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

ОТЧЁТ

к лабораторной работе №3

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ: ОБМЕН ДАННЫМИ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc182478681)

[1.1 Постановка задачи 3](#_Toc182478682)

[1.2 Демонстрация работы программы 3](#_Toc182478683)

[Заключение 5](#_Toc182478684)

[Приложение А (справочное) Исходный код  
 (к пункту 1.2) 7](#_Toc182478685)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

* 1. **Постановка задачи**

В рамках данной задачи необходимо разработать приложение, демонстрирующее работу многозадачного комплекса, которое обеспечит обмен данными между несколькими процессами с использованием механизмов межпроцессного взаимодействия (IPC [1]). Целью является преодоление изоляции процессов в многозадачной системе, что позволит осуществлять совместное использование данных. Важно обеспечить согласованный доступ к данным, что может быть достигнуто через копирование данных при их передаче или путем синхронизации процессов с использованием взаимного исключения. Для реализации данной задачи будут использоваться специализированные средства IPC, такие как именованные и неименованные каналы, очереди сообщений и разделяемая память.

Приложение должно включать два или более процессов, которые будут выполнять последовательные этапы обработки данных, что можно представить в виде конвейерной модели. В качестве примера обработки могут быть использованы статистические расчеты, сортировка, шифрование или математические вычисления. Основное внимание будет уделено технической реализации, а не качествам математической модели. Процессы должны передавать данные по цепочке конвейера, что позволит анализировать корректность работы системы, выявлять коллизии и оценивать эффективность выбранных механизмов IPC. Приложение может быть как оконным, так и консольным, в зависимости от предпочтений разработчика.

Для достижения этой цели могут быть использованы следующие функции: *CreateNamedPipe* [2], *WriteFile* [3], *ReadFile* [4].

**1.2 Демонстрация работы программы**

Генератор при запуске ожидает подключение процессора (рисунок 1.1).

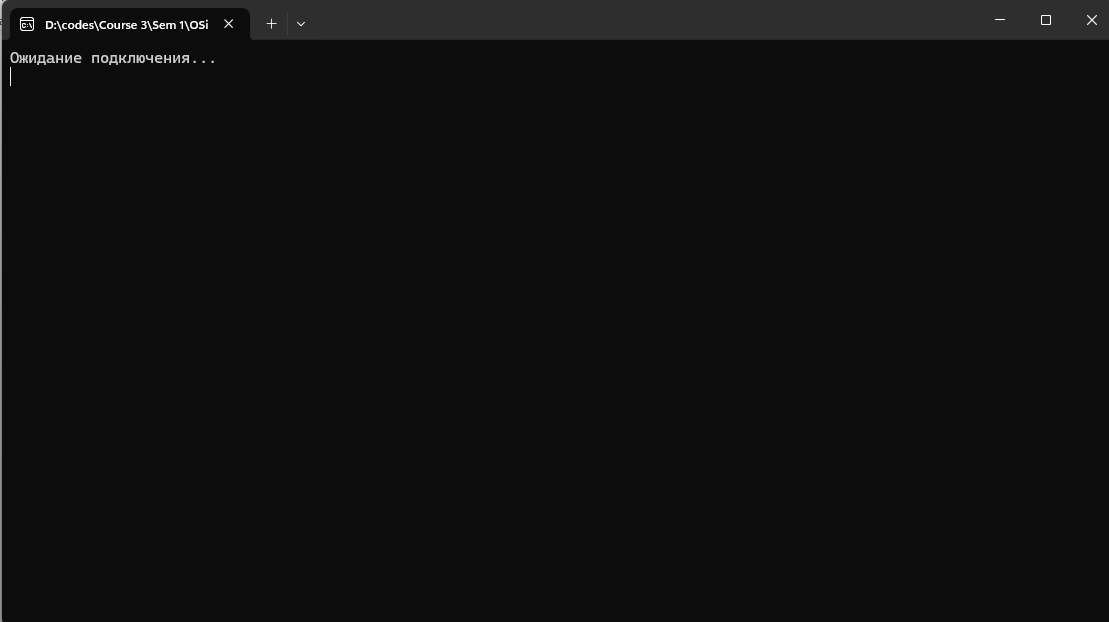


Рисунок 1.1 – Запуск генератора

При запуске процессора, генератор начинает передавать данные в процессор (рисунок 1.2), а процессор обрабатывает их (рисунок 1.3).

