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»

по теме «Межпроцессное взаимодействие»

|  |  |  |
| --- | --- | --- |
| Студентки гр. 2307 |  | Левушкина С.И. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург, 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_gjdgxs)

[ЗАДАНИЯ 3](#_30j0zll)

[1. Реализация решения задачи о читателях-писателях. 3](#_1fob9te)

[2. Использование именованных каналов для реализации 4](#_3znysh7)

[сетевого межпроцессного взаимодействия. 4](#_2et92p0)

[ЗАДАНИЕ 1 5](#_tyjcwt)

[Описание работы программ 5](#_3dy6vkm)

[Читатель и писатель: 5](#_1t3h5sf)

[Диспетчер 5](#_4d34og8)

[Примеры содержания лог-файлов 6](#_2s8eyo1)

[Графики состояний пяти читателей 7](#_17dp8vu)

[Графики состояний пяти писателей 9](#_3rdcrjn)

[Графики занятости первых пяти страниц (0-5) 11](#_26in1rg)

[Вывод 13](#_lnxbz9)

[Иcходные коды программ 13](#_1ksv4uv)

[Писатель 13](#_44sinio)

[Читатель 15](#_2jxsxqh)

[Менеджер запуска процессов 17](#_z337ya)

[ЗАДАНИЕ 2 20](#_3j2qqm3)

[Описание работы программ 20](#_1y810tw)

[Клиент 20](#_4i7ojhp)

[Сервер 20](#_2xcytpi)

[Пример работы программы 21](#_1ci93xb)

[Вывод 21](#_3whwml4)

[Исходные коды программ 22](#_2bn6wsx)

[Сервер 22](#_qsh70q)

[Клиент 24](#_3as4poj)

**ЦЕЛЬ РАБОТЫ**

Исследовать инструменты и механизмы взаимодействия процессов в Windows.

**ЗАДАНИЯ**

**1. Реализация решения задачи о читателях-писателях.**

1.1 Выполнить решение задачи о читателях-писателях, для чего необходимо разработать консольные приложения «Читатель» и «Писатель»:

а) одновременно запущенные экземпляры процессов-читателей и процессов-писателей должны совместно работать с буферной памятью в виде проецируемого файла:

- размер страницы буферной памяти равен размеру физической страницы оперативной памяти;

- число страниц буферной памяти равно сумме цифр в номере студенческого билета без учета первой цифры (3+7+1+5 = 16)

- страницы буферной памяти должны быть заблокированы в

оперативной памяти (функция VirtualLock);

- длительность выполнения процессами операций «чтения» и

«записи» задается случайным образом в диапазоне от 0,5 до 1,5 сек.;

- для синхронизации работы процессов необходимо использовать

объекты синхронизации типа «семафор» и «мьютекс»;

- процессы-читатели и процессы-писатели ведут свои журнальные

файлы, в которые регистрируют переходы из одного «состояния» в

другое (начало ожидания, запись или чтение, переход к

освобождению) с указанием кода времени (функция TimeGetTime).

Для состояний «запись» и «чтение» необходимо также

запротоколировать номер рабочей страницы.

1.2. Запустить приложения читателей и писателей, суммарное

количество одновременно работающих читателей и писателей должно

быть не менее числа страниц буферной памяти. Проверить

функционирование приложений, проанализируйте журнальные файлы

процессов, постройте сводные графики смены «состояний» для не менее 5 процессов-читателей и 5 процессов-писателей, дайте свои

комментарии относительно переходов процессов из одного состояния в другое. Построить графики занятости страниц буферной памяти (проецируемого файла) во времени, дайте свои комментарии.

**2. Использование именованных каналов для реализации**

**сетевого межпроцессного взаимодействия.**

2.1. Создать 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 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.

2.2. Запустить приложения и проверьте обмен данных между процессами. Запротоколировать результаты в отчет.

**ЗАДАНИЕ 1**

**Описание работы программ**

**Читатель и писатель:**

Каждая программа получает идентификатор текущего процесса и создает имя для файла журнала. Затем писатель открывает семафоры записи для каждой страницы и семафор читателей. Читатель – только семафор читателей.

