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**

Студент: Ющук И.А.

Преподаватель: Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177507035)

[2 Описание функций программы 4](#_Toc177507036)

[2.1 Создание и подключение к именованному каналу 4](#_Toc177507037)

[2.2 Шифрование и дешифрование данных 4](#_Toc177507038)

[2.3 Вывод данных 5](#_Toc177507039)

[Заключение 6](#_Toc177507040)

[Список использованных источников 7](#_Toc177507041)

[Приложение А (обязательное) Исходный код программы 8](#_Toc177507042)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Фактически – программная модель системы массового обслуживания с упрощениями: внимание к технической реализации, а не к качествам мат. модели. Два или более (до 4-5) процессов, соответствующим «ступеням» обработки, и передача обслуживаемых «заявок» по цепочке конвейера. Варианты реализуемой обработки: статистические расчеты, сортировка, поиск, криптография, математические вычисления и т.д. Варианты используемых IPC: именованные и неименованные каналы, почтовые ящики, очереди сообщений. Варианты организации цепочек (конвейеров): – генерация блоков данных – обработка – визуализация – генерация блоков – сортировка – сборка – вывод – генерация блоков – шифрование – вывод – дешифрование – вывод и т.д.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– функция создания и подключения к каналу;

– функция шифрования и дешифрования данных;

– функция вывода данных.

## **2.1** **Создание и подключение к именованному каналу**

При запуске программы открываются именованные каналы и главный процесс ожидает подключения других процессов. [1,2] (рисунок 2.1).

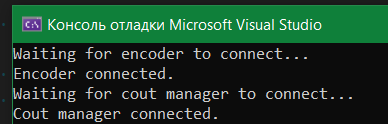
****

Рисунок 2.1 – Результат создания и подключения к именованному каналу.

## **2.2** **Шифрование и дешифрование данных**

Отдельный процесс шифрует данные передаваемые по именованному каналу с помощью метода шифровки исключающим ИЛИ (рисунок 2.2).

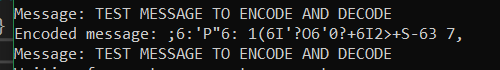


Рисунок 2.2 – Результат шифрования и дешифрования данных

## **2.3** **Вывод данных**

По условиям лабораторной работы необходимо было создать отдельный поток и через отдельный канал отправлять данные, которые процесс будет выводить в консоль. (рисунок 2.3).

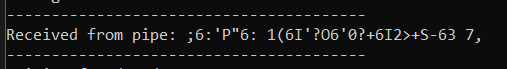


Рисунок 2.3 – Результат вывода данных

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была разработана программная модель системы массового обслуживания, сосредоточенная на технической реализации процессов обработки данных. Модель демонстрирует взаимодействие нескольких процессов, которые соответствуют различным этапам обработки заявок в рамках цепочки конвейера. В результате работы были реализованы варианты обработки данных, включая статистические расчеты, сортировку, поиск и криптографию.

Использование межпроцессорных коммуникаций (IPC), таких как именованные и неименованные каналы, почтовые ящики и очереди сообщений, обеспечило эффективную передачу данных между процессами, что является критически важным для функционирования системы. Разнообразие организации цепочек обработки, включая генерацию блоков данных, их обработку и визуализацию, показало гибкость и масштабируемость разработанной модели.

Работа над данной лабораторной задачей позволила не только углубить знания в области систем массового обслуживания и межпроцессорных коммуникаций, но и получить практические навыки в разработке и реализации программных решений, что имеет важное значение для будущих проектов в области программирования и разработки систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] ConnectNamedPipe function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-connectnamedpipe

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

using namespace std;

string xorCipher(string& text, string key) {

string result = text;

for (size\_t i = 0; i < text.size(); ++i) {

result[i] ^= key[i % key.size()];

}

return result;