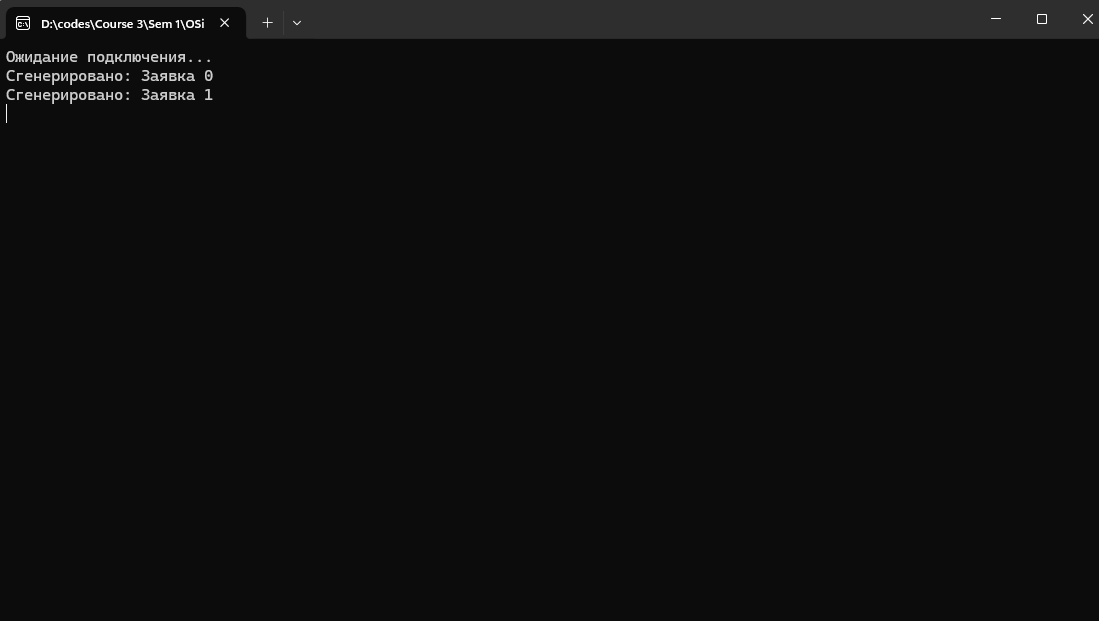


Рисунок 1.2 – Генерация данных

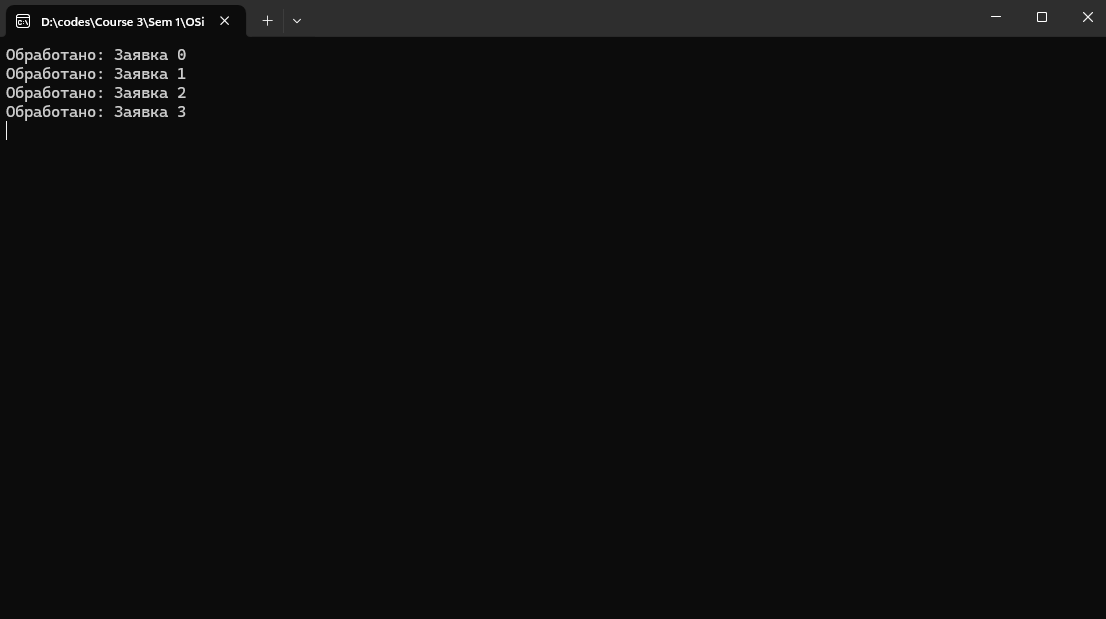


Рисунок 1.3 – Обработка данных

**ЗАКЛЮЧЕНИЕ**

В заключение, разработка приложения для демонстрации многозадачного взаимодействия и обмена данными между процессами является важной задачей, необходимой для понимания принципов межпроцессного взаимодействия и управления совместным доступом к данным. Использование механизмов IPC, таких как именованные каналы и очереди сообщений, позволяет эффективно организовать передачу данных, преодолевая изоляцию процессов и обеспечивая согласованный доступ.

Реализация конвейерной обработки данных не только иллюстрирует практическое применение теоретических концепций, но и позволяет оценить корректность работы системы через анализ возможных коллизий. Выбор подходящих инструментов и методов взаимодействия является ключевым для достижения высокой производительности и надежности приложения. Таким образом, выполненная работа способствует углублению знаний в области операционных систем и межпроцессного взаимодействия, а также развивает навыки программирования и проектирования многозадачных систем.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Файлы, отображаемые в память [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/ipc/interprocess-communications>.

[2] CreateNamedPipeA function (winbase.h) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createnamedpipea.

[3] WriteFile function (fileapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile>.

[4] ReadFile function (fileapi.h) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

Листинг А.1 – Код генератора

#include <windows.h>

#include <iostream>

#include <string>

#include <thread>

#include <vector>

#include <chrono>

const char\* pipeName = "\\\\.\\pipe\\my\_pipe";

