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 №3

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ: ОБМЕН ДАННЫМИ.**

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 6](#_Toc179742263)

[Заключение 8](#_Toc179742264)

[Список использованных источников 9](#_Toc179742265)

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

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение подходов, системных объектов и функций для обеспечения передачи данных между взаимодействующими процессами и/или совместной их обработки. Также помимо этого в процессе выполнения лабораторной работы будут рассмотрены и изучены типичные проблемы, возникающие при организации взаимодействия, и пути их решения.

В качестве задачи необходимо разработать программу на языке C++ для интерактивного взаимодействия пользователей. В частности, программа должна представлять собой упрощенный чат для нескольких пользователей.

Для достижения этой цели требуется создать серверную часть, которая будет принимать подключения от клиентов и обеспечивать необходимую инфраструктуру для межпроцессного взаимодействия. Сервер должен принимать и временно хранить сообщения от клиентов, а затем передавать полученные сообщения адресно одному или нескольким клиентам.

Также необходимо разработать клиентскую часть, которая будет устанавливать соединение с сервером, позволяя пользователю вводить сообщения и отправлять их на сервер. Клиент должен иметь возможность принимать и отображать сообщения от сервера, полученные от других пользователей, обеспечивая интерактивность общения.

Кроме того, важно определить структуру передаваемых сообщений, которая будет включать как минимум адрес (идентификатор) отправителя и тело сообщения. Необходимо также разработать протокол обмена сообщениями, который определит порядок взаимодействия между клиентом и сервером, включая правила форматирования и обработки сообщений.

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.

Существует два типа каналов для двусторонней связи: неименованные каналы и именованные каналы. Неименованные каналы позволяют связанным процессам передавать информацию друг другу. Как правило, неименованный канал используется для перенаправления стандартного ввода или вывода дочернего процесса, чтобы он мог обмениваться данными с родительским процессом. Чтобы обмениваться данными в обоих направлениях (двусторонняя связь), необходимо создать два неименованных канала. Родительский процесс записывает данные в один канал с помощью дескриптора записи, а дочерний процесс считывает данные из этого канала с помощью дескриптора чтения. Аналогичным образом дочерний процесс записывает данные в другой канал, а родительский процесс считывает их. Неименованные каналы нельзя использовать в сети, а также между несвязанными процессами.

Именованные каналы используются для передачи данных между несвязанными процессами, а также между процессами на разных компьютерах. Как правило, серверный процесс именованного канала создаёт именованный канал с известным именем или именем, которое должно быть передано его клиентам. Клиентский процесс именованного канала, который знает имя канала, может открыть его другой конец с учётом ограничений доступа, указанных серверным процессом именованного канала. После подключения сервера и клиента к каналу они могут обмениваться данными, выполняя операции чтения и записи в канале.

Ключевой момент: неименованные каналы обеспечивают эффективный способ перенаправления стандартного ввода или вывода дочерним процессам на том же компьютере. Именованные каналы обеспечивают простой программный интерфейс для передачи данных между двумя процессами, независимо от того, находятся ли они на одном компьютере или в сети. [3]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Данная программа представляет собой реализацию простого взаимодействия между сервером и клиентом с использованием именованных каналов на Windows, демонстрируя базовую архитектуру клиент-серверного взаимодействия. Основной целью программы является создание и использование канала связи, через который клиент может отправлять сообщения серверу, получать ответы и поддерживать сеанс обмена данными.

В работе программы используются функции Windows API для создания и взаимодействия с именованными каналами. Серверная часть использует CreateNamedPipe, которая создает именованный канал с указанными параметрами доступа и типом канала, позволяя обмен сообщениями между клиентом и сервером. ConnectNamedPipe ожидает подключения клиента, обеспечивая синхронизацию взаимодействия. После завершения работы с клиентом соединение закрывается с помощью DisconnectNamedPipe, а дескриптор освобождается через CloseHandle. Клиентская часть применяет функцию CreateFile для открытия существующего именованного канала, предоставляя доступ для чтения и записи. Для передачи данных между клиентом и сервером используются функции WriteFile и ReadFile, которые осуществляют запись и чтение данных из канала соответственно. Таким образом, через именованные каналы организуется обмен сообщениями между процессами, обеспечивая синхронизацию и двустороннее взаимодействие. Именованные каналы (named pipes) представляют собой механизм межпроцессного взаимодействия, позволяющий передавать данные между процессами, работающими на одном или разных устройствах. В отличие от обычных файловых операций, именованные каналы предоставляют функции чтения и записи в режиме сообщений, что позволяет передавать данные блоками.

