МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №2

По дисциплине «Аппаратное и программное обеспечение сетей»

Тема: «Организация UDP– сервера и UDP – клиента»

Выполнил:

студент 3 курса

группы ИИ-23(2)

Романюк А. П.

Проверил:

Степанчук В.И.

Брест 2024

Цель работы: 1) изучить основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрести навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов UDP.

Вариант 9.

|  |  |  |
| --- | --- | --- |
| Вариант задания для бригады студентов | Номер задания для реализации TCP (UDP) сервера (см. табл. 2). | Номер задания для реализации TCP (UDP) клиента (см. табл. 3). |
| 9 | 1 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер задания | | Функция, выполняемая TCP-сервером | | | |
| 1 | | Отсылка клиенту содержимого текстового файла<имя файла>в случае приема сервером в потокесимволов командыloadfname.txt, <имя файла>–имя некоторого текстового файла, находящегосяв каталоге сервера. В случае, если запрашиваемый файл отсутствует в каталоге сервера, сервердолжен отослать сообщение об этом и разорвать соединение. | | | |
| Вариант | Ввод символов с отсылкой введенной строки по нажатию на клавишу | | Ведение файла протокола событий, включающих\*:  1) время начала и окончания соединения;  2) передаваемую серверу строку и время передачи строки;  3) принимаемую от сервера строку и время приема строки. | Возможность разрыва соединения при помощи команды:  **disconnect <адрес> <порт>** | 1) Задание в программе клиента специальной команды и параметров: подключения к серверу:  **сonnect**  **<адрес> <порт>**  2) Автоматическое подключение к серверу с заданным по умолчанию адресом при запуске клиента |
| 9 | PgDn | | 1), 2), 3) | - | 2) |

Реализация TCP-сервера:

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <string>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <algorithm> // для std::remove

#pragma comment(lib, "Ws2\_32.lib")

const int PORT = 8080;

const int BUFFER\_SIZE = 1024;

void handle\_client(SOCKET server\_socket, sockaddr\_in& client\_addr, int client\_len) {

char buffer[BUFFER\_SIZE];

std::string request;

// Получаем данные от клиента через recvfrom

int bytes\_received = recvfrom(server\_socket, buffer, BUFFER\_SIZE - 1, 0, (struct sockaddr\*)&client\_addr, &client\_len);

if (bytes\_received == SOCKET\_ERROR) {

std::cerr << "Ошибка при получении данных от клиента." << std::endl;

return;

}

buffer[bytes\_received] = '\0';

request = buffer;

std::cout << "Запрос от клиента: " << request << std::endl;

if (request.length() >= 5 && request.substr(0, 4) == "load") {

std::string filename = request.substr(5);

filename.erase(std::remove(filename.begin(), filename.end(), '\n'), filename.end());

filename.erase(std::remove(filename.begin(), filename.end(), '\r'), filename.end());

std::cout << "Запрошенный файл: " << filename << std::endl;

std::ifstream file(filename);

if (file.is\_open()) {

std::string line;

while (std::getline(file, line)) {

line += "\n";

// Отправляем каждую строку клиенту через sendto

sendto(server\_socket, line.c\_str(), line.size(), 0, (struct sockaddr\*)&client\_addr, client\_len);

}

file.close();

}

else {

std::string error\_message = "Файл не найден\n";

sendto(server\_socket, error\_message.c\_str(), error\_message.size(), 0, (struct sockaddr\*)&client\_addr, client\_len);

}

}

else {

std::string error\_message = "Неверный формат запроса\n";

sendto(server\_socket, error\_message.c\_str(), error\_message.size(), 0, (struct sockaddr\*)&client\_addr, client\_len);

}

}

int main() {

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "Ошибка инициализации Winsock." << std::endl;

return 1;

}

SOCKET server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); // Создаем UDP сокет

if (server\_socket == INVALID\_SOCKET) {

std::cerr << "Ошибка создания сокета." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in server\_addr;

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

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

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(server\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {

std::cerr << "Ошибка привязки сокета." << std::endl;

closesocket(server\_socket);

WSACleanup();

return 1;

}

std::cout << "UDP сервер запущен. Ожидание запросов..." << std::endl;

while (true) {

sockaddr\_in client\_addr;

int client\_len = sizeof(client\_addr);

// Обрабатываем запрос от клиента

handle\_client(server\_socket, client\_addr, client\_len);

}

closesocket(server\_socket);

WSACleanup();

return 0;

}

Реализация TCP-клиента:

#include <iostream>

#include <sstream>

#include <fstream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <conio.h>

#include <chrono>

#include <ctime>

#pragma comment(lib, "ws2\_32.lib")

