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

ОТЧЁТ

к лабораторной работе №5

на тему

**. ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ.**

Выполнил: студент гр.253505 Авдошко И.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 8](#_Toc179742264)

[Список использованных источников 9](#_Toc179742265)

[Приложение А (обязательное) 10](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение основ построения и функционирования сетей, стеков протоколов и программных интерфейсов для организации взаимодействия между процессами в сетях. В процессе выполнения лабораторной работы изучаются подходы к созданию сетевых приложений, реализующих взаимодействие через протоколы прикладного уровня с использованием сокетов.

В качестве задачи необходимо разработать программу на языке C++ для интерактивного взаимодействия пользователей посредством сетевых сокетов. Программа должна представлять собой упрощенный чат для нескольких пользователей.

Для достижения этой цели требуется создать серверную часть, которая будет:

– принимать подключения от клиентов;

– временно хранить сообщения;

– передавать сообщения адресно одному или нескольким клиентам;

– поддерживать список активных клиентов.

Также потребуется создать клиентскую часть, которая будет:

– устанавливать соединение с сервером;

– позволять пользователю вводить сообщения и отправлять их на сервер;

– принимать и отображать сообщения от других пользователей, обеспечивая интерактивность общения.

В рамках работы необходимо разработать структуру передаваемых сообщений, включающую адрес отправителя и тело сообщения. Также требуется определить протокол прикладного уровня, регламентирующий порядок взаимодействия между клиентом и сервером, включая форматирование и обработку сообщений.

Архитектура системы выбрана централизованная, что подразумевает наличие выделенного сервера, отвечающего за маршрутизацию сообщений между клиентами. Транспортные протоколы: TCP или UDP.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений. [1]

Процесс – создается путем вызова функции CreateProcess. Во время выполнения процесса для нее устанавливается состояние без знака, а при завершении процесса – значение signaled.

Thread – создается при создании нового потока путем вызова функции CreateProcess, CreateThread или CreateRemoteThread. В то время как поток выполняется, для нее устанавливается состояние без знака, а при завершении потока – значение signaled. [2]

Клиент – это приложение или процесс, который запрашивает услугу у другого приложения или процесса. Сервер ­– это приложение или процесс, который отвечает на запрос клиента. Многие приложения в зависимости от ситуации могут выступать и в качестве клиента, и в качестве сервера.

Windows Sockets — это независимый от протокола интерфейс. Он использует преимущества коммуникационных возможностей базовых протоколов. В Windows Sockets 2 дескриптор сокета можно использовать в качестве дескриптора файла со стандартными функциями ввода-вывода.

Сокеты Windows основаны на сокетах, впервые появившихся в дистрибутиве Berkeley Software Distribution (BSD). Приложение, использующее сокеты Windows, может взаимодействовать с другими реализациями сокетов в системах других типов. Однако не все поставщики транспортных услуг поддерживают все доступные опции.

Ключевой момент: Windows Sockets — это независимый от протокола интерфейс, способный поддерживать существующие и новые сетевые возможности. [3]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Разработанная программа представляет собой упрощенный чат для нескольких пользователей, позволяющий взаимодействовать клиентам через сервер. Программа реализует клиент-серверную архитектуру с использованием сокетов для обмена сообщениями в реальном времени. Основные функции программы включают в себя создание серверного и клиентского процессов, обработку сообщений, а также управление подключениями.

Серверная часть программы отвечает за прием подключений от клиентов. Для этого используется функция socket(), которая создает сокет для приема соединений. Далее сервер выполняет привязку сокета к конкретному IP-адресу и порту с помощью функции bind(). Для ожидания подключений клиентов применяется функция listen(), которая переводит сокет в состояние ожидания.

Когда клиент подключается, сервер использует функцию accept() для принятия соединения. После этого сервер может получать сообщения от клиента через функцию recv(), которая читает данные из сокета. Полученные сообщения обрабатываются и передаются другим клиентам с использованием функции send(). На рисунке 3.1 изображен запущенный сервер.

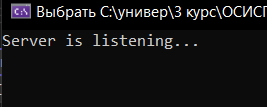


Рисунок 3.1 – Запуск сервера

Клиентская часть программы устанавливает соединение с сервером, используя ту же функцию socket() для создания сокета. Затем клиент подключается к серверу с помощью функции connect(), передавая адрес и порт сервера. Для отправки сообщений на сервер используется функция send(), а для приема ответов от сервера – функция recv(). Сервер фиксирует подключение клиента, как показано на рисунке 3.2.