В данной программе сервер создает именованный канал с именем \\.\pipe\ServerPipe. После создания канал ожидает подключения клиента, как это продемонстрировано на рисунке 3.1.

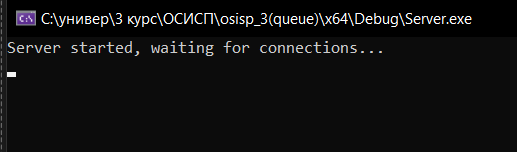


Рисунок 3.1 ­– Запуск сервера

Клиент, в свою очередь, подключается к каналу с помощью CreateFile (рисунок 3.2).

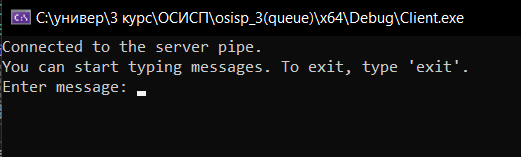


Рисунок 3.2 ­– Подключение клиента к серверу

После установления связи клиент отправляет серверу сообщения, которые сервер обрабатывает и возвращает подтверждение или ответное сообщение (рисунок 3.3).

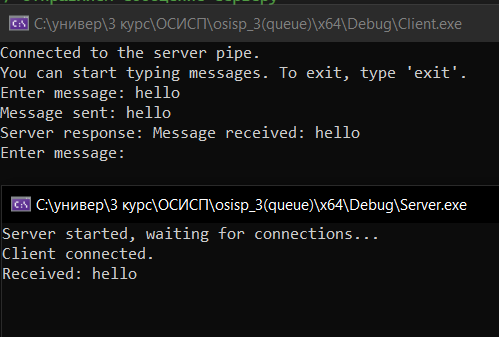


Рисунок 3.3 ­– Отправка и обработка сообщений

Сервер поддерживает выход клиентов и подключение новых, как в качестве примера продемонстрировано на рисунке 3.4.

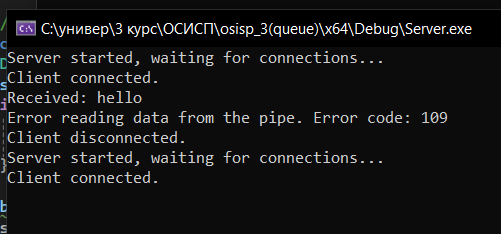


Рисунок 3.4 ­– Демонстрация выхода клиента и подключения нового

Для обмена данными сервер и клиент используют функции ReadFile и WriteFile, что позволяет реализовать синхронный обмен сообщениями.

Таким образом, данная программа демонстрирует базовые принципы взаимодействия процессов через именованные каналы с использованием функций WinAPI.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была создана программа для интерактивного обмена сообщениями между пользователями на основе клиент-серверной архитектуры с использованием именованных каналов. Основной целью работы стало изучение подходов, системных объектов и функций Windows API, обеспечивающих передачу данных между взаимодействующими процессами, а также решение типичных проблем, возникающих при организации межпроцессного взаимодействия.

Разработанное приложение успешно демонстрирует возможность передачи сообщений между клиентами через сервер с использованием именованных каналов. Серверная часть отвечает за создание каналов, подключение клиентов и маршрутизацию сообщений. Клиенты, подключаясь к серверу, могут обмениваться текстовыми сообщениями, которые передаются в реальном времени.

В процессе разработки программы применялись ключевые функции Windows API, включая CreateNamedPipe, ConnectNamedPipe, ReadFile и WriteFile, которые обеспечивают надежное взаимодействие между процессами через именованные каналы. Проведенная работа позволила углубить понимание механизмов межпроцессного обмена данными, выявить и устранить типичные проблемы, такие как обработка многократных подключений и передача данных с синхронизацией.

