МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 4.2

по дисциплине «Операционные системы»

ТЕМА: «Межпроцессорное взаимодействие»

Студент гр. 3311 Баймухамедов Р. Р.

Преподаватель Тимофеев А. В.

Санкт-Петербург

2025

Цель работы
Исследовать инструменты и механизмы взаимодействия процессов в Windows
Задание
Использование именнованных каналов для реализации сетевого межпроцессорного взаимодействия

Постановка задачи

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отедльному пункту меню), которые выполняют:
   - Приложение-сервер создает именованный канал (функция Win32 API – CreateNamedPipe), выполняет установление и отключение соединения (функции Win32 API – ConnectNamedPipe, DisconnectNamedPipe), создает обьект “событие” (функция Win32 API – CreateEvent) осуществляет ввод данных с клавиатуры и их асинхронную запись в именованный канал (функция Win 32 API – WriteFile), выполняет ожидание завершения операции ввода/вывода (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. Запротоклируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.
3. Подготовьте итоговый отчет с развернутыми выводами по заданию.

Выполнение задания

В данной лабораторной работе реализован механизм межпроцессного взаимодействия (IPC) на основе именованных каналов (Named Pipes) в операционной системе Windows. Этот механизм обеспечивает передачу данных между двумя независимыми приложениями: сервером и клиентом.
Обмен информацией осуществляется по следующей схеме:
На стороне сервера:

создаётся именованный канал через CreateNamedPipe,
выполняется ожидание подключения клиента с помощью ConnectNamedPipe,
ввод с клавиатуры осуществляется через fgets,
запись в канал выполняется асинхронно с использованием WriteFile и структуры OVERLAPPED,
завершение операции записи контролируется через WaitForSingleObject.

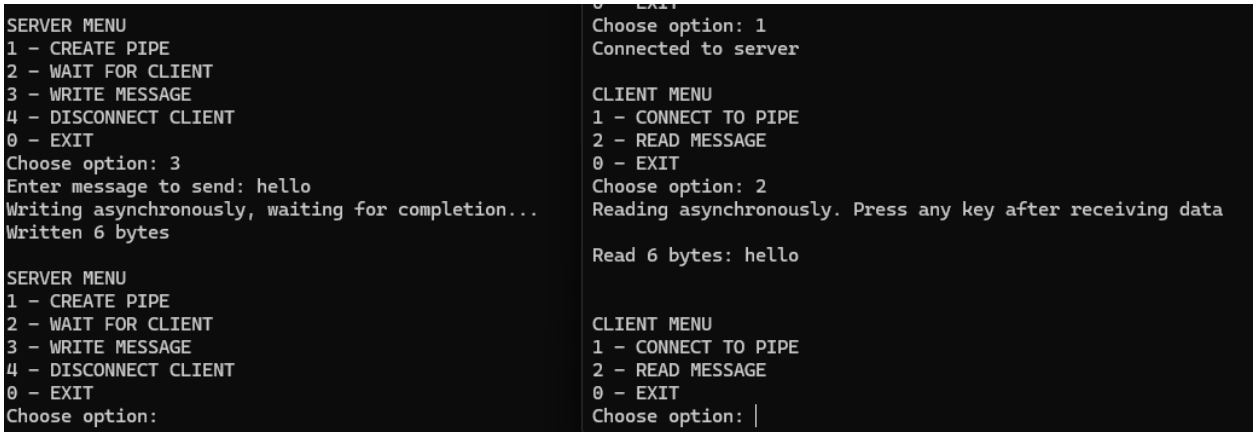
На стороне клиента:

подключение к каналу выполняется через CreateFile,

чтение из канала производится асинхронно с помощью ReadFileEx,
при завершении чтения вызывается callback-функция (ReadCompleted), указанная при вызове ReadFileEx,
ожидание завершения операции реализовано через SleepEx с alertable-состоянием потока.

Для уведомления о завершении операций используется событийный механизм (CreateEvent) и структура OVERLAPPED. Это позволяет выполнять ввод-вывод в фоновом режиме без блокировки основного потока.

Пример работоспособности программы



Заключение

В рамках данного задания был реализован альтернативный механизм межпроцессного взаимодействия — через именованные каналы (Named Pipes). По сравнению с разделяемой памятью, данный подход отличается более простой реализацией: не требуется ручное управление состояниями и синхронизацией доступа.

Код программы

server.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PIPE_NAME "\\\\.\\pipe\\PipeOS"
#define BUFFER_SIZE 4096

static HANDLE hPipe = INVALID_HANDLE_VALUE;
static HANDLE hConnectEvent = NULL;
static OVERLAPPED ovWrite = {0};

int enter_integer(const char *message, int a, int b) {
    int number;
    char input[64];
    while (1) {
        printf("%s", message);
        if (!fgets(input, sizeof(input), stdin)) {
            printf("Input interrupted. Exiting...\n");
```

