**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: «Межпроцессное взаимодействие»**

Студента гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Половникова А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

исследовать инструменты и механизмы взаимодействия процессов в Windows.

**Задание 4.2.**

Использование именованных каналов для реализации сетевого межпроцессного взаимодействия.

**Указания к выполнению.**

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

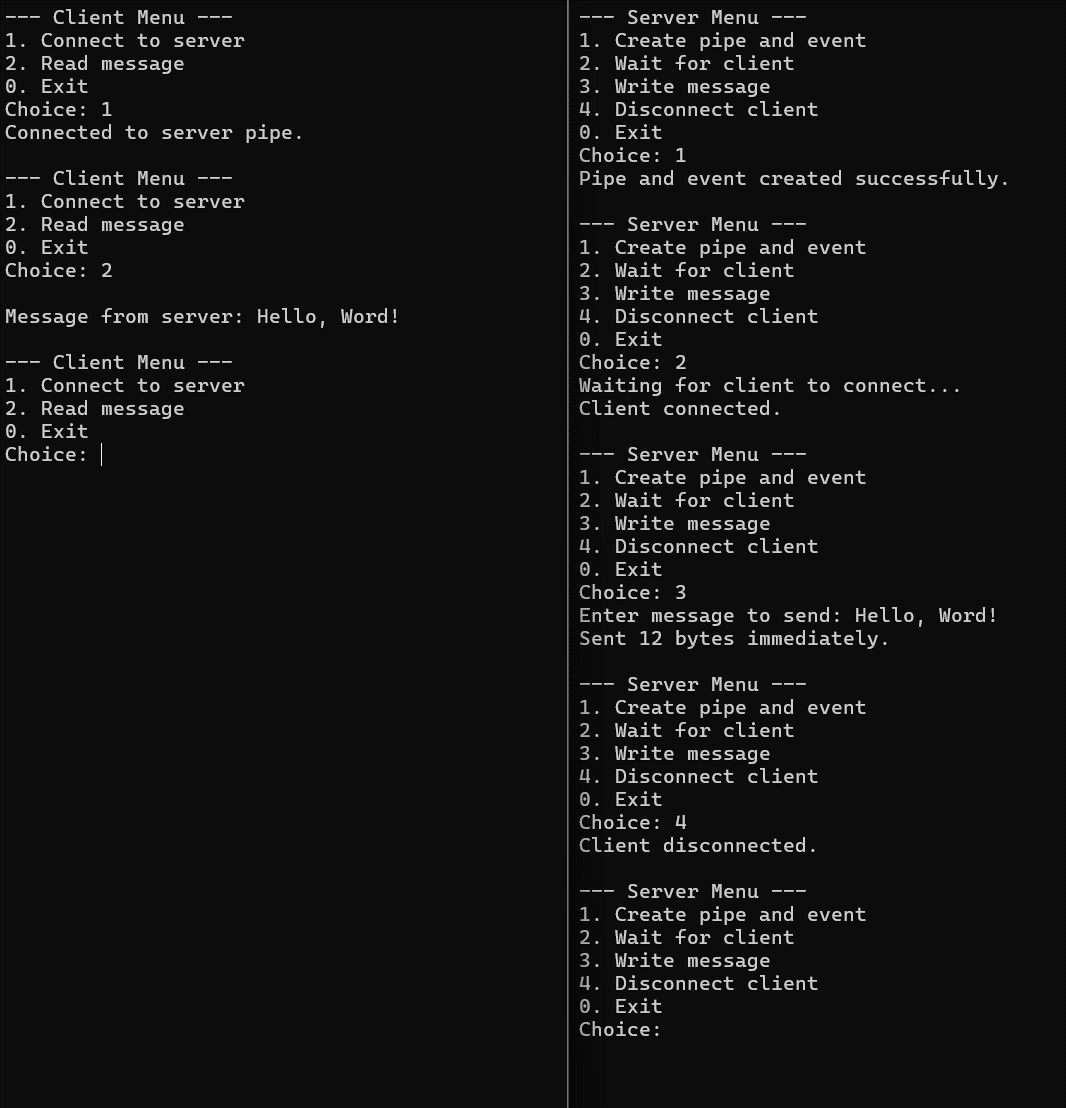
Приложение-сервер создает именованный канал (функция Win32  API – **CreateNamedPipe**), выполняет установление и отключение  соединения (функции Win32 API – **ConnectNamedPipe**,  **DisconnectNamedPipe**), создает объект «событие» (функция Win32  API – **CreateEvent**) осуществляет ввод данных с клавиатуры и их  асинхронную запись в именованный канал (функция Win32 API – **WriteFile**), выполняет ожидание завершения операции  вводавывода (функция Win32 API – **WaitForSingleObject**);

Приложение-клиент подключается к именованному каналу (функция Win32 API – **CreateFile**), в асинхронном режиме считывает содержимое из именованного канала файла (функция Win32 API – **ReadFileEx**) и отображает на экран.

2. Запустите приложения и проверьте обмен данных между процессами. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по заданию.

