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 №8

на тему

**ИНТЕРФЕЙС СОКЕТОВ И ОСНОВЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ (WINDOWS).**

**ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ЧЕРЕЗ**

**СЕТЬ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА**

**СОКЕТОВ. РЕАЛИЗАЦИЯ СЕТЕВЫХ ПРОТОКОЛОВ: СОБСТВЕННЫХ ИЛИ СТАНДАРТНЫХ**

Студент А. С. Сорокин

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы заключается в изучении основ сетевого программирования, сетевых протоколов, приобретении практических навыков использования основных функций для работы с сокетами в *Windows*, а также закреплении полученных знаний. Для достижения цели будет создано оконное приложение, которое использует механизм именованных каналов (Named Pipes) для обмена данными между клиентом и сервером.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства обмена данными в операционной системе Windows предоставляют различные механизмы для передачи информации между разными процессами или компонентами системы. Один из таких механизмов - именованные каналы (Named Pipes).

Именованные каналы являются односторонними или двусторонними связями для обмена данными между клиентским и серверным приложениями. Клиент и сервер могут быть запущены в разных процессах или на разных компьютерах в сети. Именованные каналы предоставляют надежный и эффективный способ обмена данными между этими процессами.

Сетевое программирование – это область программирования, связанная с разработкой приложений, способных взаимодействовать между собой через компьютерные сети. Эта область включает в себя создание программ, которые могут обмениваться данными и ресурсами с удаленными системами через различные сетевые протоколы.

*UDP protocol* – протокол, обеспечивающий передачу данных (датаграмм) без предварительного создания соединения между хостами. При отправке датаграмм нет уверенности в существовании получателя и его готовности к обмену.

Протокол передачи данных *TCP –* протокол, обеспечивающий надежную доставку пакетов данных, он обеспечивает установку соединения между двумя хостами методом «рукопожатия», после которого может осуществляться обмен данными.

Одним из ключевых инструментов для реализации сетевого взаимодействия являются сокеты.

Сокеты – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет – абстрактный объект, представляющий конечную точку соединения.

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы. Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции.

Использование именованных каналов в оконном приложении позволяет обмениваться данными между клиентским и серверным компонентами, например, между графическим интерфейсом и фоновым процессом. Это особенно полезно при разработке многопоточных или распределенных приложений, где требуется совместный доступ к данным или синхронизация действий между разными частями системы.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной работы было создано серверное окно, которое создает именованный канал, ожидает подключения клиента. (рисунок 1).

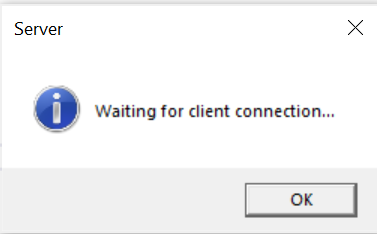


Рисунок 1 – Окно приложения

Пользователь может ввести и отправить сообщение серверу. (рисунок 2).

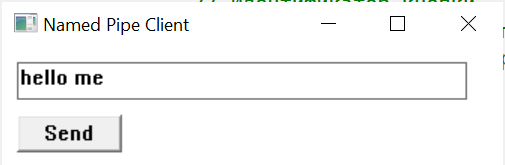


Рисунок 2– Ввод сообщения

Сервер принимает данные от клиента и выводит их в диалоговом окне. (рисунок 3).

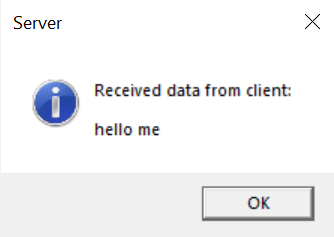


Рисунок 3– Вывод сообщение на сервер

# ЗАКЛЮЧЕНИЕ

Таким образом, в ходе данной лабораторной работы было проведено изучение и применение на практике средства обмена данными (Windows). Изучение и использованием средств обмена данными и совместного доступа. Для достижения цели будет создано оконное приложение, которое использует механизм именованных каналов (Named Pipes) для обмена данными между клиентом и сервером.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/>.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab\_7.cpp

#include <Windows.h>

#include <iostream>

#define BUFFER\_SIZE 1024

#define PIPE\_NAME L"\\\\.\\pipe\\MyNamedPipe"

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE:

{

// Создание именованного канала (в серверном окне)

HANDLE hPipe = CreateNamedPipe(

PIPE\_NAME, // Имя канала

PIPE\_ACCESS\_DUPLEX, // Дуплексный доступ

PIPE\_TYPE\_MESSAGE | // Режим сообщений

PIPE\_READMODE\_MESSAGE | // Режим чтения сообщений

PIPE\_WAIT, // Режим блокировки

PIPE\_UNLIMITED\_INSTANCES, // Количество экземпляров

BUFFER\_SIZE, // Размер входного/выходного буфера

BUFFER\_SIZE, // Размер выходного/входного буфера

0, // Время ожидания соединения (0 - неограниченное время)