```
            exit(1);
        }
        if (sscanf(input, "%d", &number) == 1 && number >= a && number <= b) {
            return number;
        } else {
            printf("Invalid input. Enter a number in range [%d, %d]\n", a, b);
        }
    }
}

void create_pipe() {
    if (hPipe != INVALID_HANDLE_VALUE) {
        printf("Pipe already created\n");
        return;
    }
    hPipe = CreateNamedPipeA(PIPE_NAME, PIPE_ACCESS_OUTBOUND | FILE_FLAG_OVERLAPPED, PIPE_TYPE_MESSAGE
| PIPE_READMODE_MESSAGE | PIPE_WAIT, 1, BUFFER_SIZE, BUFFER_SIZE, 0, NULL);
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("CreateNamedPipe error: %lu\n", GetLastError());
    } else {
        printf("Pipe created\n");
    }
}

void wait_for_client() {
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("Pipe not created\n");
        return;
    }
    hConnectEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    OVERLAPPED ov = {0};
    ov.hEvent = hConnectEvent;

    if (!ConnectNamedPipe(hPipe, &ov)) {
        DWORD err = GetLastError();
        if (err == ERROR_IO_PENDING) {
            printf("Waiting for client...\n");
            WaitForSingleObject(hConnectEvent, INFINITE);
            printf("Client connected\n");
        } else if (err == ERROR_PIPE_CONNECTED) {
            printf("Client already connected\n");
        } else {
            printf("ConnectNamedPipe error: %lu\n", err);
        }
    }
}

void write_message() {
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("No pipe available\n");
        return;
    }

    char buffer[BUFFER_SIZE];
    printf("Enter message to send: ");
    fgets(buffer, sizeof buffer, stdin);

    if (ovWrite.hEvent == NULL)
        ovWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    DWORD bytesWritten = 0;
    if (!WriteFile(hPipe, buffer, (DWORD)strlen(buffer), NULL, &ovWrite)) {
        if (GetLastError() != ERROR_IO_PENDING) {
            printf("WriteFile error: %lu\n", GetLastError());
            return;
        }
    }

    printf("Writing asynchronously, waiting for completion...\n");
```

```
    WaitForSingleObject(ovWrite.hEvent, INFINITE);
    GetOverlappedResult(hPipe, &ovWrite, &bytesWritten, FALSE);
    printf("Written %lu bytes\n", bytesWritten);
    ResetEvent(ovWrite.hEvent);
}

void disconnect_client() {
    if (DisconnectNamedPipe(hPipe)) {
        printf("Client disconnected\n");
    } else {
        printf("DisconnectNamedPipe error: %lu\n", GetLastError());
    }
}

void cleanup() {
    if (hPipe != INVALID_HANDLE_VALUE) CloseHandle(hPipe);
    if (hConnectEvent) CloseHandle(hConnectEvent);
    if (ovWrite.hEvent) CloseHandle(ovWrite.hEvent);
}

int main() {
    int choice;
    do {
        printf("\nSERVER MENU\n");
        printf("1 - CREATE PIPE\n");
        printf("2 - WAIT FOR CLIENT\n");
        printf("3 - WRITE MESSAGE\n");
        printf("4 - DISCONNECT CLIENT\n");
        printf("0 - EXIT\n");
        choice = enter_integer("Choose option: ", 0, 4);

        switch (choice) {
            case 1: create_pipe(); break;
            case 2: wait_for_client(); break;
            case 3: write_message(); break;
            case 4: disconnect_client(); break;
            case 0: break;
            default: printf("Invalid option\n"); break;
        }
    } while (choice != 0);

    cleanup();
    return 0;
}
```

client.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PIPE_NAME "\\\\.\\pipe\\PipeOS"
#define BUFFER_SIZE 4096

static HANDLE hPipe = INVALID_HANDLE_VALUE;
static char buffer[BUFFER_SIZE];
static OVERLAPPED ovRead = {0};

int enter_integer(const char *message, int a, int b) {
    int number;
    char input[64];
    while (1) {
        printf("%s", message);
        if (!fgets(input, sizeof(input), stdin)) {
            printf("Input interrupted. Exiting...\n");
```

```
            exit(1);
        }
        if (sscanf(input, "%d", &number) == 1 && number >= a && number <= b) {
            return number;
        } else {
            printf("Invalid input. Enter a number in range [%d, %d]\n", a, b);
        }
    }
}

VOID CALLBACK read_completed(DWORD err, DWORD bytes, LPOVERLAPPED lpOv) {
    if (err == 0) {
        printf("\nRead %lu bytes: %.*s\n", bytes, bytes, buffer);
    } else {
        printf("ReadFileEx error: %lu\n", err);
    }
}

void connect_pipe() {
    if (hPipe != INVALID_HANDLE_VALUE) {
        printf("Already connected\n");
        return;
    }
    hPipe = CreateFileA(PIPE_NAME, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_OVERLAPPED, NULL);
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("CreateFile error: %lu\n", GetLastError());
    } else {
        printf("Connected to server\n");
    }
}

void async_read() {
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("Not connected\n");
        return;
    }
    if (ovRead.hEvent == NULL)
        ovRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    memset(buffer, 0, sizeof buffer);
    if (!ReadFileEx(hPipe, buffer, sizeof buffer, &ovRead, read_completed)) {
        printf("ReadFileEx error: %lu\n", GetLastError());
        return;
    }

    printf("Reading asynchronously. Press any key after receiving data\n");
    SleepEx(INFINITE, TRUE);
}

void cleanup() {
    if (hPipe != INVALID_HANDLE_VALUE) CloseHandle(hPipe);
    if (ovRead.hEvent) CloseHandle(ovRead.hEvent);
}

int main() {
    int choice;
    do {
        printf("\nCLIENT MENU\n");
        printf("1 - CONNECT TO PIPE\n");
        printf("2 - READ MESSAGE\n");
        printf("0 - EXIT\n");
        choice = enter_integer("Choose option: ", 0, 2);

        switch (choice) {
            case 1: connect_pipe(); break;
            case 2: async_read(); break;
            case 0: break;
            default: printf("Invalid option\n"); break;
```

```
        }
    } while (choice != 0);

    cleanup();
    return 0;
}
```