Описание работы



**Сервер**

**1. CreateNamedPipeW**

**Назначение**: создание именованного канала на стороне сервера.  
**Параметры**:

* PIPE\_ACCESS\_OUTBOUND | FILE\_FLAG\_OVERLAPPED: асинхронный вывод (только запись);
* PIPE\_TYPE\_MESSAGE: передача сообщений (message-based);
* PIPE\_WAIT: синхронное ожидание чтения клиентом.

Установка флага FILE\_FLAG\_OVERLAPPED указывает на использование асинхронных операций ввода-вывода.

**2. CreateEventW**

**Назначение**: создание события для отслеживания завершения асинхронных операций.   
Создается вручную сбрасываемое событие (manual-reset = TRUE). Это позволяет использовать одно событие повторно для нескольких операций. Используется для связи с структурой OVERLAPPED.

**3. ConnectNamedPipe**

**Назначение**: ожидание подключения клиента к каналу.  
**Особенности**:

* При ERROR\_IO\_PENDING — асинхронная операция; требуется ожидание события.
* При ERROR\_PIPE\_CONNECTED — клиент уже подключен.

Используется с OVERLAPPED, как и предполагает асинхронный режим. Обработка ConnectNamedPipe с учётом возможных кодов возврата.

**4. WriteFile**

**Назначение**: отправка сообщения клиенту.  
Асинхронная запись с OVERLAPPED. При ERROR\_IO\_PENDING сервер ожидает завершения операции через WaitForSingleObject(hEvent, INFINITE). Далее используется GetOverlappedResult для получения количества переданных байтов.

**Клиент**

**1. CreateFileW**

**Назначение**: подключение к существующему именованному каналу.  
**Параметры**:

* GENERIC\_READ: только чтение;
* FILE\_FLAG\_OVERLAPPED: асинхронный ввод.

Клиент корректно использует асинхронное подключение, готов к асинхронному чтению.

**2. ReadFileEx**

**Назначение**: асинхронное чтение из канала с указанием завершения через callback.   
Используется структура OVERLAPPED и callback-функция ReadCompleted. После запуска чтения вызывается SleepEx(INFINITE, TRUE) — это обязательное условие для обработки завершения асинхронных операций через APC (Asynchronous Procedure Call).

**3. ReadCompleted (callback)**

**Назначение**: вызывается при завершении ReadFileEx.   
Проверяется код ошибки, количество прочитанных байт. Буфер корректно завершается нулем для вывода как строки.

**Итоги**

Программа демонстрирует корректную работу с асинхронным вводом-выводом.

* Используются **ключевые возможности Win32 API**: OVERLAPPED, CreateEvent, ConnectNamedPipe, WriteFile, ReadFileEx.
* Реализована обработка ошибок и возможных сценариев (например, ERROR\_IO\_PENDING, ERROR\_PIPE\_CONNECTED).
* Механизм завершения чтения через APC (ReadFileEx + SleepEx).

Выводы

В ходе выполнения лабораторной работы было успешно реализовано клиент-серверное приложение на основе именованных каналов (Named Pipes) с использованием асинхронного ввода-вывода (Overlapped I/O) средствами Win32 API.

Серверная часть создает именованный канал и событие, ожидает подключения клиента и выполняет отправку данных в асинхронном режиме. Клиент подключается к каналу и считывает сообщение, используя асинхронное чтение с callback-функцией. Оба приложения корректно обрабатывают состояния и ошибки, связанные с асинхронными операциями.

Работа позволила на практике закрепить навыки работы с ключевыми функциями Win32 API:  
CreateNamedPipe, ConnectNamedPipe, WriteFile, ReadFileEx, CreateEvent, GetOverlappedResult и другими.

Таким образом, поставленные цели были достигнуты, а полученный результат демонстрирует уверенное применение механизмов межпроцессного взаимодействия Windows на низком уровне.

Текст программы

Client.cpp

#include <windows.h>

#include <iostream>

const wchar\_t\* pipeName = L"\\\\.\\pipe\\Pipe";

HANDLE hPipe = INVALID\_HANDLE\_VALUE;

struct OverlappedData {

    OVERLAPPED overlapped{};

    char buffer[1024]{};

};

void CALLBACK ReadCompleted(DWORD dwErrCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {

    OverlappedData\* data = reinterpret\_cast<OverlappedData\*>(lpOverlapped);

    if (dwErrCode == ERROR\_SUCCESS && dwNumberOfBytesTransfered > 0) {

        data->buffer[dwNumberOfBytesTransfered] = '\0';

        std::cout << "\nMessage from server: " << data->buffer << std::endl;

    }

    else {

        std::cerr << "\nReadFileEx failed or no data received. Error code: " << dwErrCode << std::endl;

    }

    delete data;

}

void ConnectToPipe() {

    hPipe = CreateFileW(

        pipeName,

        GENERIC\_READ,

        0,

        nullptr,

        OPEN\_EXISTING,

        FILE\_FLAG\_OVERLAPPED,

        nullptr);

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

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

    }

    else {

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

    }

}

void ReadMessage() {

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

        std::cerr << "Not connected to pipe.\n";

        return;

