Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №5

на тему

Взаимодействие процессов: обмен данными

Студент П. С. Павлюткин

Преподаватель С. И. Сиротко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель работы изучить и применить на практике подходы, системные объекты и функции для обеспечения передачи данных между взаимодействующими процессами и совместной их обработки, научиться решать типичные проблемы, возникающие при организации взаимодействия, и пути их решения.Для достижения цели будет создано приложение моделирующее распределенную обработку (вычислений).

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Задача обмена данными (совместного использования данных) – одна из

задач взаимодействия процессов. Преодоление изоляции процессов в многозадачной системе.

Необходимость решать также проблемы согласованного доступа к данным: путем копирования данных при передаче между процессами либо синхронизации (взаимного исключения). Использование специализированных средств IPC для обмена данными:

каналы и подобные им объекты, разделяемая память, очередь сообщений.

Взаимодействие процессов - это ключевой аспект в современных операционных системах и программировании. Процессы могут быть независимыми сущностями, работающими параллельно, и взаимодействие между ними часто необходимо для решения различных задач. Одним из основных методов взаимодействия процессов является обмен данными. Вот некоторые теоретические аспекты этой темы:

Процессы могут требовать обмена данными для передачи информации, синхронизации, координации и выполнения распределенных задач. При обмене данными между процессами важно обеспечивать безопасность и конфиденциальность передаваемой информации, особенно в распределенных системах. Обеспечивают двунаправленную связь между процессами, работающими на одной машине.

Позволяют взаимодействовать между процессами, работающими на разных машинах в сети. Используются для синхронизации доступа к ресурсам между процессами. Позволяют посылать сообщения между процессами. Позволяет нескольким процессам разделять общую область памяти. Процессы в операционных системах имеют свои адресные пространства, и для взаимодействия между ними требуются специальные механизмы. Операционные системы обеспечивают множество средств для межпроцессорного взаимодействия, включая средства синхронизации, обмена сообщениями и разделяемую память.

Возникают, когда два или более процесса пытаются изменить общий ресурс одновременно. Ситуации, когда несколько процессов блокируют друг друга, ожидая освобождения ресурсов. Задача предотвращения одновременного доступа нескольких процессов к общему ресурсу. Увеличение производительности, повышение отказоустойчивости и повышение масштабируемости за счет использования ресурсов нескольких вычислительных узлов. Распределенные базы данных, облачные вычисления, распределенные вычислительные сети.

Взаимодействие между процессами и обмен данными - это обширная и важная тема, и выбор методов зависит от конкретных требований приложения. Он играет ключевую роль в разработке многозадачных, многопоточных и распределенных систем.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение, моделирующее распределенную обработку (вычислений). (Рисунок 1).

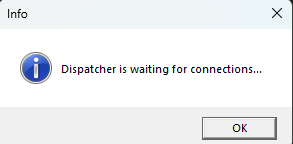


Рисунок 1 – Окно приложения диспетчера

Данная программа реализует взаимодействие между двумя процессами процессом-диспетчером и процессом-рабочим с использованием именованного канала (*Named Pipe*) в среде Windows. Вот основные шаги, которые выполняет программа:

Процесс-диспетчер:

Создает именованный канал с использованием функции `*CreateNamedPipe*`.

Ожидает подключения процесса-рабочего к созданному каналу с использованием функции `*ConnectNamedPipe*`.

После успешного подключения, отправляет задание (в данном случае, строку "*YourTaskDataHere*") в процесс-рабочий через именованный канал.

Ждет ответ от процесса-рабочего через тот же канал и выводит результат в диалоговом окне.

Процесс-рабочий:

Пытается открыть именованный канал с использованием функции `*CreateFile*`.

Если успешно подключается к каналу, считывает данные из канала (в данном случае, строку "*YourTaskDataHere*"), и, возможно, выполняет некоторую обработку данных.

Отправляет обработанные данные обратно в процесс-диспетчер через тот же именованный канал с использованием функции `*WriteFile*`.

Оба процесса используют функции `*MessageBox*` для отображения информации в диалоговом окне *Windows*.

Эта программа представляет собой простой пример распределенной обработки данных, где процесс-диспетчер выдает задание процессу-рабочему, а тот возвращает результат обратно. В реальных приложениях это может использоваться, например, для распределенных вычислений, обработки данных в разных частях системы, или для организации взаимодействия между компонентами программы, работающими в разных процессах. (Рисунок 2).

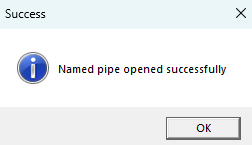


Рисунок 2– Выход за игровое поле

Наше приложение демонстрирует то, как можно моделировать распределенную обработку (вычислений).

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были исследованы и применены методы взаимодействия процессов с использованием технологии именованных каналов в среде *Windows*, а также библиотеки *WinAPI* и языка программирования *C++*. Работа включала в себя моделирование распределенной обработки данных, в которой взаимодействие между процессами выполнялось с целью обмена заданиями и результатами.

