Лабораторная работа #1, 2 «Программирование сетевых серверов и клиентов» Вариант #5	Выполнил	Ноздренков С.В.
	Группа	ЭВМ-1.Н
	Проверил	Жариков Д. Н.
	Подпись	

Цель работы

Изучение транспортных и прикладных протоколов семейства TCP/IP, структуры сетевых приложений, основных приемов программирования Internet-приложений на основе этих протоколов с использованием программных интерфейсов сокетов BSD UNIX и Windows Sockets 2.

Задание

Разработать две программы: клиент и сервер, позволяющие получать список файлов указанного каталога на компьютере, где функционирует сервер, переименовывать, копировать в другой каталог или удалять указанный файл, запускать программу на выполнение, выполнять перезагрузку компьютера.

Engine.hpp

```
#ifndef ENGINE_HPP
#define ENGINE_HPP
#pragma comment(lib, "WS2_32.lib")
#pragma comment(linker, "/STACK:36777216")

#include <iostream>
#include <string>
#include <cstring>
#include <WinSock2.h>
using namespace std;

#define die(s) { echo(s); return; }
#define dief(s) { echo(s); return false;}

/**
@brief Universal class for working with sockets
*/
class engine_t
{
    string type;

    WSADATA wsaData;
    SOCKET mysock, remsock;
    sockaddr_in sai;
    char buf[2000000];
public:
    /**
    @brief Shows message
    @detailed We can overload this function for another way of log-messaging
    @param s - Message
    */
    void echo(const string &s) { cout << s << endl; }
```

```
/**
@brief Initialisation
@param mtype - Application type. It can be: "client" or "server"
@param ip - ip-address
@param port - port
*/
engine_t(const string &mtype, const string &ip, int port)
{
    type = mtype;

    // Windows sockets initialisation
    if (WSAStartup(MAKEWORD(2, 0), &wsaData))
        die("Can't startup Windows Sockets");
        echo("Windows Sockets started");

    // Creates a socket that is bound to a specific transport service provider
    if ((mysock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET)
        die("Can't create socket");
        echo("Socket Created");

    memset(&sai, 0, sizeof(sockaddr_in));
    sai.sin_family = AF_INET;
    sai.sin_port = htons(port);
    sai.sin_addr.s_addr = type == "server" ? INADDR_ANY : inet_addr(ip.c_str());

    if (type == "server")
    {
        // Associates a local address with a socket
        if (bind(mysock, (sockaddr*)(&sai), sizeof(sai)) == SOCKET_ERROR)
            die("Bind error");
            echo("Bind OK!");

        // Places a socket in a state in which it is listening for an incoming connection
        if (listen(mysock, 1) == SOCKET_ERROR)
            die("Listen error");
            echo("Listen OK!");
    }
}

/**
@brief Connects to client/server for chatting
*/
bool connect()
{
    if (type == "client")
    {
        echo("Connecting...");
        if (::connect(mysock, (sockaddr*)(&sai), sizeof(sai)) == SOCKET_ERROR)
            dief("Connect error!");
            echo("Connection complete!");
    }
    else
    {
        echo("Accepting...");
        if ((remsock = accept(mysock, NULL, NULL)) == INVALID_SOCKET)
            dief("Accept error!");
            echo("Accepted!");
    }
    return true;
}
```

```
    /**
    @brief Sends message
    @param s - message
    */
    bool write(const string &s)
    {
        int len = s.size();
        SOCKET to = type == "server" ? remsock : mysock;
        int f1 = send(to, (char*)(&len), sizeof(len), NULL);
        strcpy(buf, s.c_str());
        int f2 = send(to, buf, len + 1, NULL);
        return f1 == sizeof(int) && f2 == len + 1;
    }

    /**
    @brief Gets message
    @param s - message
    */
    bool read(string &s)
    {
        int len = 0;
        SOCKET from = type == "server" ? remsock : mysock;
        int f1 = recv(from, (char*)(&len), sizeof(len), NULL);
        int f2 = recv(from, buf, len + 1, NULL);
        s = string(buf);
        return f1 == sizeof(int) && f2 == len + 1;
    }

    /**
    @brief Destructor
    @detailed Closes sockets
    */
    ~engine_t()
    {
        closesocket(mysock);
        WSACleanup();
    }
};

#endif
```