#define BUFFER\_SIZE 1024

std::string custom\_input() {

std::string input\_text;

char ch;

bool pgdn\_pressed = false;

while (true) {

ch = \_getch();

if (ch == 0x00 || ch == 0xE0) {

ch = \_getch();

if (ch == 0x51) {

pgdn\_pressed = true;

break;

}

}

else if (ch == '@') {

break;

}

else if (ch == 0x08) {

if (!input\_text.empty()) {

input\_text.pop\_back();

std::cout << "\b \b" << std::flush;

}

}

else {

input\_text += ch;

std::cout << ch << std::flush;

}

}

if (pgdn\_pressed) {

std::cout << "\nНажата клавиша Page Down.\n";

}

std::cout << std::endl;

return input\_text;

}

std::string int\_to\_string(int number) {

std::ostringstream oss;

oss << number;

return oss.str();

}

void log\_event(const std::string& message) {

std::ofstream log\_file("protocol\_log.txt", std::ios::app);

if (log\_file.is\_open()) {

auto now = std::chrono::system\_clock::now();

std::time\_t now\_time = std::chrono::system\_clock::to\_time\_t(now);

char time\_str[26];

ctime\_s(time\_str, sizeof(time\_str), &now\_time);

log\_file << time\_str << message << std::endl;

}

}

void send\_command(const std::string& server\_ip, int port, const std::string& command) {

WSADATA wsaData;

SOCKET SendSocket = INVALID\_SOCKET;

struct addrinfo\* result = nullptr,

\* ptr = nullptr,

hints;

auto start\_time = std::chrono::system\_clock::now();

log\_event("Начало отправки");

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "Ошибка WSAStartup: " << WSAGetLastError() << std::endl;

return;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_DGRAM;

hints.ai\_protocol = IPPROTO\_UDP;

if (getaddrinfo(server\_ip.c\_str(), int\_to\_string(port).c\_str(), &hints, &result) != 0) {

std::cerr << "Ошибка getaddrinfo: " << WSAGetLastError() << std::endl;

WSACleanup();

return;

}

SendSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (SendSocket == INVALID\_SOCKET) {

std::cerr << "Ошибка создания сокета: " << WSAGetLastError() << std::endl;

freeaddrinfo(result);

WSACleanup();

return;

}

std::string message = command + "\n";

auto send\_time = std::chrono::system\_clock::now();

std::time\_t send\_time\_t = std::chrono::system\_clock::to\_time\_t(send\_time);

char send\_time\_str[26];

ctime\_s(send\_time\_str, sizeof(send\_time\_str), &send\_time\_t);

log\_event("Отправка команды: " + command + " в " + send\_time\_str);

int bytes\_sent = sendto(SendSocket, message.c\_str(), (int)message.length(), 0, result->ai\_addr, (int)result->ai\_addrlen);

if (bytes\_sent == SOCKET\_ERROR) {

std::cerr << "Ошибка отправки: " << WSAGetLastError() << std::endl;

closesocket(SendSocket);

WSACleanup();

return;

}

char recvbuf[BUFFER\_SIZE];

sockaddr\_in recv\_addr;

int recv\_addr\_len = sizeof(recv\_addr);

int bytes\_received = recvfrom(SendSocket, recvbuf, BUFFER\_SIZE - 1, 0, (struct sockaddr\*)&recv\_addr, &recv\_addr\_len);

if (bytes\_received > 0) {

recvbuf[bytes\_received] = '\0';

auto receive\_time = std::chrono::system\_clock::now();

std::time\_t receive\_time\_t = std::chrono::system\_clock::to\_time\_t(receive\_time);

char receive\_time\_str[26];

ctime\_s(receive\_time\_str, sizeof(receive\_time\_str), &receive\_time\_t);

log\_event("Получено от сервера: " + std::string(recvbuf) + " в " + receive\_time\_str);

std::cout << "Получено от сервера: " << recvbuf << std::endl;

}

else if (bytes\_received == 0) {

std::cout << "Соединение закрыто" << std::endl;

}

else {

std::cerr << "Ошибка получения: " << WSAGetLastError() << std::endl;

}

closesocket(SendSocket);

WSACleanup();

auto end\_time = std::chrono::system\_clock::now();

log\_event("Окончание отправки");

}

int main() {

std::string server\_ip = "127.0.0.1";

int port = 8080;

std::cout << "Нажмите Page Down для отправки команды.\n";

while (true) {

std::cout << "\nВведите команду для отправки на сервер: ";

std::string command = custom\_input();

send\_command(server\_ip, port, command);

}

return 0;

}

**Вывод:** 1) изучил основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрёл навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов TCP.