**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:

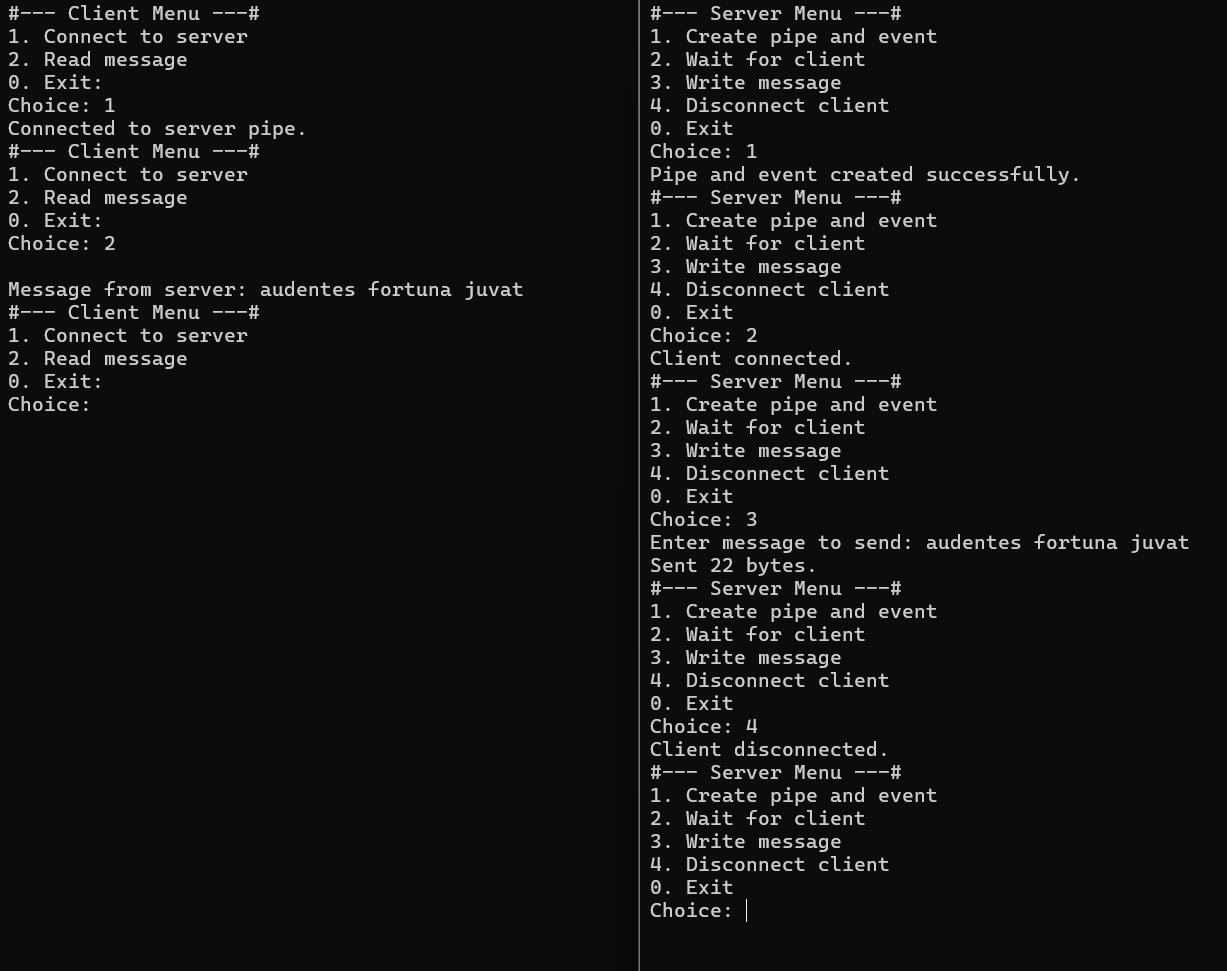
Приложение-сервер создает именованный канал (функция Win32  API – **CreateNamedPipe**), выполняет установление и отключение  соединения (функции Win32 API – **ConnectNamedPipe**,  **DisconnectNamedPipe**), создает объект «событие» (функция Win32  API – **CreateEvent**) осуществляет ввод данных с клавиатуры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

