**КЛИЕНТ-СЕРВЕР. СОЗДАНИЕ WEB-СЕРВЕРОВ**

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

студент группы 9091:

\_\_\_\_\_\_\_ Чалый С.М

Великий Новгород

2021

**Цель работы:** использовать полученные во время учебного процесса навыки и использовать их в лабораторной работе.

**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).

При возникновении трудностей в выполнении проконсультироваться у преподавателя.

**Реализация:**

**Server.c:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <netinet/in.h>

#include <errno.h>

#include <unistd.h>

*long* getFileSize(FILE\* *file*)

{

    fseek(file, 0L, SEEK\_END);

*long* size = ftell(file);

    rewind(file);

    return size;

}

*void* print\_help()

{

    printf("Commands to use:\n");

    printf("/help - shows existing commangs\n");

    printf("/cls - clear console\n");

    printf("/exit - close the program\n");

}

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

{

*int* port = atoi(argv[1]);

*struct* sockaddr\_in server\_addr;

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

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

    server\_addr.sin\_port = htons(port);

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

    if (sock < 0)

    {

        perror("Socket error");

        return -1;

    }

    if (bind(sock, (*struct* sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

    {

        server\_addr.sin\_port = 0;

        if (bind(sock, (*struct* sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

        {

            perror("Bind error");

            return -1;

        }

    }

*socklen\_t* server\_len = sizeof(server\_addr);

    listen(sock, 100);

    getsockname(sock, (*struct* sockaddr \*)&server\_addr, &server\_len);

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

    if (fork() == 0)

    {

        while (1)

        {

*struct* sockaddr\_in client\_addr;

*socklen\_t* client\_len = sizeof(client\_addr);

*int* newSock = accept(sock, (*struct* sockaddr \*)&client\_addr, &client\_len);

            if (fork() == 0)

            {

                continue;

            }

            while (1)

            {

*char* input[128];

*int* n = read(newSock, input, sizeof(input));

                if (n == 0)

                {

                    close(newSock);

                    exit(0);

                }

                FILE\* file = fopen(input, "r");

                if (!file)

                {

*long* fileSize = -1 ;

                    write(newSock, &fileSize, sizeof(fileSize));

                }

                else

                {

*long* fileSize = getFileSize(file);

                    write(newSock, &fileSize, sizeof(fileSize));

*char* message[fileSize];

                    fread(message, sizeof(*char*), fileSize, file);

                    fclose(file);

                    write(newSock, message, sizeof(message));

                }

            }

        }

    }

    else

    {

        print\_help();

*char* message[128];

        while (1)

        {

            scanf("%s", message);

            if (!strcmp(message, "/help"))

            {

                print\_help();

            }

            else if (!strcmp(message, "/cls"))

            {

                system("clear");

                print\_help();

            }

            else if (!strcmp(message, "/exit"))

            {

                close(sock);

                exit(0);

            }

            else

            {

                printf("You entered a nonexistent command");

            }

        }

    }

}

**Client.c:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <unistd.h>

*void* take\_ip(*char*\* *args*, *char*\* *ip*, *size\_t* *pointer*)

{

    memset(ip, 0, pointer + 1);

    for (*size\_t* i = 0; i < pointer; ++i)

    {

        ip[i] = args[i];

    }

}

*void* print\_help()

{

    printf("Commands to use:\n");

    printf("/help - shows existing commangs\n");

    printf("/cls - clear console\n");

    printf("/exit - close the program\n");

}

*void* recvFile(*int* *sock*, *char*\* *filename*, *long* *fileSize*)

{

*char* buffer[fileSize];

    memset(buffer, 0, fileSize);

    FILE\* file = fopen(filename, "w");

    for(*int* i = 0; i < fileSize; i++)

    {

        printf("%c", buffer[i]);

    }

    printf("\n");

    if (file == NULL)

    {

        printf("Error file opening\n");

        return;

    }

*int* n = 0;

    while (n < fileSize)

    {

        n += read(sock, buffer, sizeof(buffer));

        fprintf(file, "%s", buffer);

    }

    for(*int* i = 0; i < fileSize; i++)

    {

        printf("%c", buffer[i]);

    }

    printf("\n");

    fclose(file);

    printf("File received successfully as %s!\n", filename);

}

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

{

*char*\* args = argv[1];

*char*\* port = strchr(args, ':');

*size\_t* pointer = port - args;

*char* ip[pointer + 1];

    take\_ip(args, ip, pointer);

    printf("%s:%s\n", ip, port + 1);

*struct* sockaddr\_in server\_addr;

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

    if (sock < 0)

    {

        perror("Socket error");

        return -1;

    }

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(atoi(port + 1));

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

*int* con = connect(sock, (*struct* sockaddr \*)&server\_addr, sizeof(server\_addr));

    if (con < 0)

    {

        perror("Connect error");

        return -1;

    }

    printf("You are connected!\n");

    printf("Enter the file name or command shown below\n");

    print\_help();

*char* message[128];

    while (1)

    {

        scanf("%s", message);

        if (!strcmp(message, "/help"))

        {

            print\_help();

        }

        else if (!strcmp(message, "/cls"))

        {

            system("clear");

            print\_help();

        }

        else if (!strcmp(message, "/exit"))

        {

            close(sock);

            exit(0);

        }

        else

        {

*long* fileSize;

            write(sock, message, strlen(message) + 1);

*int* n = read(sock, &fileSize, sizeof(*long*));

            if (n == 0)

            {

                close(sock);

                perror("Connection lost");

                exit(0);

            }

            if (fileSize < 0)

            {

                printf("The file is busy with another program or does not exist\n");

            }

            else

            {

                recvFile(sock, message, fileSize);

            }

        }

    }

}

**Результат выполнения программы:**

Файл из папки server (text.txt) перенесён в папку client.