Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики

и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе №7

на тему

«Средства обмена данными (Windows). Изучение и использованием средств обмена данными и совместного доступа.»

Выполнил:

студент гр. 153504

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты](#_1fob9te) 5

[Вывод](#_4q3p755gzys6) 6

[Список использованных источников](#_4q3p755gzys6) 7

[Приложение А (обязательное) Листинг кода](#_4q3p755gzys6) 8

**1 ЦЕЛИ РАБОТЫ**

1 Изучить средства обмена данными и совместного доступа.

2 Создать приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сокет ­ это точка, через которую программы общаются в сети. Протокол определяет правила, по которым данные обмениваются между сокетами. Например, протокол TCP обеспечивает надежную передачу данных, а протокол UDP - более быструю, но менее надежную передачу.

Работа с сокетами в *Win Api:*

1 Инициализация и завершение работы с *Winsock. Winsock* (*Windows* *Sockets*) представляет собой API для разработки сетевых приложений в среде Windows. Прежде чем использовать функции *Winsock*, необходимо инициализировать библиотеку с помощью *WSAStartup* и завершить ее с помощью *WSACleanup*.

2 Создание сокетов. Сокет представляет собой конечную точку взаимодействия в сети. Функция socket создает сокет, указывая семейство адресов (например, *AF\_INET* для *IPv4*), тип сокета (например, *SOCK\_STREAM* для потокового сокета) и протокол (обычно 0 для выбора подходящего протокола).

3 Привязка сокетов к адресу и порту. Прежде чем начать слушать входящие соединения, сокет должен быть привязан к определенному адресу и порту с использованием структуры *sockaddr\_in*.

4 Ожидание соединений и установка соединений. Функция *listen* устанавливает сокет в режим ожидания входящих соединений, а *accept* блокируется, ожидая, пока клиент не попытается подключиться. Когда клиентское подключение получено, accept возвращает новый сокет для взаимодействия с клиентом.

5 Чтение и отправка данных по сокету. Для обмена данными между сервером и клиентом используются функции *recv* и *send*. Функция recv читает данные из сокета в буфер, а функция send отправляет данные из буфера по сокету.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

На рисунке 3.1 продемонстрирована результат работы программы, где видно окна обоих клиентов и сообщения которыми они обмениваются.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.1 ­ Результат работы программы

.

**ВЫВОД**

В результате было разработано приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

Для этого был изучен теоретический материал и средства Win32 Api, используемые для этих целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Функция socket (winsock2.h), документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winsock2/nf-winsock2-socket>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

**Server.cpp**

#include <iostream>

#include <winsock2.h>

#include <vector>

#include <thread>

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

std::vector<SOCKET> clients;

void HandleClient(SOCKET clientSocket) {

char buffer[1024];

int bytesReceived;

while (true) {

bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);

if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {

std::cerr << "Client disconnected\n";

closesocket(clientSocket);

break;

}

buffer[bytesReceived] = '\0';

// Отправляем сообщение всем остальным клиентам

for (SOCKET otherClient : clients) {

if (otherClient != clientSocket) {

send(otherClient, buffer, bytesReceived, 0);

}

}

}

}

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock\n";

return 1;

}

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create socket\n";

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

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

serverAddr.sin\_port = htons(12345);

if (bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Bind failed with error\n";

closesocket(serverSocket);

WSACleanup();

return 1;

}

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cerr << "Listen failed with error\n";

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Server listening on port 12345...\n";

while (true) {

SOCKET clientSocket = accept(serverSocket, NULL, NULL);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Accept failed with error\n";

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Client connected\n";

clients.push\_back(clientSocket);

std::thread(HandleClient, clientSocket).detach();

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

**Client.cpp**

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h> // Для InetPton

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

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock\n";

return 1;

}

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create socket\n";

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

if (InetPton(AF\_INET, TEXT("127.0.0.1"), &(serverAddr.sin\_addr)) != 1) {

std::cerr << "Invalid address/Address not supported\n";

closesocket(clientSocket);

WSACleanup();

return 1;

}

serverAddr.sin\_port = htons(12345);

if (connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Connect failed with error\n";

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::cout << "Connected to server\n";

char message[1024];

while (true) {

std::cout << "Enter message: ";

std::cin.getline(message, sizeof(message));

send(clientSocket, message, strlen(message), 0);

int bytesReceived = recv(clientSocket, message, sizeof(message), 0);

if (bytesReceived == SOCKET\_ERROR) {

std::cerr << "Recv failed with error\n";

closesocket(clientSocket);

WSACleanup();

return 1;

}

message[bytesReceived] = '\0';

std::cout << "Client2 response: " << message << std::endl;

}

closesocket(clientSocket);

WSACleanup();

return 0;

}