**Выполнение программы**



**Используемые функции Win32 API**

**1. CreateNamedPipeW**

* **Назначение:** создаёт именованный канал на стороне сервера.
* **Комментарий:**  
  Канал создаётся с флагом FILE\_FLAG\_OVERLAPPED, что позволяет выполнять асинхронные операции. Указан тип канала PIPE\_TYPE\_MESSAGE с режимом PIPE\_READMODE\_MESSAGE, что позволяет передавать данные как отдельные сообщения, а не поток байтов.

**2. CreateEventW**

* **Назначение:** создаёт объект синхронизации (event), необходимый для управления асинхронными операциями.
* **Комментарий:**  
  Событие создается вручную (TRUE), что означает необходимость ручного сброса (ResetEvent). Оно используется в структуре OVERLAPPED для отслеживания завершения операций ConnectNamedPipe и WriteFile.

**3. ConnectNamedPipe**

* **Назначение:** ожидает подключения клиента к каналу.
* **Комментарий:**  
  Используется в асинхронном режиме с OVERLAPPED, и возможен сценарий с ERROR\_PIPE\_CONNECTED, когда клиент уже подключён. В этом случае вызывается SetEvent, чтобы не потерять сигнал завершения.

**4. WriteFile (с OVERLAPPED)**

* **Назначение:** отправляет сообщение клиенту по каналу.
* **Комментарий:**  
  Асинхронная операция записи. После вызова WriteFile выполнение не блокируется. Ожидание завершения реализовано через WaitForSingleObject, а окончание проверяется через GetOverlappedResult.

**5. GetOverlappedResult**

* **Назначение:** получает результат асинхронной операции.
* **Комментарий:**  
  Позволяет узнать, сколько байт было реально отправлено клиенту.

**6. DisconnectNamedPipe**

* **Назначение:** завершает текущую сессию соединения с клиентом.
* **Комментарий:**  
  Используется для корректного закрытия соединения, после чего канал может быть переиспользован.

**На стороне клиента**

**1. CreateFileW**

* **Назначение:** устанавливает соединение с уже созданным каналом на стороне сервера.
* **Комментарий:**  
  Используется с флагом FILE\_FLAG\_OVERLAPPED для поддержки асинхронного чтения. В случае ошибки отображается сообщение.

**2. ReadFileEx**

* **Назначение:** инициирует асинхронную операцию чтения.
* **Комментарий:**  
  Буфер передается как hEvent в OVERLAPPED, что является нестандартным, но работающим способом доставки результата в OnReadComplete. Это возможно, так как в колбэке hEvent может быть использовано для передачи пользовательских данных.

**3. SleepEx**

* **Назначение:** переводит поток в режим ожидания асинхронных вызовов.
* **Комментарий:**  
  Используется с параметром TRUE, чтобы разрешить выполнение завершённых асинхронных операций (APC).

**Выводы по выполнению задания**

1. **Функции Win32 API успешно использованы** для реализации взаимодействия между клиентом и сервером с помощью именованных каналов в асинхронном режиме.
2. **Реализована асинхронность**, что позволяет избежать блокировки основного потока и управлять операциями через событие (Event) и APC (ReadFileEx + SleepEx).

**Вывод**

Задание выполнено корректно: реализована работа с именованными каналами и асинхронными вызовами Win32 API. Взаимодействие между клиентом и сервером работает по модели «запрос-ответ». Все ключевые функции API задействованы правильно и используются по назначению.