Далее открывается объект проецируемого файла, который представляет общий буфер, и отображается его в память. Этот буфер блокируется в оперативной памяти, чтобы предотвратить его выгрузку.

В основном цикле писатели начинают эстафету – выбирают страницу и записывают, затем передают в общую память для номера страницы номер прочитанной и открывают семафор читателя.

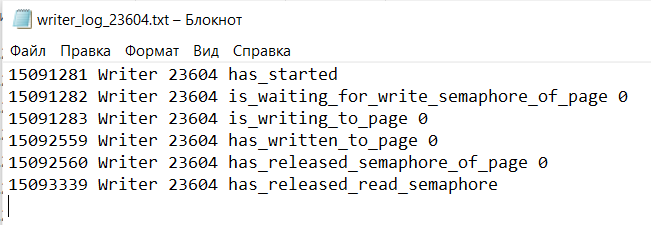
Далее читатель берет из общей памяти первый свободный номер страницы и читают с нее. И далее с 1 пункта по новой.

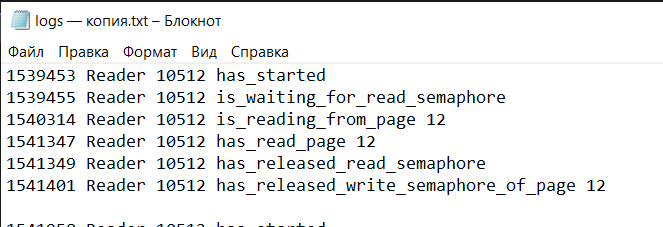
**Диспетчер**

Диспетчер создает семафор для читателей и семафоры для каждой страницы для писателей, далее использует пути к исполняемым файлам программ, запускает сначала 16 писателей, затем 16 читателей, делает задержку 10с и завершает все процессы.

**Примеры содержания лог-файлов**

время – тип процесса – номер процесса – операция – (страница)





**Графики состояний пяти читателей**

1 – has\_started

2 – is\_waiting\_for\_read\_semaphore

3 – is\_reading\_from\_page

4 – has\_read\_page

5 - has\_released\_read\_semaphore

6 - has\_released\_write\_semaphore\_of\_page











**Графики состояний пяти писателей**

1 – has\_started

2 – is\_waiting\_for\_write\_semaphore\_of\_page

3 – is\_reading\_from\_page

4 – has\_read\_from\_page

5 - has\_released\_read\_semaphore











**Графики занятости первых пяти страниц (0-5)**

7 – запись

8 – чтение











**Вывод**

На графиках состояний читателей и писателей видно, что все они ведут себя примерно одинаково - выполняют операции в порядке 1-2-…-6-1 и по новой. У всех наблюдается небольшая задержка на операции чтения/записи – она соответствует намеренно прописанной случайной паузе от 0.5 до 1.5 секунд.

На графиках занятости страниц видно чередование операций чтения и записи без наложения друг на друга, так как вероятно недостаточно читателей/писателей.

**Иcходные коды программ**

**Писатель**

#include <iostream>

#include <fstream>

#include <string>

#include <windows.h>

#include <ctime>

#include <random>

#include <thread>

#define \_CRT\_SECURE\_NO\_WARNINGS

using namespace std;

#pragma comment(lib, "winmm.lib")

const int PAGE\_SIZE = 4096;

const int NUM\_PAGES = 16;

const int BUFFER\_SIZE = PAGE\_SIZE \* NUM\_PAGES;

int rand\_delay() {

static default\_random\_engine engine(static\_cast<unsigned>(time(0)));

static uniform\_int\_distribution<int> distribution(500, 1500);

return distribution(engine);

}

void log(const string& filename, const string& message) {

ofstream logFile(filename, ios::app);

if (logFile.is\_open()) {

DWORD currentTime = timeGetTime();

logFile << currentTime << " " << message << endl;

logFile.close();

}

}

string getLogFilename(DWORD processId) {

string logFilename = "writer\_log\_" + to\_string(processId) + ".txt";

return logFilename;