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 <windows.h>

int main() {

// Создаем именованный канал

HANDLE hPipe = INVALID\_HANDLE\_VALUE;

while (true) {

hPipe = CreateNamedPipe(

L"\\\\.\\pipe\\ServerPipe", // Имя канала

PIPE\_ACCESS\_DUPLEX, // Доступ для чтения и записи

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, // Тип канала

PIPE\_UNLIMITED\_INSTANCES, // Неограниченное количество клиентов

1024, // Размер выходного буфера

1024, // Размер входного буфера

0, // Ожидание по умолчанию

NULL // Атрибуты безопасности

);

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

std::cerr << "Failed to create named pipe. Error code: " << GetLastError() << std::endl;

return 1;

}

std::cout << "Server started, waiting for connections...\n";

// Ожидаем подключения клиента

BOOL success = ConnectNamedPipe(hPipe, NULL);

if (!success) {

std::cerr << "Failed to connect to the named pipe. Error code: " << GetLastError() << std::endl;

CloseHandle(hPipe);

continue; // Пропускаем текущую итерацию и пытаемся снова

}

std::wcout << L"Client connected.\n";

// Буфер для чтения данных

char buffer[1024];

DWORD bytesRead;

// Чтение данных от клиента

while (true) {

success = ReadFile(hPipe, buffer, sizeof(buffer), &bytesRead, NULL);

if (!success || bytesRead == 0) {

std::cerr << "Error reading data from the pipe. Error code: " << GetLastError() << std::endl;

break; // Выход из цикла, если данные не были получены

}

buffer[bytesRead] = '\0'; // Завершаем строку

std::wcout << L"Received: " << buffer << std::endl;

// Отправляем ответ клиенту

std::string response = "Message received: " + std::string(buffer);

DWORD bytesWritten;

success = WriteFile(hPipe, response.c\_str(), response.length(), &bytesWritten, NULL);

if (!success) {

std::cerr << "Error writing response to pipe. Error code: " << GetLastError() << std::endl;

break;

}

}

// Закрываем соединение с клиентом

DisconnectNamedPipe(hPipe);

std::cout << "Client disconnected.\n";

// Закрываем дескриптор канала для текущего клиента

CloseHandle(hPipe);

}

return 0;

}

#include <iostream>

#include <windows.h>

#include <string>

int main() {

// Имя канала

const std::wstring pipeName = L"\\\\.\\pipe\\ServerPipe";

// Открываем именованный канал

HANDLE hPipe = CreateFile(

pipeName.c\_str(), // Имя канала

GENERIC\_READ | GENERIC\_WRITE, // Доступ для чтения и записи

0, // Нет совместного доступа

NULL, // Атрибуты безопасности

OPEN\_EXISTING, // Открыть уже существующий канал

0, // Атрибуты файла

NULL // Шаблон объекта

);

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

std::cerr << "Failed to connect to the server pipe. Error code: " << GetLastError() << std::endl;

return 1;

}

std::cout << "Connected to the server pipe.\n";

std::cout << "You can start typing messages. To exit, type 'exit'.\n";

std::string message;

while (true) {

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

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

if (message == "exit") {

break;

}

// Отправляем сообщение серверу

DWORD bytesWritten;

BOOL success = WriteFile(hPipe, message.c\_str(), message.length(), &bytesWritten, NULL);

if (!success) {

std::cerr << "Error writing message to the pipe. Error code: " << GetLastError() << std::endl;

break;

}

std::cout << "Message sent: " << message << std::endl;

// Чтение ответа от сервера

char buffer[1024];

DWORD bytesRead;

success = ReadFile(hPipe, buffer, sizeof(buffer), &bytesRead, NULL);

if (!success || bytesRead == 0) {

std::cerr << "Error reading server response. Error code: " << GetLastError() << std::endl;

break;

}

buffer[bytesRead] = '\0'; // Завершаем строку

std::cout << "Server response: " << buffer << std::endl;

}

CloseHandle(hPipe);

return 0;

}