}

int main() {

string namedPipeCoutManager = "\\\\.\\pipe\\coutPipe";

string namedPipeEncoder = "\\\\.\\pipe\\encoderPipe";

string namedPipeDecoder = "\\\\.\\pipe\\decoderPipe";

HANDLE hPipeEncoder = CreateNamedPipeA(

namedPipeEncoder.c\_str(), // Имя канала

PIPE\_ACCESS\_DUPLEX, // Чтение и запись

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, // Тип сообщения

PIPE\_UNLIMITED\_INSTANCES, // Максимальное количество инстанций

512, // Выходной буфер

512, // Входной буфер

0, // Тайм-аут

NULL // Без защиты

);

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

cerr << "CreateFile failed. Error: " << GetLastError() << endl;

return 1;

}

HANDLE hPipeDecoder = CreateNamedPipeA(

namedPipeDecoder.c\_str(), // Имя канала

PIPE\_ACCESS\_DUPLEX, // Чтение и запись

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, // Тип сообщения

PIPE\_UNLIMITED\_INSTANCES, // Максимальное количество инстанций

512, // Выходной буфер

512, // Входной буфер

0, // Тайм-аут

NULL // Без защиты

);

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

cerr << "CreateFile failed. Error: " << GetLastError() << endl;

return 1;

}

HANDLE hPipeCoutManager = CreateNamedPipeA(

namedPipeCoutManager.c\_str(), // Имя канала

PIPE\_ACCESS\_DUPLEX, // Чтение и запись

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, // Тип сообщения

PIPE\_UNLIMITED\_INSTANCES, // Максимальное количество инстанций

512, // Выходной буфер

512, // Входной буфер

0, // Тайм-аут

NULL // Без защиты

);

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

cerr << "CreateNamedPipe failed. Error: " << GetLastError() << endl;

return 1;

}

string encoderPath = "D:\\253505\\5sem\\OSiSP\\LR3\\LR3\\x64\\Debug\\Encoder.exe";

PROCESS\_INFORMATION pi;

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

wstring encoderPathModefied(encoderPath.begin(), encoderPath.end());

if (!CreateProcess(

&encoderPathModefied[0], // Путь к исполняемому файлу

NULL, // Командная строка

NULL, // Указатель на процесс

NULL, // Указатель на поток

FALSE, // Наследовать дескрипторы

NULL, // Флаги создания

NULL, // Переменные окружения

NULL, // Рабочий каталог

&si, // Структура информации о запуске

&pi)) // Структура информации о процессе

{

cerr << "CreateProcess failed (" << GetLastError() << ").\n";

return -1;

}

cout << "Waiting for encoder to connect..." << endl;

BOOL result = ConnectNamedPipe(hPipeEncoder, NULL);

if (!result) {

cerr << "ConnectNamedPipe failed. Error: " << GetLastError() << endl;

CloseHandle(hPipeEncoder);

return 1;

}

cout << "Encoder connected." << endl;

DWORD bytesWritten;

string buffer("TEST MESSAGE TO ENCODE AND DECODE");

/\* string key("osisp");

cout << "Message: " << buffer<<'\n';

string encoded = xorCipher(buffer, key);

cout << "Encoded message: " << encoded << '\n';

cout << "Message: " << xorCipher(encoded, key) << '\n';\*/

WriteFile(hPipeEncoder, &buffer[0], buffer.size()+1, &bytesWritten, NULL);

DWORD bytesRead;

ReadFile(hPipeEncoder, &buffer[0], buffer.size() + 1, &bytesRead, NULL);

//cout << "Received from encoder: "<< buffer << endl;

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

string coutManagerPath = "D:\\253505\\5sem\\OSiSP\\LR3\\LR3\\x64\\Debug\\CoutManager.exe";

wstring coutManagerPathModefied(coutManagerPath.begin(), coutManagerPath.end());

if (!CreateProcess(

&coutManagerPathModefied[0], // Путь к исполняемому файлу

NULL, // Командная строка

NULL, // Указатель на процесс

NULL, // Указатель на поток

FALSE, // Наследовать дескрипторы

NULL, // Флаги создания

NULL, // Переменные окружения

NULL, // Рабочий каталог

&si, // Структура информации о запуске

&pi)) // Структура информации о процессе

{

cerr << "CreateProcess failed (" << GetLastError() << ").\n";

return -1;