void processRequest(const std::string& data) {

std::cout << "Обрабатывается: " << data << std::endl;

std::this\_thread::sleep\_for(std::chrono::seconds(1));

}

int main() {

setlocale(LC\_ALL, "Russian");

HANDLE hPipe = CreateNamedPipeA(

pipeName,

PIPE\_ACCESS\_OUTBOUND,

PIPE\_TYPE\_BYTE | PIPE\_WAIT,

PIPE\_UNLIMITED\_INSTANCES,

1024,

1024,

0,

NULL);

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

std::cerr << "Ошибка создания канала." << std::endl;

return 1;

}

std::cout << "Ожидание подключения..." << std::endl;

ConnectNamedPipe(hPipe, NULL);

for (int i = 0; i < 10; ++i) {

std::string data = "Заявка " + std::to\_string(i);

DWORD written;

WriteFile(hPipe, data.c\_str(), data.size() + 1, &written, NULL);

std::cout << "Сгенерировано: " << data << std::endl;

std::this\_thread::sleep\_for(std::chrono::seconds(1));

}

CloseHandle(hPipe);

return 0;

}

Листинг А.1 – Код процессора

#include <windows.h>

#include <iostream>

#include <string>

#include <thread>

#include <vector>

const char\* pipeName = "\\\\.\\pipe\\my\_pipe";

void processRequest(const std::string& data) {

std::cout << "Обрабатывается: " << data << std::endl;

std::this\_thread::sleep\_for(std::chrono::seconds(1));

}

void readFromPipe(HANDLE hPipe) {

char buffer[128];

DWORD bytesRead;

while (ReadFile(hPipe, buffer, sizeof(buffer), &bytesRead, NULL) && bytesRead > 0) {

buffer[bytesRead] = '\0';

std::string data(buffer);

std::thread(processRequest, data).detach();

}

}

int main() {

setlocale(LC\_ALL, "Russian");

HANDLE hPipe = CreateFileA(

pipeName,

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

0,

NULL);

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

std::cerr << "Ошибка открытия канала." << GetLastError() << std::endl;

return 1;

}

readFromPipe(hPipe);

CloseHandle(hPipe);

return 0;

}