}

int main(int argc, char\* argv[]) {

DWORD processId = GetCurrentProcessId();

string logFilename = getLogFilename(processId);

HANDLE fileMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, L"Global\\SharedBuffer");

if (fileMapping == nullptr) {

cerr << "Failed to open file mapping. Error: " << GetLastError() << endl;

return 1;

}

LPVOID buffer = MapViewOfFile(fileMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, BUFFER\_SIZE);

if (buffer == nullptr) {

cerr << "Failed to map view of file. Error: " << GetLastError() << endl;

return 1;

}

HANDLE pageNumberMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, L"Global\\PageNumber");

if (pageNumberMapping == nullptr) {

cerr << "Failed to open page number mapping. Error: " << GetLastError() << endl;

return 1;

}

int\* sharedPageNumber = (int\*)MapViewOfFile(pageNumberMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

if (sharedPageNumber == nullptr) {

cerr << "Failed to map view of page number. Error: " << GetLastError() << endl;

return 1;

}

HANDLE writeSemaphore;

HANDLE readSemaphore;

int cnt = 2;

while (true) {

srand(processId \* cnt);

log(logFilename, "Writer " + to\_string(processId) + " has\_started");

int page = rand() % NUM\_PAGES;

writeSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, (L"Global\\WriteSemaphore" + to\_wstring(page)).c\_str());

readSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, (L"Global\\ReadSemaphore"));

log(logFilename, "Writer " + to\_string(processId) + " is\_waiting\_for\_write\_semaphore\_of\_page " + to\_string(page));

WaitForSingleObject(writeSemaphore, INFINITE);

log(logFilename, "Writer " + to\_string(processId) + " is\_writing\_to\_page " + to\_string(page));

this\_thread::sleep\_for(chrono::milliseconds(rand\_delay()));

log(logFilename, "Writer " + to\_string(processId) + " has\_written\_to\_page " + to\_string(page));

\*sharedPageNumber = page;

log(logFilename, "Writer " + to\_string(processId) + " has\_released\_read\_semaphore");

ReleaseSemaphore(readSemaphore, 1, NULL);

this\_thread::sleep\_for(chrono::milliseconds(rand\_delay()));

cnt++;

}

UnmapViewOfFile(buffer);

UnmapViewOfFile(sharedPageNumber);

CloseHandle(fileMapping);

CloseHandle(pageNumberMapping);

CloseHandle(writeSemaphore);

CloseHandle(readSemaphore);

return 0;

}

**Читатель**

#include <iostream>

#include <fstream>

#include <string>

#include <windows.h>

#include <ctime>

#include <random>

#include <thread>

#define \_CRT\_SECURE\_NO\_WARNINGS

using namespace std;

#pragma comment(lib, "winmm.lib")

const int PAGE\_SIZE = 4096;

const int NUM\_PAGES = 16;

const int BUFFER\_SIZE = PAGE\_SIZE \* NUM\_PAGES;

int rand\_delay() {

static default\_random\_engine engine(static\_cast<unsigned>(time(0)));

static uniform\_int\_distribution<int> distribution(500, 1500);

return distribution(engine);

}

void log(const string& filename, const string& message) {

ofstream logFile(filename, ios::app);

if (logFile.is\_open()) {

DWORD currentTime = timeGetTime();

logFile << currentTime << " " << message << endl;

logFile.close();

}

}

string getLogFilename(DWORD processId) {

string logFilename = "reader\_log\_" + to\_string(processId) + ".txt";

return logFilename;

}

int main(int argc, char\* argv[]) {

DWORD processId = GetCurrentProcessId();

string logFilename = getLogFilename(processId);

HANDLE fileMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, L"Global\\SharedBuffer");

if (fileMapping == nullptr) {

cerr << "Failed to open file mapping. Error: " << GetLastError() << endl;

return 1;

}

LPVOID buffer = MapViewOfFile(fileMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, BUFFER\_SIZE);

if (buffer == nullptr) {

cerr << "Failed to map view of file. Error: " << GetLastError() << endl;

return 1;

