**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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Поляков А.И.

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:

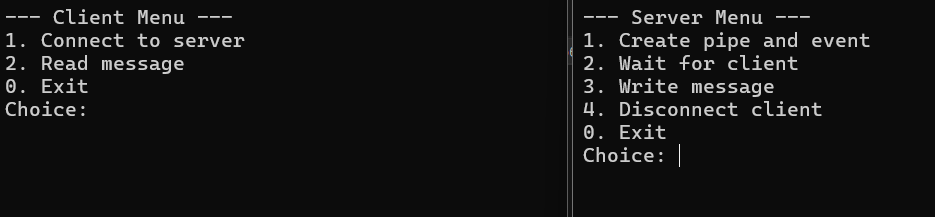
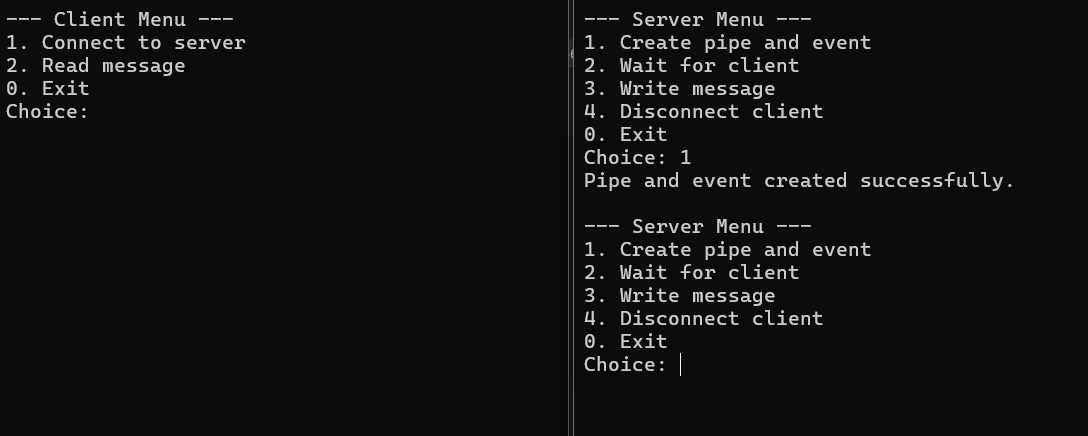
Приложение-сервер создает именованный канал (функция Win32  API – **CreateNamedPipe**), выполняет установление и отключение  соединения (функции Win32 API – **ConnectNamedPipe**,  **DisconnectNamedPipe**), создает объект «событие» (функция Win32  API – **CreateEvent**) осуществляет ввод данных с клавиатуры 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.

**Пример выполнения программы**

****

На стороне сервера (справа):

Выбор 1. Create pipe and event

* Пользователь вызвал функцию CreatePipeAndEvent(), которая:

CreateNamedPipeW(...)

Создаёт именованный канал (named pipe) с именем:

L"\\\\.\\pipe\\MyPipe"

Этот путь указывает на локальную машину (\\\\.\\) и имя канала (MyPipe). Ключевые параметры:

* PIPE\_ACCESS\_OUTBOUND | FILE\_FLAG\_OVERLAPPED: позволяет запись и использование асинхронных операций.
* PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT: работает в режиме сообщений, что обеспечивает чёткое разделение переданных блоков.
* Буферы ввода/вывода по 1024 байта.
* Один инстанс канала.

Если всё прошло успешно, дескриптор канала сохраняется в переменной hPipe.

CreateEventW(...)

