Осуживающий процесс PID=%d\n", getpid());
            close(sock);
            if(recv(attached_socket, reply , 1024 , 0) > 0){
                printf("Сервер принял данные\n");
                qsort(reply, MSG_LEN, sizeof(int), cmp);
                printf("Сервер отсортировал и отправил данные\n");
            }
            send(attached_socket, reply, MSG_LEN*4, 0);
            close(attached_socket);
            exit(EXIT_SUCCESS);
            break;
        default:
            break;
    }
}

int main(int argc , char *argv[])
{
    struct sockaddr_in server , client;
    int rv;
    struct timeval tv;

    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Ошибка при создании сокета\n");
    }
    printf("Процесс-сервер %d создал сокет\n", getpid());

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    if( bind(sock,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        printf("bind не произошел\n");
        exit(EXIT_FAILURE);
    }
}
```

```

printf("Произошла привязка к сетевому адресу\n");

listen(sock , 5);

printf("Сервер ожидает подключения клиентов...\n");

fd_set readfds;
FD_ZERO(&readfds);
FD_SET(sock, &readfds);
tv.tv_sec = 5;

while(1){
    rv = select(sock+1, &readfds, NULL, NULL, &tv);
    if(rv > 0){
        attached_socket = accept(sock, (struct sockaddr *)&client,
(socklen_t*)&client);
        if(attached_socket>0){
            printf("Подключился клиент\n");
            service_process();
        }
    }
    else{
        printf("Timeout 5 секунд\n");
        break;
    }
    tv.tv_sec = 5;
}

close(sock);
return 0;
}

```

ПРИЛОЖЕНИЕ Б

Текст программы клиента

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

#define MSG_LEN 10

int *arr;

void generate_nums(){
    arr = (int*)malloc(MSG_LEN);
    printf("Сгенерированная последовательность: ");
    for(int i = 0; i < MSG_LEN; i++){
        arr[i] = rand() % 21;
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    int server_reply[1024];
    int rv;
    struct timeval tv;

    srand(time(NULL));

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);

    if (sock == -1)
    {
        printf("Ошибка при создании сокета\n");
    }

    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(sock, &readfds);
    tv.tv_sec = 5;
    rv = select(sock+1, &readfds, NULL, NULL, &tv);
    if(rv > 0){
        if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0){
            printf("No connect\n");
        }
        else{
            printf("Присоединились к серверу\n");
            generate_nums();

            //Send some data
            if( send(sock , arr , MSG_LEN*4 , 0) < 0)
```

```

    {
        printf("Отправка не удалась\n");
        exit(EXIT_FAILURE);
    }
    printf("Отправили данные\n");

    //Receive a reply from the server
    if( recv(sock, server_reply , 1024 , 0) < 0)
    {
        printf("Прием не удался\n");
        exit(EXIT_FAILURE);
    }
    printf("Сортироваанная сервером последовательность: ");
    for(int i = 0; i < MSG_LEN; i++){
        printf("%d ", server_reply[i]);
    }
    printf("\n");
}
}else{
    printf("Timeout 5 секунд");
}

return 0;
}

```