}

HANDLE pageNumberMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, L"Global\\PageNumber");

if (pageNumberMapping == nullptr) {

cerr << "Failed to open page number mapping. Error: " << GetLastError() << endl;

return 1;

}

int\* sharedPageNumber = (int\*)MapViewOfFile(pageNumberMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

if (sharedPageNumber == nullptr) {

cerr << "Failed to map view of page number. Error: " << GetLastError() << endl;

return 1;

}

HANDLE writeSemaphore;

HANDLE readSemaphore;

while (true) {

log(logFilename, "Reader " + to\_string(processId) + " has\_started");

int page = \*sharedPageNumber;

if (page == -1) {

this\_thread::sleep\_for(chrono::milliseconds(rand\_delay()));

continue;

}

readSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, (L"Global\\ReadSemaphore"));

writeSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, (L"Global\\WriteSemaphore" + to\_wstring(page)).c\_str());

log(logFilename, "Reader " + to\_string(processId) + " is\_waiting\_for\_read\_semaphore\_of\_page " + to\_string(page));

WaitForSingleObject(readSemaphore, INFINITE);

log(logFilename, "Reader " + to\_string(processId) + " is\_reading\_page " + to\_string(page));

this\_thread::sleep\_for(chrono::milliseconds(rand\_delay()));

log(logFilename, "Reader " + to\_string(processId) + " has\_read\_page " + to\_string(page));

log(logFilename, "Reader " + to\_string(processId) + " has\_released\_read\_semaphore");

ReleaseSemaphore(readSemaphore, 1, NULL);

log(logFilename, "Reader " + to\_string(processId) + " has\_released\_write\_semaphore\_of\_page" + to\_string(page));

ReleaseSemaphore(writeSemaphore, 1, NULL);

this\_thread::sleep\_for(chrono::milliseconds(rand\_delay()));

}

UnmapViewOfFile(buffer);

UnmapViewOfFile(sharedPageNumber);

CloseHandle(fileMapping);

CloseHandle(pageNumberMapping);

CloseHandle(readSemaphore);

CloseHandle(writeSemaphore);

return 0;

}

**Диспетчер**

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

using namespace std;

const int NUM\_READERS = 16;

const int NUM\_WRITERS = 16;

const int NUM\_PAGES = 16;

const int WORK\_DURATION\_MS = 120000;

const int PAGE\_SIZE = 4096;

struct Page {

HANDLE writeSemaphore;

int pageNumber;

};

struct ProcessInfo {

PROCESS\_INFORMATION pi;

wstring name;

};

bool startProcess(const wstring& path, ProcessInfo& procInfo) {

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&procInfo.pi, sizeof(procInfo.pi));

wstring commandLine = path;

if (!CreateProcess(NULL, const\_cast<LPWSTR>(commandLine.c\_str()), NULL, NULL, FALSE, 0, NULL, NULL, &si, &procInfo.pi)) {

wcerr << L"Failed to start process: " << path << endl;

return false;

}

procInfo.name = path;

return true;

}

void terminateProcess(const ProcessInfo& procInfo) {

if (!TerminateProcess(procInfo.pi.hProcess, 0)) {

wcerr << L"Failed to terminate process: " << procInfo.name << L" Error: " << GetLastError() << endl;

}

CloseHandle(procInfo.pi.hProcess);

CloseHandle(procInfo.pi.hThread);

}

int main() {

vector<ProcessInfo> processes;

vector<Page> pages(NUM\_PAGES);

wstring readerPath = L"C:\\Users\\pipop\\source\\repos\\os\_4\_reader\\x64\\Debug\\os\_4\_reader.exe";

wstring writerPath = L"C:\\Users\\pipop\\source\\repos\\os\_4\_writer\\x64\\Debug\\os\_4\_writer.exe";

const int BUFFER\_SIZE = PAGE\_SIZE \* NUM\_PAGES;

HANDLE fileMapping = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, BUFFER\_SIZE, L"Global\\SharedBuffer");

if (fileMapping == nullptr) {

cerr << "Failed to create file mapping. Error: " << GetLastError() << endl;

return 1;