Создаёт событие синхронизации, которое будет использоваться в OVERLAPPED-структуре для асинхронных операций, например, ожидания подключения клиента или завершения записи:

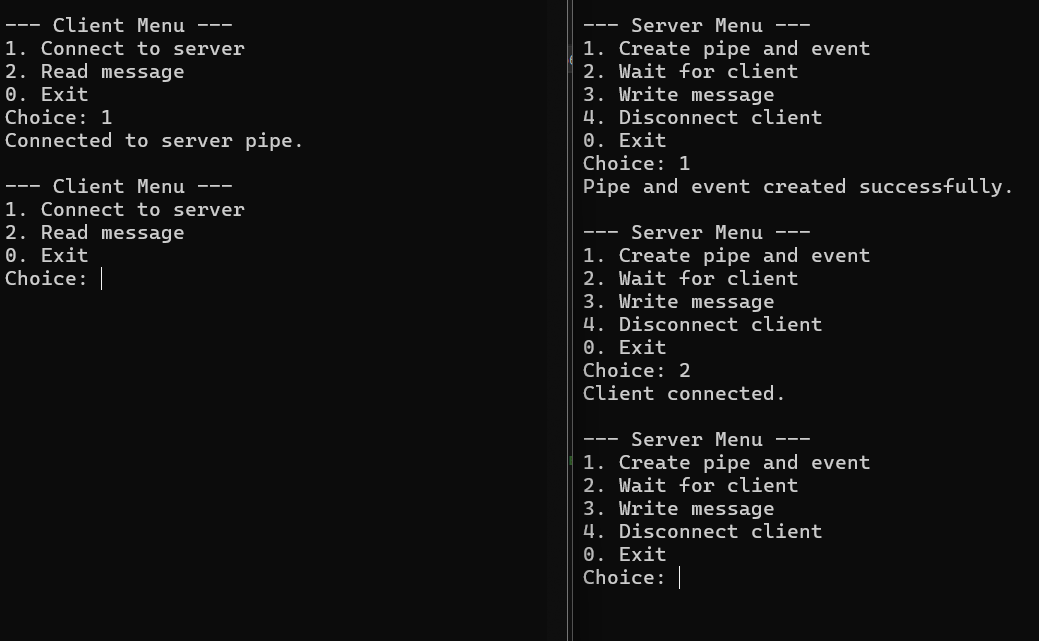
* TRUE — manual reset.
* FALSE — изначально не установленное состояние.

Если событие создано — оно привязывается к overlapped.hEvent.

На стороне клиента (слева):

Пока клиент просто запущен и ожидает действий от пользователя. Он ещё не подключён к каналу. Как видно:

* Отображено основное меню с опциями:
  + Подключиться к серверу.
  + Прочитать сообщение.
  + Выход.

****

После запуска клиента, была выбрана опция "1. Connect to server", в результате чего вызвана функция CreateFileW. Эта функция подключается к ранее созданному именованному каналу \\.\pipe\MyPipe. Клиент использует только чтение, что соответствует архитектуре.

На серверной стороне для ожидания подключения используется ConnectNamedPipe с асинхронной моделью через OVERLAPPED. В случае асинхронного ожидания используется WaitForSingleObject, ожидающий сигнала события hEvent.

Успешное подключение подтверждается сообщениями:

* Клиент: "Connected to server pipe."
* Сервер: "Client connected."

****

На стороне сервера:

Выбор 3. Write message

Выполняется функция WriteToClient(). Она отвечает за отправку текста, введённого пользователем, в канал (клиенту).

Ввод сообщения:

Пользователь ввёл:

Hello, World!

(13 байт, включая пробел и восклицательный знак)

Что делает код:

ResetEvent(hEvent);

Сбрасывает событие, чтобы обеспечить корректное ожидание завершения текущей асинхронной операции.

WriteFile(...)

Пытается асинхронно записать данные в канал:

WriteFile(

hPipe,

input.c\_str(),

input.size(),

nullptr,

&overlapped);

GetLastError() и ожидание:

Если WriteFile вернул FALSE, а GetLastError() == ERROR\_IO\_PENDING, то это означает, что операция ещё не завершилась, но выполняется асинхронно.