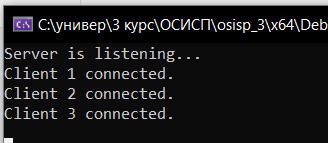


Рисунок 3.2 –Подключение клиентов к серверу

В то же время клиент получает доступ к окну с информацией о том, что он был успешно подключен к серверу и возможностью отправлять сообщения. Интерфейс окна клиента изображен на рисунке 3.3.

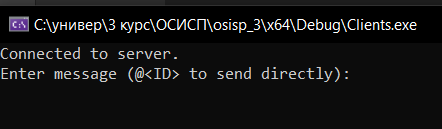


Рисунок 3.3 – Запуск окна клиента

Клиент может отправлять широковещательные сообщения, которые будут видны всем клиентам данного сервера. Пример такого ввода представлен на рисунке 3.4.

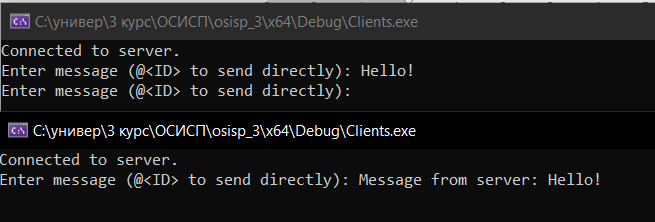


Рисунок 3.4 – Демонстрация обмена сообщениями

Также у клиентов имеется возможность отправлять адресные сообщения, которые будут доступны только для того пользователя, которому они были предназначены. Отправить такое сообщение можно, введя перед текстом сообщения *@<ID>*, где *ID* – идентификатор клиента, которому предназначено сообщение. Пример отправки адресного сообщения приведен на рисунке 3.5.

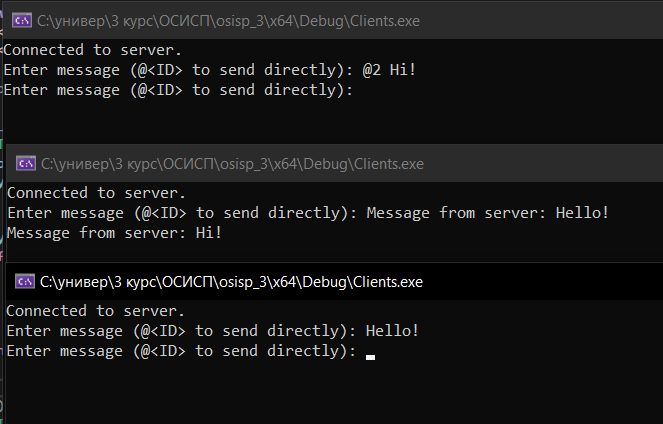


Рисунок 3.5 – Обмен личными сообщениями

В данной программе были использованы и другие функции WinAPI для управления потоками и синхронизацией, такие как CreateThread(), которая создает новый поток для обработки многопользовательских подключений, и WaitForSingleObject(), позволяющая ожидать завершения потока.

В результате работы программы обеспечивается интерактивное взаимодействие пользователей в режиме реального времени, что позволяет обмениваться сообщениями между клиентами через сервер, демонстрируя основные принципы межпроцессного взаимодействия и сетевого общения.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была разработана программа для интерактивного взаимодействия пользователей в виде упрощенного чата для нескольких клиентов. Основной целью работы было изучение подходов, системных объектов и функций, необходимых для передачи данных между взаимодействующими процессами, а также исследование типичных проблем, возникающих при организации такого взаимодействия.

Программа успешно реализует клиент-серверную архитектуру, позволяя пользователям обмениваться текстовыми сообщениями в реальном времени. Серверная часть обеспечивает прием подключений от клиентов, обработку входящих сообщений и их адресную передачу другим клиентам. Клиенты могут вводить текстовые сообщения и отправлять их на сервер, а также получать сообщения от других пользователей.

В процессе разработки программы были использованы функции WinAPI для создания сокетов, управления подключениями и обработки сообщений, такие как socket(), bind(), listen(), accept(), send(), и recv(). Эти функции позволили организовать эффективное взаимодействие между клиентами и сервером.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.