}

LPVOID buffer = MapViewOfFile(fileMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, BUFFER\_SIZE);

if (buffer == nullptr) {

cerr << "Failed to map view of file. Error: " << GetLastError() << endl;

return 1;

}

if (!VirtualLock(buffer, BUFFER\_SIZE)) {

cerr << "Failed to lock buffer in memory. Error: " << GetLastError() << endl;

return 1;

}

HANDLE readSemaphore = CreateSemaphore(NULL, 0, 1, (L"Global\\ReadSemaphore"));

for (int i = 0; i < NUM\_PAGES; ++i) {

pages[i].pageNumber = i;

pages[i].writeSemaphore = CreateSemaphore(NULL, 1, 1, (L"Global\\WriteSemaphore" + to\_wstring(i)).c\_str());

if (pages[i].writeSemaphore == nullptr || readSemaphore == nullptr) {

cerr << "Failed to create semaphores for page " << i << ". Error: " << GetLastError() << endl;

return 1;

}

}

HANDLE pageNumberMapping = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, sizeof(int), L"Global\\PageNumber");

if (pageNumberMapping == nullptr) {

cerr << "Failed to create page number mapping. Error: " << GetLastError() << endl;

return 1;

}

int\* sharedPageNumber = (int\*)MapViewOfFile(pageNumberMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

if (sharedPageNumber == nullptr) {

cerr << "Failed to map view of page number. Error: " << GetLastError() << endl;

return 1;

}

\*sharedPageNumber = -1;

for (int i = 0; i < NUM\_WRITERS; ++i) {

ProcessInfo procInfo;

if (startProcess(writerPath, procInfo)) {

processes.push\_back(procInfo);

}

}

for (int i = 0; i < NUM\_READERS; ++i) {

ProcessInfo procInfo;

if (startProcess(readerPath, procInfo)) {

processes.push\_back(procInfo);

}

}

wcout << L"Started " << NUM\_READERS << L" readers and " << NUM\_WRITERS << L" writers." << endl;

Sleep(WORK\_DURATION\_MS);

for (const auto& procInfo : processes) {

terminateProcess(procInfo);

}

wcout << L"Terminated all processes." << endl;

VirtualUnlock(buffer, BUFFER\_SIZE);

for (auto& page : pages) {

CloseHandle(page.writeSemaphore);

}

CloseHandle(fileMapping);

UnmapViewOfFile(buffer);

UnmapViewOfFile(sharedPageNumber);

CloseHandle(pageNumberMapping);

CloseHandle(readSemaphore);

return 0;

}

**ЗАДАНИЕ 2**

**Описание работы программ**

**Клиент**

Пользователь видит меню с тремя опциями: установить соединение, прочитать данные, завершить работу.

1: Вызывается CreateFile, создается канал и выполняется подключение. Если соединение успешно, дескриптор канала сохраняется в переменной hPipe, иначе – выводится сообщение об ошибке.

2: Вызывается ReadFileEx, читаются данные (либо выводистя сообщение об ошибке) и вызывается SleepEx, чтобы приостановить выполнение до завершения асинхронной операции.

3: Вызывается CloseHandle(hPipe) и соединение завершается.

**Сервер**

Пользователь видит меню с пятью опциями: создать именованный канал, установить соединение, записать данные, разорвать соединение, завершить работу.

1: Вызывается CreateNamedPipe с именем канала "\\\\.\\pipe\\my\_pipe", а затем CreateEvent для создания события, связанного с асинхронными операциями.