В этом случае вызывается:

WaitForSingleObject(hEvent, INFINITE);

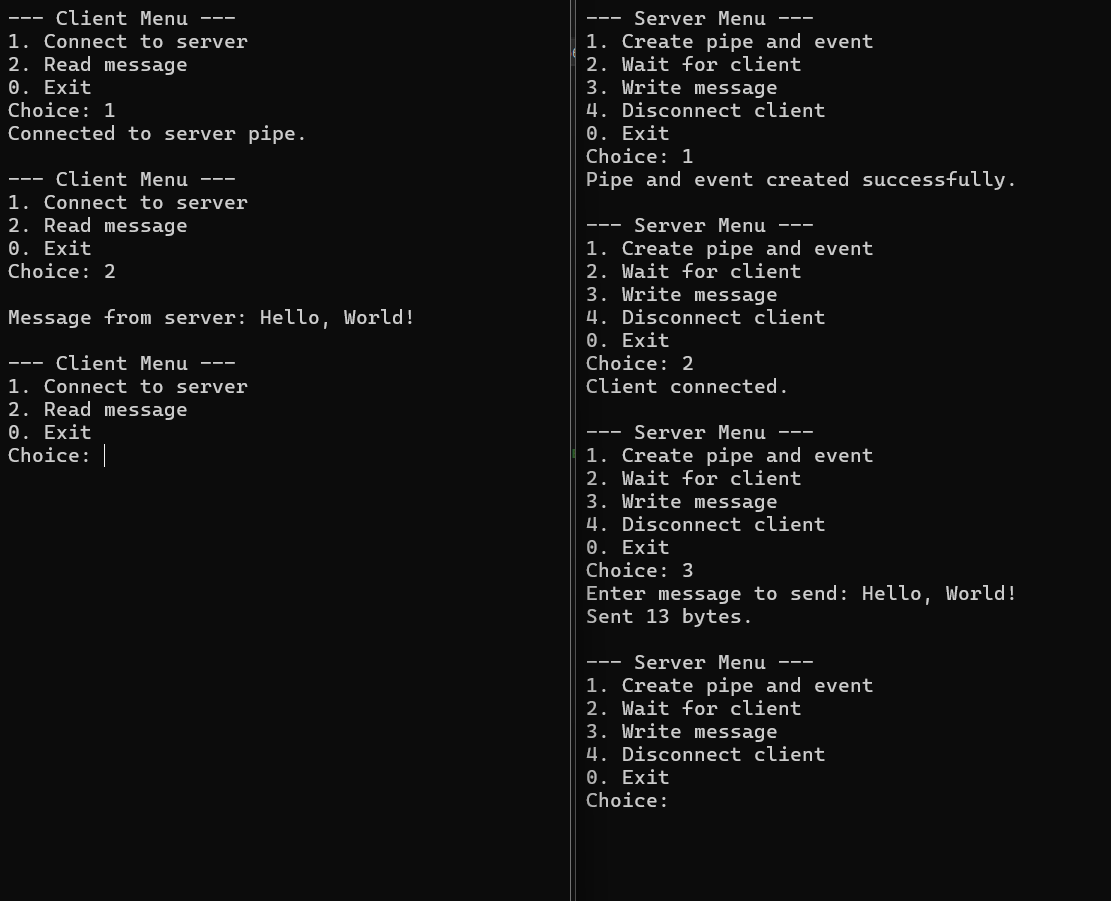
чтобы дождаться завершения.

Затем GetOverlappedResult(...) извлекает количество реально записанных байт:

GetOverlappedResult(hPipe, &overlapped, &bytesWritten, FALSE);

Успешная передача подтверждается сообщением:

"Sent 13 bytes."

****

На стороне клиента:

Выбор 2. Read message

Запускается функция ReadMessage():

Буфер и OVERLAPPED:

char buffer[1024] = {};

OVERLAPPED overlapped = {};

overlapped.hEvent = buffer;

Вместо дескриптора события, как обычно, в поле hEvent записывается указатель на буфер, чтобы потом этот буфер легко получить в колбэке ReadCompleted.

ReadFileEx(...)

Чтение происходит асинхронно:

ReadFileEx(

hPipe,

buffer,

sizeof(buffer),

&overlapped,

ReadCompleted);

Если чтение начинается успешно, программа переходит в "ожидание завершения" с помощью:

SleepEx(INFINITE, TRUE);

* TRUE в аргументе говорит системе: "разрешить выполнение асинхронных завершений (APC)", в частности, функцию ReadCompleted.

