Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**Семейство протоколов TCP/IP. Сокеты в UNIX и работа с ними**

Лабораторная работа №11 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_ Кудряшов И. С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** объединить полученные в процессе выполнения предыдущих лабораторных работ знания в рамках одной комплексной программы.

**Задание:**

Написать программу-клиент и программу-сервер для выполнения следующей задачи: сервер запускается и ждёт запросов от клиентов; клиент подключается к серверу и может запрашивать файлы, находящиеся на сервере (в папке с программой-сервером), задавая полное имя файла; если файл найден, сервер передаёт файл клиенту, если не найден, то сервер возвращает сообщение о том, что файл не найден.

Сервер должен параллельно обслуживать несколько клиентов (для каждого свой процесс). Клиент может запрашивать файлы последовательно вводя имена. Выход из программы клиента можно осуществить по текстовой команде (например, «exit»). Сервер так же должен иметь возможность вводить команды (например, для закрытия программы-сервера).

Протокол транспортного уровня – TCP.

При запуске программы-сервера через параметры командной строки передаётся порт, на котором будет работать сервер. Если данный порт занят, программа-сервер должна получить порт по усмотрению системы. При успешном запуске сервер должен выдать на экран номер порта, на котором он работает.

При запуске программы-клиента через параметры командной строки передаётся IP-адрес и порт (сокет) программы-сервера в формате <IP-address>:<port> (например, 192.168.0.100:4242).

**Содержание файла server.c:**

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <signal.h>

#define MAX\_FILENAME\_SIZE 256

long fsize(FILE \*fp)

{

    long prev = ftell(fp);

    fseek(fp, 0L, SEEK\_END);

    long sz = ftell(fp);

    fseek(fp, prev, SEEK\_SET);

    return sz;

}

void command(int fork\_pid)

{

    printf("Ready to recive commands\n");

    char command[MAX\_FILENAME\_SIZE];

    while (1)

    {

        if (kill(fork\_pid, 0) == -1)

            exit(-1);

        scanf("%s", command);

        if (strcmp(command, "exit") == 0)

        {

            kill(fork\_pid, SIGKILL);

            exit(0);

        }

        else if (strcmp(command, "help") == 0)

        {

            printf("Avalible commands:\n");

            printf("exit - closes app\n");

            printf("help - shows avalible commands\n");

        }

        else

        {

            printf("Unknown command, please use help to get list of avalible commands\n");

        }

    }

}

void client(struct sockaddr\_in cliaddr, int sockfd)

{

    while (1)

    {

        char filename[MAX\_FILENAME\_SIZE];

        int n = read(sockfd, filename, MAX\_FILENAME\_SIZE);

        if (n == 0)

        {

            close(sockfd);

            exit(0);

        }

        FILE \*fin = fopen(filename, "r");

        if (fin == NULL)

        {

            long statusmsg = -1;

            write(sockfd, &statusmsg, sizeof(statusmsg));

        }

        else

        {

            long filesize = fsize(fin);

            write(sockfd, &filesize, sizeof(filesize));

            char msg[filesize];

            fread(msg, sizeof(char), filesize, fin);

            fclose(fin);

            write(sockfd, msg, sizeof(msg));

        }

    }

}

void lisener(int port)

{

    int sockfd;

    struct sockaddr\_in servaddr;

    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(port);

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    sockfd = socket(PF\_INET, SOCK\_STREAM, 0);

    if (bind(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

        servaddr.sin\_port = 0;

        if (bind(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

        {

            perror(NULL);

            close(sockfd);

            exit(1);

        }

    }

    socklen\_t servlen = sizeof(servaddr);

    listen(sockfd, 5);

    getsockname(sockfd, (struct sockaddr \*)&servaddr, &servlen);

    printf("Listening on port: %d\n", ntohs(servaddr.sin\_port));

    while (1)

    {

        struct sockaddr\_in cliaddr;

        socklen\_t clilen = sizeof(cliaddr);

        int newsockfd = accept(sockfd, (struct sockaddr \*)&cliaddr, &clilen);

        if (fork() == 0)

            client(cliaddr, newsockfd);

    }

}

int main(int argv, char \*argc[])

{

    if (argv != 2)

    {

        perror("No port given");

        return -1;

    }

    int port = atoi(argc[1]);

    if (port == 0)

    {

        perror("Not correct port");

        return -1;

    }

    int fork\_pid = fork();

    if (fork\_pid == 0)

        lisener(port);

    else

        command(fork\_pid);

}

**Содержание файла client.c:**

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/mman.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <unistd.h>

#define MAX\_FILENAME\_SIZE 256

int main(int argv, char \*argc[])

{

    if (argv != 2)

    {

        perror("No address given");

        return -1;

    }

    int border = strstr(argc[1], ":") - argc[1];

    if (border + argc[1] == NULL)

    {

        perror("Not correct format of address");

        return -1;

    }

    char ip[border + 1];

    strncpy(ip, argc[1], border);

    ip[border] = '\0';

    int port = atoi(argc[1] + border + 1);

    char path\_to\_save[] = "downloads";

    struct sockaddr\_in servaddr;

    int sockfd = socket(PF\_INET, SOCK\_STREAM, 0);

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(port);

    inet\_aton(ip, &(servaddr.sin\_addr));

    if (connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

        printf("Can't connect to server\n");

        return 1;

    }

    printf("Ready to recive commands\n");

    char command[MAX\_FILENAME\_SIZE];

    while (1)

    {

        scanf("%s", command);

        if (strcmp(command, "exit") == 0)

        {

            break;

        }

        else if (strcmp(command, "help") == 0)

        {

            printf("Avalible commands:\n");

            printf("exit - closes app\n");

            printf("help - shows avalible commands\n");

            printf("any other command will be interpreted as filename\n");

        }

        else

        {

            write(sockfd, command, strlen(command) + 1);

            long status;

            int n = read(sockfd, &status, sizeof(status));

            if (n == 0)

            {

                close(sockfd);

                printf("Connection lost\n");

                exit(1);

            }

            else if (status == -1)

            {

                printf("File not found by server\n");

            }

            else

            {

                printf("Downloading file\n");

                long filesize = status;

                int comand\_len = strlen(command), path\_len = strlen(path\_to\_save);

                char \*save\_path = calloc(path\_len + comand\_len + 2, sizeof(char));

                strncpy(save\_path, path\_to\_save, path\_len);

                save\_path[path\_len] = '\_'; // здесь должен быть / пути но по абсолютно неясной причине оно не хочет писать в папку без прав администратора

                strncpy(save\_path + path\_len + 1, command, comand\_len);

                save\_path[path\_len + comand\_len + 1] = '\0';

                //здесь должна быть проверка наличия папки и создание при необходимости

                /\*struct stat st = {0};

                if (stat(path\_to\_save, &st) == -1)

                    mkdir(path\_to\_save, 0666);\*/

                int output\_file = open(save\_path, O\_RDWR | O\_CREAT, 0666);

                if (output\_file == -1)

                {

                    printf("File opening error %d \n", errno);

                    exit(-1);

                }

                ftruncate(output\_file, filesize);

                char \*output\_file\_data = (char \*)mmap(NULL, filesize, PROT\_WRITE | PROT\_READ, MAP\_SHARED, output\_file, 0);

                read(sockfd, output\_file\_data, filesize);

                munmap(output\_file\_data, filesize);

                close(output\_file);

                free(save\_path);

                printf("Downloading finished\n");

            }

        }

    }

    close(sockfd);

}

**Вывод:** В процессе выполнения лабораторной работы я на практике закрепил знания о различных аспектах UNIX.