[2] Microsoft. "Объекты синхронизации" [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

[3] Microsoft. "Interprocess communications" [Электронный ресурс]. – Режим доступа https://learn.microsoft.com/en-  
us/windows/win32/ipc/interprocess-communications.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <winsock2.h>

#include <process.h> // для \_beginthreadex

#include <map>

#include <string>

#include <vector>

#include <mutex>

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

std::map<int, SOCKET> clients; // Хранение клиентов с их ID

std::mutex clientsMutex;

struct Message {

int senderID;

int receiverID; // -1 для широковещательного сообщения

std::string text;

};

// Временное хранилище сообщений

std::vector<Message> messageQueue;

std::mutex messageQueueMutex;

unsigned \_\_stdcall clientHandler(void\* clientData) {

auto clientPair = \*reinterpret\_cast<std::pair<int, SOCKET>\*>(clientData);

int clientID = clientPair.first;

SOCKET clientSocket = clientPair.second;

delete reinterpret\_cast<std::pair<int, SOCKET>\*>(clientData); // очистка

char buffer[1024];

int bytesReceived;

while (true) {

bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesReceived <= 0) {

std::cerr << "Client " << clientID << " disconnected." << std::endl;

closesocket(clientSocket);

std::lock\_guard<std::mutex> lock(clientsMutex);

clients.erase(clientID);

return 0;

}

buffer[bytesReceived] = '\0';

std::string messageText(buffer);

// Обработка сообщения: предполагаем, что первые символы - ID получателя

int receiverID = -1; // -1 для широковещательной передачи

if (messageText[0] == '@') {

receiverID = std::stoi(messageText.substr(1, messageText.find(' ')));

messageText = messageText.substr(messageText.find(' ') + 1);

}

// Добавление сообщения в очередь

{

std::lock\_guard<std::mutex> lock(messageQueueMutex);

messageQueue.push\_back({ clientID, receiverID, messageText });

}

}

return 0;

}

unsigned \_\_stdcall messageDispatcher(void\* param) {

while (true) {

std::lock\_guard<std::mutex> lock(messageQueueMutex);

for (const auto& msg : messageQueue) {

std::lock\_guard<std::mutex> lock(clientsMutex);

if (msg.receiverID == -1) {

// Широковещательная рассылка всем клиентам

for (const auto& client : clients) {

if (client.first != msg.senderID) {

send(client.second, msg.text.c\_str(), msg.text.length(), 0);

}

}

}

else {

// Отправка конкретному клиенту

auto it = clients.find(msg.receiverID);

if (it != clients.end()) {

send(it->second, msg.text.c\_str(), msg.text.length(), 0);

}

}

}

messageQueue.clear();

Sleep(100); // ожидание перед следующим циклом обработки

}

return 0;

}

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock." << std::endl;

return 1;

}

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

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create server socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

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

serverAddr.sin\_port = htons(8080);

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

std::cerr << "Bind failed." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

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

std::cerr << "Listen failed." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Server is listening..." << std::endl;

HANDLE dispatcherThread = (HANDLE)\_beginthreadex(nullptr, 0, messageDispatcher, nullptr, 0, nullptr);

int clientID = 1;

while (true) {

sockaddr\_in clientAddr;

int clientAddrSize = sizeof(clientAddr);

SOCKET clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrSize);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Accept failed." << std::endl;

continue;

}

{

std::lock\_guard<std::mutex> lock(clientsMutex);

clients[clientID] = clientSocket;

}

std::cout << "Client " << clientID << " connected." << std::endl;

auto clientData = new std::pair<int, SOCKET>(clientID, clientSocket);

HANDLE hThread = (HANDLE)\_beginthreadex(nullptr, 0, clientHandler, clientData, 0, nullptr);

clientID++;

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

#include <iostream>

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

#include <process.h> // для \_beginthreadex

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

unsigned \_\_stdcall receiveMessages(void\* clientSocketPtr) {

SOCKET clientSocket = \*(SOCKET\*)clientSocketPtr;

char buffer[1024];

int bytesReceived;

while (true) {

bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesReceived > 0) {

buffer[bytesReceived] = '\0';

std::cout << "Message from server: " << buffer << std::endl;

}

}

return 0;

}

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock." << std::endl;

return 1;

}

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

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create client socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serverAddr.sin\_port = htons(8080);

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

std::cerr << "Connection failed." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::cout << "Connected to server." << std::endl;

HANDLE receiveThread = (HANDLE)\_beginthreadex(nullptr, 0, receiveMessages, &clientSocket, 0, nullptr);

char message[1024];

while (true) {

std::cout << "Enter message (@<ID> to send directly): ";

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

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

}

closesocket(clientSocket);

WSACleanup();

return 0;

}