Колбэк ReadCompleted:

Когда данные приходят, вызывается:

void CALLBACK ReadCompleted(...) {

std::cout << "\nMessage from server: " << (char\*)lpOverlapped->hEvent << std::endl;

}

* Тут и происходит дешифровка hEvent — мы получаем указатель на буфер, в который пришли данные, и печатаем сообщение:

"Message from server: Hello, World!"

****

После завершения обмена данными сервер вызывает функцию DisconnectNamedPipe, которая разрывает соединение с клиентом, оставляя именованный канал доступным для повторного подключения.

Клиент, не зная о разрыве, выполняет ReadFileEx, однако чтение завершается неудачно. Причиной является отсутствие соединения с сервером: дескриптор существует, но передача данных невозможна.

Вызов ReadFileEx возвращает FALSE, и клиент сообщает об ошибке. Такой результат демонстрирует корректное поведение API в условиях завершения соединения и подчеркивает важность проверки состояния канала перед чтением.

**Вывод**

Приведённая реализация демонстрирует базовые принципы межпроцессного взаимодействия (IPC) с помощью именованных каналов в Windows:

1. Асинхронность  
   Использование флага FILE\_FLAG\_OVERLAPPED позволяет избежать блокирующих вызовов и выполнять операции ввода-вывода в фоне, повышая отзывчивость приложения.
2. События и OVERLAPPED  
   Механизм событий (CreateEvent, WaitForSingleObject) и структура OVERLAPPED позволяют синхронизировать асинхронные операции, обеспечивая гибкость при выполнении ReadFileEx и WriteFile.
3. Простое меню  
   Меню управления на стороне клиента и сервера делает взаимодействие интерактивным и позволяет вручную управлять жизненным циклом соединения.
4. Практическое применение  
   Подобный подход может использоваться для создания локальных клиент-серверных приложений, систем логирования, сервисов, требующих коммуникации между процессами и т.д.

Текст программы

*Client.cpp*

#include <windows.h>

#include <iostream>

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

HANDLE hPipe;

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

if (dwErrCode == ERROR\_SUCCESS) {

std::cout << "\nMessage from server: " << (char\*)lpOverlapped->hEvent << std::endl;

}

else {

std::cerr << "\nReadFileEx failed with code: " << dwErrCode << std::endl;

}

}

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.\n";

return;

}

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

}

void ReadMessage() {

char buffer[1024] = {};

OVERLAPPED overlapped = {};

overlapped.hEvent = buffer;

BOOL success = ReadFileEx(

hPipe,

buffer,

sizeof(buffer),

&overlapped,

ReadCompleted);

if (!success) {

std::cerr << "ReadFileEx failed.\n";

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);

CloseHandle(hPipe);

return 0;

}

*Server.cpp*

#include <windows.h>

#include <iostream>

#include <string>

HANDLE hPipe, hEvent;

OVERLAPPED overlapped = {};

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

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.\n";

return;

}

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

if (!hEvent) {

std::cerr << "Failed to create event.\n";

return;

}

overlapped.hEvent = hEvent;

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

}

void WaitForClient() {

if (!ConnectNamedPipe(hPipe, &overlapped)) {

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

WaitForSingleObject(hEvent, INFINITE);

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

}

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

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

SetEvent(hEvent);

}

else {

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

}

}

}

void WriteToClient() {

std::string input;

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

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

DWORD bytesWritten;

ResetEvent(hEvent);

BOOL success = WriteFile(

hPipe,

input.c\_str(),

input.size(),

nullptr,

&overlapped);

if (success && GetLastError() == ERROR\_IO\_PENDING) {

WaitForSingleObject(hEvent, INFINITE);

GetOverlappedResult(hPipe, &overlapped, &bytesWritten, FALSE);

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

}

else {

std::cerr << "WriteFile failed.\n";

}

}

void Disconnect() {

DisconnectNamedPipe(hPipe);

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

CloseHandle(hPipe);

CloseHandle(hEvent);

return 0;

}