Тест программы

Client

#include <windows.h>

#include <iostream>

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

HANDLE gPipeHandle = nullptr;

void CALLBACK OnReadComplete(DWORD errorCode, DWORD bytesRead, LPOVERLAPPED overlappedData);

void ReceiveData();

void StartConnection();

int main() {

int userInput = -1;

while (userInput != 0) {

std::cout << "#--- Client Menu ---#" << std::endl;

std::cout << "1. Connect to server" << std::endl;

std::cout << "2. Read message" << std::endl;

std::cout << "0. Exit: " << std::endl;

std::cout << "Choice: ";

std::cin >> userInput;

switch (userInput) {

case 1:

StartConnection();

break;

case 2:

ReceiveData();

break;

}

}

if (gPipeHandle != nullptr && gPipeHandle != INVALID\_HANDLE\_VALUE) {

CloseHandle(gPipeHandle);

}

return 0;

}

void CALLBACK OnReadComplete(DWORD errorCode, DWORD bytesRead, LPOVERLAPPED overlappedData) {

if (errorCode == ERROR\_SUCCESS) {

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

}

else {

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

}

}

void ReceiveData() {

char recvBuffer[1024] = {};

OVERLAPPED asyncStruct = {};

asyncStruct.hEvent = recvBuffer;

BOOL readResult = ReadFileEx(

gPipeHandle,

recvBuffer,

sizeof(recvBuffer),

&asyncStruct,

OnReadComplete

);

if (!readResult) {

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

return;

}

SleepEx(INFINITE, TRUE);

}

void StartConnection() {

gPipeHandle = CreateFileW(

gPipeName,

GENERIC\_READ,

0,

nullptr,

OPEN\_EXISTING,

FILE\_FLAG\_OVERLAPPED,

nullptr

);

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

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

}

else {

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

}

}

Server

#include <windows.h>

#include <iostream>

#include <string>

HANDLE gPipeHandle = nullptr;

HANDLE gEventHandle = nullptr;

OVERLAPPED gAsyncOp = {};

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

void InitializeResources();

void WaitForIncomingConnection();

void SendMessageToClient();

void CloseConnection();

int main() {

int menuChoice = -1;

while (menuChoice != 0) {

std::cout << "#--- Server Menu ---#" << std::endl;

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

std::cout << "2. Wait for client" << std::endl;

std::cout << "3. Write message" << std::endl;

std::cout << "4. Disconnect client" << std::endl;

std::cout << "0. Exit " << std::endl;

std::cout << "Choice: ";

std::cin >> menuChoice;

std::cin.ignore();

switch (menuChoice) {

case 1: InitializeResources(); break;

case 2: WaitForIncomingConnection(); break;

case 3: SendMessageToClient(); break;

case 4: CloseConnection(); break;

}

}

if (gPipeHandle) CloseHandle(gPipeHandle);

if (gEventHandle) CloseHandle(gEventHandle);

return 0;

}

void InitializeResources() {

gPipeHandle = CreateNamedPipeW(

kPipePath,

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

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

1, 1024, 1024, 0, nullptr

);

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

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

return;

}

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

if (!gEventHandle) {

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

return;

}

gAsyncOp.hEvent = gEventHandle;

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

}

void WaitForIncomingConnection() {

BOOL isPending = ConnectNamedPipe(gPipeHandle, &gAsyncOp);

if (!isPending) {

DWORD error = GetLastError();

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

WaitForSingleObject(gEventHandle, INFINITE);

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

}

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

SetEvent(gEventHandle);

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

}

else {

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

}

}

}

void SendMessageToClient() {

std::string message;

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

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

ResetEvent(gEventHandle);

DWORD bytesSent = 0;

WriteFile(

gPipeHandle,

message.c\_str(),

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

nullptr,

&gAsyncOp

);

WaitForSingleObject(gEventHandle, INFINITE);

GetOverlappedResult(gPipeHandle, &gAsyncOp, &bytesSent, FALSE);

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

}

void CloseConnection() {

DisconnectNamedPipe(gPipeHandle);

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

}