NULL

);

if (hPipe == INVALID\_HANDLE\_VALUE) {

MessageBox(hwnd, L"Failed to create named pipe.", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

MessageBox(hwnd, L"Waiting for client connection...", L"Server", MB\_OK | MB\_ICONINFORMATION);

// Ожидание подключения клиента

BOOL isConnected = ConnectNamedPipe(hPipe, NULL);

if (!isConnected) {

MessageBox(hwnd, L"Failed to connect to client.", L"Error", MB\_OK | MB\_ICONERROR);

CloseHandle(hPipe);

return -1;

}

MessageBox(hwnd, L"Client connected.", L"Server", MB\_OK | MB\_ICONINFORMATION);

// Чтение данных от клиента

char buffer[BUFFER\_SIZE];

DWORD bytesRead;

BOOL success = ReadFile(hPipe, buffer, BUFFER\_SIZE, &bytesRead, NULL);

if (!success) {

MessageBox(hwnd, L"Failed to read data from client.", L"Error", MB\_OK | MB\_ICONERROR);

}

else {

buffer[bytesRead] = '\0';

std::string receivedData(buffer);

std::wstring message = L"Received data from client:\n\n" + std::wstring(receivedData.begin(), receivedData.end());

MessageBox(hwnd, message.c\_str(), L"Server", MB\_OK | MB\_ICONINFORMATION);

}

// Закрытие канала

FlushFileBuffers(hPipe);

DisconnectNamedPipe(hPipe);

CloseHandle(hPipe);

DestroyWindow(hwnd);

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

const wchar\_t CLASS\_NAME[] = L"MyWindowClass";

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"Named Pipe Server",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

500, 300,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

MessageBox(NULL, L"Failed to create window.", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

ShowWindow(hwnd, nCmdShow);

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

Листинг 2 – Файл Lab\_7\_client.cpp

#include <Windows.h>

#include <iostream>

#define BUFFER\_SIZE 1024

#define PIPE\_NAME L"\\\\.\\pipe\\MyNamedPipe"

HWND hwndEdit;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE:

{

hwndEdit = CreateWindowEx(

0,

L"EDIT",

NULL,

WS\_CHILD | WS\_VISIBLE |

WS\_BORDER | ES\_AUTOHSCROLL,

10, 10,

300, 25,

hwnd,

NULL,

NULL,

NULL

);

CreateWindowEx(

0,

L"BUTTON",

L"Send",

WS\_CHILD | WS\_VISIBLE,

10, 45,

70, 25,

hwnd,

(HMENU)1,

NULL,

NULL

);

break;

}

case WM\_COMMAND:

{

if (LOWORD(wParam) == 1)

{

int textLength = GetWindowTextLength(hwndEdit);

if (textLength > 0)

{

int bufferSize = (textLength + 1) \* sizeof(wchar\_t);

wchar\_t\* buffer = new wchar\_t[bufferSize];

GetWindowText(hwndEdit, buffer, textLength + 1);

// Преобразование текста в формат const char\*

int messageSize = WideCharToMultiByte(CP\_UTF8, 0, buffer, -1, NULL, 0, NULL, NULL);

char\* message = new char[messageSize];

WideCharToMultiByte(CP\_UTF8, 0, buffer, -1, message, messageSize, NULL, NULL);

// Создание и открытие именованного канала

HANDLE hPipe = CreateFile(

PIPE\_NAME, // Имя канала

GENERIC\_READ | GENERIC\_WRITE, // Режим доступа

0, // Нет совместного доступа

NULL, // Атрибуты безопасности по умолчанию

OPEN\_EXISTING, // Открытие существующего канала

0, // Дополнительные флаги и атрибуты

NULL // Использование шаблона файла

);

if (hPipe != INVALID\_HANDLE\_VALUE)

{

// Отправка данных на сервер

DWORD bytesWritten;

BOOL success = WriteFile(hPipe, message, messageSize, &bytesWritten, NULL);

if (success)

{

MessageBox(hwnd, L"Data sent to server.", L"Client", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(hwnd, L"Failed to send data to server.", L"Error", MB\_OK | MB\_ICONERROR);

}

// Закрытие канала

CloseHandle(hPipe);

}

else

{

MessageBox(hwnd, L"Failed to connect to server.", L"Error", MB\_OK | MB\_ICONERROR);

}

// Очистка буферов

delete[] buffer;

delete[] message;

}

else

{

MessageBox(hwnd, L"Please enter some text.", L"Client", MB\_OK | MB\_ICONWARNING);

}

}

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

const wchar\_t CLASS\_NAME[] = L"MyWindowClass";

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"Named Pipe Client",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

350, 120,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

MessageBox(NULL, L"Failed to create window.", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

ShowWindow(hwnd, nCmdShow);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}