nsv_client.cpp

```cpp
#include <iostream>
#include "../common/engine.hpp"
using namespace std;

void hint()
{
    puts("\n=== OPERATIONS ==========================================================");
    puts("info id            -- gets information about account id");
    puts("open               -- opens new account and gets new id for user");
    puts("close id           -- tries to close account with id");
    puts("add id amount      -- deposits money to account id");
    puts("get id amount      -- tries to take money from account id");
    puts("mov src dst amount -- tries to move money from account src to account dst");
    puts("==========================================================================\n");
}

int main()
{
    puts("CLIENT-BANK!");
    engine_t engine("client", "127.0.0.1", 5001);
    engine.connect();

    hint();

    while (true)
    {
        printf("> ");

        string query, ans;
        getline(cin, query);

        engine.write(query);
        engine.read(ans);

        puts("\n======= RESULT ====================================");
        puts(ans.c_str());
        puts("==================================================\n");
    }

    cout << "GOOD BYE!" << endl;
    return 0;
}
```

nsv_server.cpp

```
#include "../common/engine.hpp"
#include <sstream>
#include <fstream>
#include <cstdio>
#include <Windows.h>
#include "dirent.h"
using namespace std;

/**
@brief Simple file manager class
*/
class fm_t
{
    engine_t *engine;

public:
    /**
    @brief Initialisation
    */
    fm_t()
    {
        engine = new engine_t("server", "127.0.0.1", 5001);
        engine->connect();
    }

    /**
    @brief Starts main process
    */
    void start()
    {
        while (true)
        {
            string s;
            if (engine->read(s))
                process(s);
            else
            {
                cout << "Connection closed!" << endl;
                engine->connect();
            }
        }
    }

    /**
    @brief Switch how to process request req
    @param req - clients request
    */
    void process(const string &req)
    {
        istringstream is(req);
        string type; is >> type;

        if (type == "show")
        {
            string dir; is >> dir;
            show(dir);
        }
        else if (type == "rename")
        {
            string file, name;
            is >> file >> name;
            rename(file, name);
        }
```

```
    else if (type == "copy")
    {
        string from, to;
        is >> from >> to;
        copy(from, to);
    }
    else if (type == "remove")
    {
        string file; is >> file;
        remove(file);
    }
    else if (type == "exec")
    {
        string file; is >> file;
        exec(file);
    }
    else if (type == "reboot")
    {
        reboot();
    }
    else
        engine->write("Invalid request!");
}

/**
@brief Shows files in directory s
*/
void show(const string &s)
{
    ostringstream os;
    os << "FILES:" << endl;

    DIR *dir;
    struct dirent *ent;
    if (dir = opendir(s.c_str()))
    {
        while (ent = readdir(dir))
            os << string(ent->d_name) << endl;
        closedir(dir);
    }
    else
        os << "ERROR!" << endl;

    engine->write(os.str());
}

/**
@brief Renames file
*/
void rename(const string &file, const string &new_name)
{
    ostringstream os;
    os << "RENAME" << endl;
    os << "file: " << file << endl;
    os << "new name: " << new_name << endl;
    if (::rename(file.c_str(), new_name.c_str()) == 0)
        os << "SUCCESS!";
    else
        os << "FAIL! :(";
    engine->write(os.str());
}
```

```
    /**
    @brief Copies file
    */
    void copy(const string &from, const string &to)
    {
        ostringstream os;
        os << "COPY" << endl;
        os << "from: " << from << endl;
        os << "to: " << to << endl;

        ifstream  src(from, std::ios::binary);
        ofstream  dst(to,   std::ios::binary);
        dst << src.rdbuf();

        os << "Finish" << endl;
        engine->write(os.str());
    }

    /**
    @brief Remove file
    */
    void remove(const string &file)
    {
        ostringstream os;
        os << "REMOVE" << endl;
        os << "file: " << file << endl;

        if (::remove(file.c_str()) == 0)
            os << "SUCCESS!";
        else
            os << "FAIL! :(";

        engine->write(os.str());
    }

    /**
    @brief Executes file
    */
    void exec(const string &file)
    {
        ostringstream os;
        os << "EXEC" << endl;
        os << "file: " << file << endl;

        if (WinExec(file.c_str(), 1) > 31)
            os << "SUCCESS!";
        else
            os << "FAIL! :(";

        engine->write(os.str());
    }

    /**
    @brief Reboots comp
    */
    void reboot()
    {
        engine->write("OK, let's reboot!");
        system("shutdown -r -t 0");
    }
};
```

```cpp
int main()
{
    cout << "SERVER-FILE-MANAGER" << endl;

    fm_t manager;
    manager.start();

    cerr << "Good bye!" << endl;
    return 0;
}
```

Результат работы программы