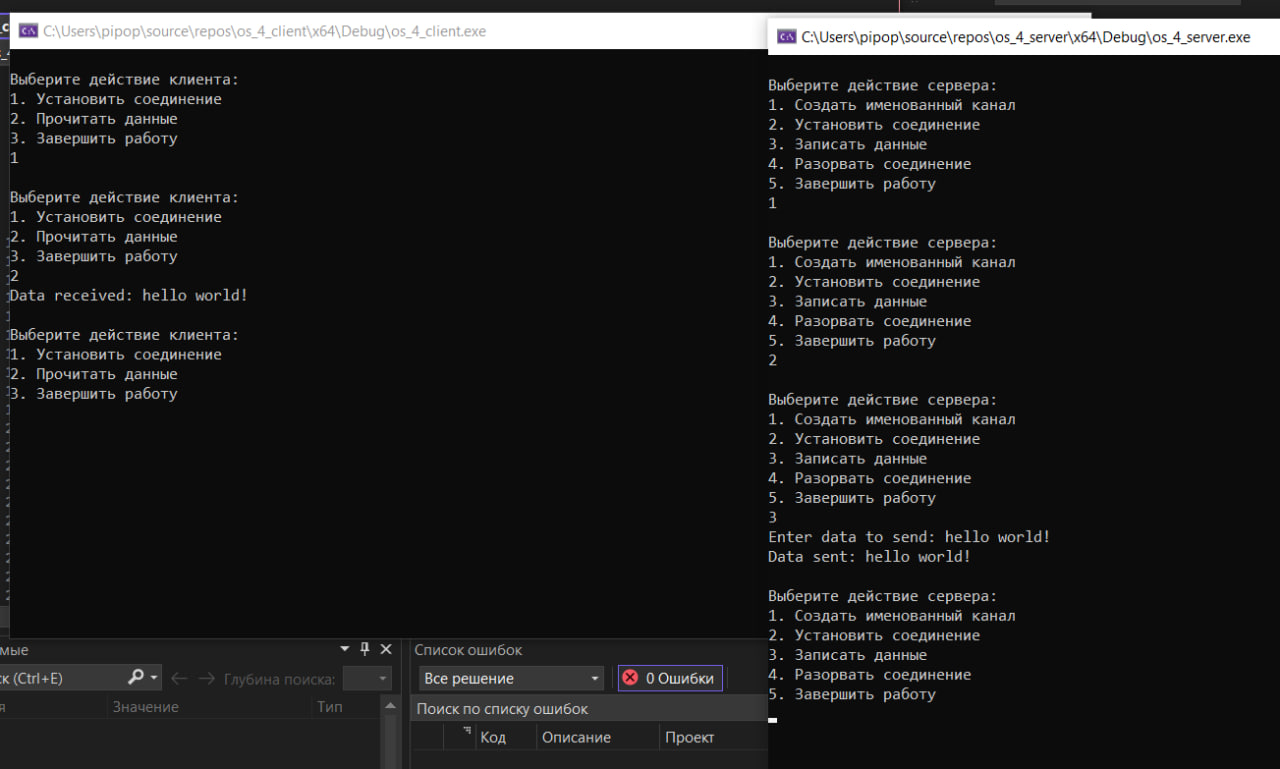
2: Вызывается ConnectNamedPipe

3: Вызывается WriteFile и записываются введенные с клавиатуры данные (размер буфера – 1024)

4: Вызывается DisconnectNamedPipe и соединение завершается.

5: Вызываются CloseHandle(hPipe) и CloseHandle(hEvent)Ю закрываются дескрипторы канала и события.

**Пример работы программы**



**Вывод**

Было проверено установление подключение между приложением-сервером и приложением-клиентом, а также была проверена связь между двумя приложениями, в результате чего сообщение, записанное сервером было успешно передано клиенту с использованием именованного канала my\_pipe. Это подтверждает асинхронный ввод/вывод по каналам связи. Таким образом и были использованы именованные каналы для реализации сетевого межпроцессорного взаимодействия

**Исходные коды программ**

**Сервер**

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

HANDLE hPipe, hEvent;

OVERLAPPED overlapped = { 0 };

char buffer[1024];

void createPipe() {

hPipe = CreateNamedPipe(

L"\\\\.\\pipe\\my\_pipe",

PIPE\_ACCESS\_OUTBOUND | FILE\_FLAG\_OVERLAPPED, // режим открытия

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

PIPE\_UNLIMITED\_INSTANCES, // максимальное количество экземпляров

1024, // размер выходного буфера

1024, // размер входного буфера

0, // тайм-аут

NULL); // атрибуты безопасности

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

cerr << "Ошибка создания канала: " << GetLastError() << endl;

return;