Проект включает в себя два типа процессов: процесс-диспетчер и рабочий процесс. Процесс-диспетчер отвечал за создание и управление объектами *IPC* (в данном случае, именованным каналом), выдачу заданий рабочим процессам и сбор результатов. Рабочий процесс, в свою очередь, принимал задание, выполнял его и возвращал результат.

Были использованы именованные каналы для обеспечения двунаправленного обмена данными между процессами. В процессе разработки уделено внимание обработке ошибок, таким образом, для детального контроля и обеспечения безопасности использовались диалоговые окна с сообщениями об ошибках и информацией о ходе выполнения программы.

Эта лабораторная работа предоставила практический опыт в реализации взаимодействия между процессами в рамках операционной системы *Windows*, а также позволила применить знания *WinAPI* для создания и управления процессами, обеспечивая эффективный обмен данными и координацию между ними. Эти навыки являются важными в контексте разработки распределенных систем и приложений, требующих взаимодействия между отдельными компонентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. *Win32 API*. Разработка приложений для *Windows*. ─ СПБ: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для *Windows* с использованием *API Win32* [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab5(Dispatcher).cpp

#include <windows.h>

#include <string>

#include <iostream>

const int BUFFER\_SIZE = 256;

const LPCWSTR PIPE\_NAME = L"\\\\.\\pipe\\MyNamedPipe";

void DispatchTask(HANDLE hPipe, const char\* taskData) {

// Конвертация char[] в std::wstring

std::wstring wTaskData = std::wstring(taskData, taskData + strlen(taskData));

DWORD bytesWritten;

if (!WriteFile(hPipe, wTaskData.c\_str(), (wTaskData.size() + 1) \* sizeof(wchar\_t), &bytesWritten, nullptr)) {

MessageBox(nullptr, L"Error writing to pipe", L"Error", MB\_ICONERROR);

return;

}

wchar\_t resultData[BUFFER\_SIZE];

DWORD bytesRead;

if (!ReadFile(hPipe, resultData, BUFFER\_SIZE, &bytesRead, nullptr)) {

MessageBox(nullptr, L"Error reading from pipe", L"Error", MB\_ICONERROR);

return;

}

// Конвертация wchar\_t[] в std::wstring

std::wstring wResultData(resultData, resultData + bytesRead / sizeof(wchar\_t));

// Формирование строки результата

std::wstring resultString = L"Dispatcher received result: " + wResultData;

MessageBox(nullptr, resultString.c\_str(), L"Result", MB\_ICONINFORMATION);

CloseHandle(hPipe);

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

while (true) {

HANDLE hNamedPipe = CreateNamedPipe(PIPE\_NAME, PIPE\_ACCESS\_DUPLEX, PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, PIPE\_UNLIMITED\_INSTANCES, BUFFER\_SIZE, BUFFER\_SIZE, 0, nullptr);

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

MessageBox(nullptr, L"Error creating named pipe", L"Error", MB\_ICONERROR);

return 1;

}

MessageBox(nullptr, L"Dispatcher is waiting for connections...", L"Info", MB\_ICONINFORMATION);

else {

MessageBox(nullptr, L"Error connecting to named pipe", L"Error", MB\_ICONERROR);

}

CloseHandle(hNamedPipe);

}

return 0;

}

Листинг 2 – Файл Lab5(Worker).cpp

#include <windows.h>

#include <string>

const int BUFFER\_SIZE = 256;

const LPCWSTR PIPE\_NAME = L"\\\\.\\pipe\\MyNamedPipe";

void ProcessTask(HANDLE hPipe) {

char buffer[BUFFER\_SIZE];

DWORD bytesRead;

if (ReadFile(hPipe, buffer, BUFFER\_SIZE, &bytesRead, nullptr)) {

// Печать прочитанных данных

buffer[bytesRead] = '\0';

MessageBoxA(nullptr, buffer, "Read Data", MB\_ICONINFORMATION);

// Можете выполнить здесь обработку данных по заданию

if (!WriteFile(hPipe, buffer, bytesRead, nullptr, nullptr)) {

MessageBox(nullptr, L"Error writing to pipe", L"Error", MB\_ICONERROR);

}

else {

MessageBoxA(nullptr, buffer, "Written Data", MB\_ICONINFORMATION);

}

}

else {

MessageBox(nullptr, L"Error reading from pipe", L"Error", MB\_ICONERROR);

}

CloseHandle(hPipe);

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

HANDLE hNamedPipe = CreateFile(PIPE\_NAME, GENERIC\_READ | GENERIC\_WRITE, 0, nullptr, OPEN\_EXISTING, 0, nullptr);

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

DWORD error = GetLastError();

MessageBox(nullptr, (L"Error opening named pipe. Error code: " + std::to\_wstring(error)).c\_str(), L"Error", MB\_ICONERROR);

return 1;

}

else {

MessageBox(nullptr, L"Named pipe opened successfully", L"Success", MB\_ICONINFORMATION);

}

ProcessTask(hNamedPipe);

CloseHandle(hNamedPipe);

return 0;

}