}

cout << "Waiting for cout manager to connect..." << endl;

result = ConnectNamedPipe(hPipeCoutManager, NULL);

if (!result) {

cerr << "ConnectNamedPipe failed. Error: " << GetLastError() << endl;

CloseHandle(hPipeCoutManager);

return 1;

}

cout << "Cout manager connected." << endl;

WriteFile(hPipeCoutManager, &buffer[0], buffer.size() + 1, &bytesWritten, NULL);

cout << "Message has been encoded." << endl;

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

CloseHandle(hPipeCoutManager);

string decoderPath = "D:\\253505\\5sem\\OSiSP\\LR3\\LR3\\x64\\Debug\\Decoder.exe";

wstring decoderPathModified(decoderPath.begin(), decoderPath.end());

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

if (!CreateProcess(

&decoderPathModified[0], // Путь к исполняемому файлу

NULL, // Командная строка

NULL, // Указатель на процесс

NULL, // Указатель на поток

FALSE, // Наследовать дескрипторы

NULL, // Флаги создания

NULL, // Переменные окружения

NULL, // Рабочий каталог

&si, // Структура информации о запуске

&pi)) // Структура информации о процессе

{

cerr << "CreateProcess failed (" << GetLastError() << ").\n";

return -1;

}

cout << "Waiting for decoder to connect..." << endl;

result = ConnectNamedPipe(hPipeDecoder, NULL);

if (!result) {

cerr << "ConnectNamedPipe failed. Error: " << GetLastError() << endl;

CloseHandle(hPipeDecoder);

return 1;

}

cout << "Decoder connected." << endl;

WriteFile(hPipeDecoder, &buffer[0], buffer.size() + 1, &bytesWritten, NULL);

ReadFile(hPipeDecoder, &buffer[0], buffer.size() + 1, &bytesRead, NULL);

// cout << "Received from decoder: " << buffer << endl;

cout << "Message has been decoded." << endl;

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

hPipeCoutManager = CreateNamedPipeA(

namedPipeCoutManager.c\_str(), // Имя канала

PIPE\_ACCESS\_DUPLEX, // Чтение и запись

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, // Тип сообщения

PIPE\_UNLIMITED\_INSTANCES, // Максимальное количество инстанций

512, // Выходной буфер

512, // Входной буфер

0, // Тайм-аут

NULL // Без защиты

);

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

cerr << "CreateNamedPipe failed. Error: " << GetLastError() << endl;

return 1;

}

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

if (!CreateProcess(

&coutManagerPathModefied[0], // Путь к исполняемому файлу

NULL, // Командная строка

NULL, // Указатель на процесс

NULL, // Указатель на поток

FALSE, // Наследовать дескрипторы

NULL, // Флаги создания

NULL, // Переменные окружения

NULL, // Рабочий каталог

&si, // Структура информации о запуске

&pi)) // Структура информации о процессе

{

cerr << "CreateProcess failed (" << GetLastError() << ").\n";

return -1;

}

cout << "Waiting for cout manager to connect..." << endl;

result = ConnectNamedPipe(hPipeCoutManager, NULL);

if (!result) {

cerr << "ConnectNamedPipe failed. Error: " << GetLastError() << endl;

CloseHandle(hPipeCoutManager);

return 1;

}

cout << "Cout manager connected." << endl;

WriteFile(hPipeCoutManager, &buffer[0], buffer.size() + 1, &bytesWritten, NULL);

cout << "Message has been encoded." << endl;

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

CloseHandle(hPipeCoutManager);

CloseHandle(hPipeDecoder);

CloseHandle(hPipeEncoder);

return 0;

}