}

// Создание события

hEvent = CreateEvent(

NULL, // атрибуты безопасности

TRUE, // ручной сброс (manual reset)

FALSE, // начальное состояние - не сигнальное (nonsignaled)

NULL); // имя события

if (hEvent == NULL) {

cerr << "CreateEvent failed, GLE=" << GetLastError() << endl;

return;

}

}

void connectPipe() {

// Подключение к существующему именованному каналу

hPipe = CreateFile(

L"\\\\.\\pipe\\my\_pipe",

GENERIC\_WRITE, // режим доступа

0, // режим совместного доступа

NULL, // атрибуты безопасности

OPEN\_EXISTING, // как открыть

FILE\_FLAG\_OVERLAPPED, // атрибуты

NULL); // шаблон файла

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

cerr << "Ошибка подключения: " << GetLastError() << endl;

return;

}

}

void writeData() {

cout << "Введите данные для записи: ";

string data;

getline(cin, data);

DWORD bytesWritten;

overlapped.hEvent = hEvent;

// Попытка записать данные в канал асинхронно

BOOL result = WriteFile(

hPipe,

data.c\_str(),

data.length(),

&bytesWritten,

&overlapped);

if (!result && GetLastError() == ERROR\_IO\_PENDING) {

// Ожидание принятия данных клиентом

WaitForSingleObject(hEvent, INFINITE);

}

}

void menu() {

int choice;

do {

cout << "\nВыберите действие сервера: \n";

cout << "1. Создать именованный канал\n";

cout << "2. Установить соединение\n";

cout << "3. Записать данные\n";

cout << "4. Разорвать соединение\n";

cout << "5. Завершить работу\n";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

createPipe();

break;

case 2:

connectPipe();

break;

case 3:

writeData();

break;

case 4:

disconnectPipe();

break;

case 5:

CloseHandle(hPipe);

CloseHandle(hEvent);

break;

default:

cout << "Недопустимое значения выбора\n";

}

} while (choice != 5);

}

int main() {

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}

**Клиент**

#include <windows.h>

#include <iostream>

using namespace std;

HANDLE hPipe;

OVERLAPPED overlapped = { 0 };

char buffer[1024];

// Завершенная процедура для асинхронного чтения

VOID CALLBACK readCompletionRoutine(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {

if (dwErrorCode == 0) {

cout << "Прочитанные данные: " << buffer << endl;

}

else {

cerr << "Ошибка чтения: " << dwErrorCode << endl;

}

}

void connectPipe() {

// Подключение к существующему именованному каналу

hPipe = CreateFile(

L"\\\\.\\pipe\\my\_pipe",

GENERIC\_READ, // режим доступа

0, // режим совместного доступа

NULL, // атрибуты безопасности

OPEN\_EXISTING, // как открыть

FILE\_FLAG\_OVERLAPPED, // атрибуты

NULL); // шаблон файла

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

cerr << "Ошибка подключения: " << GetLastError() << endl;

return;

}

}

void readData() {

DWORD bytesRead;

// Создание события для асинхронной операции

overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

if (overlapped.hEvent == NULL) {

cerr << "CreateEvent failed: " << GetLastError() << endl;

return;

}

// Асинхронное чтение данных из канала

BOOL result = ReadFileEx(

hPipe,

buffer,

sizeof(buffer),

&overlapped,

readCompletionRoutine);

if (!result) {

cerr << "Ошибка чтения : " << GetLastError() << endl;

}

SleepEx(INFINITE, TRUE);

}

void menu() {

int choice;

do {

cout << "\nВыберите действие клиента: \n";

cout << "1. Установить соединение\n";

cout << "2. Прочитать данные\n";

cout << "3. Завершить работу\n";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

connectPipe();

break;

case 2:

readData();

break;

case 3:

// Закрытие дескриптора канала

CloseHandle(hPipe);

break;

default:

cout << "Недопустимое значение выбора\n";

}

} while (choice != 3);

}

int main() {

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}