    }

    auto\* data = new OverlappedData();

    ZeroMemory(&data->overlapped, sizeof(OVERLAPPED));

    BOOL success = ReadFileEx(

        hPipe,

        data->buffer,

        sizeof(data->buffer) - 1,

        &data->overlapped,

        ReadCompleted);

    if (!success) {

        std::cerr << "ReadFileEx failed. Error: " << GetLastError() << std::endl;

        delete data;

        return;

    }

    SleepEx(INFINITE, TRUE);

}

int main() {

    int choice;

    do {

        std::cout << "\n--- Client Menu ---\n";

        std::cout << "1. Connect to server\n";

        std::cout << "2. Read message\n";

        std::cout << "0. Exit\nChoice: ";

        std::cin >> choice;

        switch (choice) {

        case 1:

            ConnectToPipe();

            break;

        case 2:

            ReadMessage();

            break;

        }

    } while (choice != 0);

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

        CloseHandle(hPipe);

    }

    return 0;

}

Server.cpp

#include <windows.h>

#include <iostream>

#include <string>

HANDLE hPipe = INVALID\_HANDLE\_VALUE;

HANDLE hEvent = nullptr;

OVERLAPPED overlapped = {};

const wchar\_t\* pipeName = L"\\\\.\\pipe\\Pipe";

void CreatePipeAndEvent() {

    hPipe = CreateNamedPipeW(

        pipeName,

        PIPE\_ACCESS\_OUTBOUND | FILE\_FLAG\_OVERLAPPED,

        PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

        1, 1024, 1024, 0, nullptr);

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

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

        return;

    }

    hEvent = CreateEventW(nullptr, TRUE, FALSE, nullptr);

    if (!hEvent) {

        std::cerr << "Failed to create event. Error: " << GetLastError() << "\n";

        CloseHandle(hPipe);

        hPipe = INVALID\_HANDLE\_VALUE;

        return;

    }

    overlapped = {};

    overlapped.hEvent = hEvent;

    std::cout << "Pipe and event created successfully.\n";

}

void WaitForClient() {

    if (hPipe == INVALID\_HANDLE\_VALUE || !hEvent) {

        std::cerr << "Pipe or event not initialized.\n";

        return;

    }

    BOOL result = ConnectNamedPipe(hPipe, &overlapped);

    if (!result) {

        DWORD err = GetLastError();

        if (err == ERROR\_IO\_PENDING) {

            std::cout << "Waiting for client to connect...\n";

            WaitForSingleObject(hEvent, INFINITE);

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

        }

        else if (err == ERROR\_PIPE\_CONNECTED) {

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

            SetEvent(hEvent);

        }

        else {

            std::cerr << "Failed to connect to client. Error: " << err << "\n";

        }

    }

    else {

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

    }

}

void WriteToClient() {

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

        std::cerr << "Pipe is not connected.\n";

        return;

    }

    std::string input;

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

    std::getline(std::cin, input);

    if (input.empty()) {

        std::cerr << "Empty message. Nothing sent.\n";

        return;

    }

    DWORD bytesWritten = 0;

    ResetEvent(hEvent);

    BOOL success = WriteFile(

        hPipe,

        input.c\_str(),

        static\_cast<DWORD>(input.size()),

        nullptr,

        &overlapped);

    if (!success) {

        DWORD err = GetLastError();

        if (err == ERROR\_IO\_PENDING) {

            WaitForSingleObject(hEvent, INFINITE);

            if (GetOverlappedResult(hPipe, &overlapped, &bytesWritten, FALSE)) {

                std::cout << "Sent " << bytesWritten << " bytes.\n";

            }

            else {

                std::cerr << "GetOverlappedResult failed. Error: " << GetLastError() << "\n";

            }

        }

        else {

            std::cerr << "WriteFile failed immediately. Error: " << err << "\n";

        }

    }

    else {

        if (GetOverlappedResult(hPipe, &overlapped, &bytesWritten, FALSE)) {

            std::cout << "Sent " << bytesWritten << " bytes immediately.\n";

        }

    }

}

void Disconnect() {

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

        DisconnectNamedPipe(hPipe);

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

    }

    else {

        std::cerr << "No pipe to disconnect.\n";

    }

}

int main() {

    int choice;

    do {

        std::cout << "\n--- Server Menu ---\n";

        std::cout << "1. Create pipe and event\n";

        std::cout << "2. Wait for client\n";

        std::cout << "3. Write message\n";

        std::cout << "4. Disconnect client\n";

        std::cout << "0. Exit\nChoice: ";

        std::cin >> choice;

        std::cin.ignore();

        switch (choice) {

        case 1: CreatePipeAndEvent(); break;

        case 2: WaitForClient(); break;

        case 3: WriteToClient(); break;

        case 4: Disconnect(); break;

        }

    } while (choice != 0);

    if (hPipe != INVALID\_HANDLE\_VALUE) CloseHandle(hPipe);

    if (hEvent) CloseHandle(hEvent);

    return 0;

}