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.253505 Снежко М.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 6](#_Toc179742264)

[Список использованных источников 7](#_Toc179742265)

[Приложение А (обязательное) 8](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение методов и инструментов организации конвейерной обработки с применением различных системных объектов и функций для обеспечения передачи данных между процессами и их совместной обработки. В процессе выполнения лабораторной работы также будут рассмотрены типичные проблемы, возникающие при организации взаимодействия между процессами, и подходы к их решению.

В качестве задачи необходимо разработать программу на языке C++, моделирующую конвейерную обработку данных. Конвейер будет включать несколько процессов, каждый из которых будет выполнять определенный этап обработки данных. В выбранной организации цепочки обработки этапы состоят из генерации блоков данных, их сортировки, сборки и вывода результатов.

Для выполнения этой задачи потребуется настроить межпроцессное взаимодействие с использованием именованных и неименованных каналов, очередей сообщений или почтовых ящиков. Каждый процесс должен взаимодействовать с другим, передавая обработанные данные по цепочке.

Лабораторная работа направлена на изучение методов и инструментов организации конвейерной обработки с применением системных объектов и функций для передачи данных между процессами, которые взаимодействуют, передавая обработанные данные по цепочке. В рамках обработки генерируется блок данных, который передается на следующую ступень для сортировки, где данные упорядочиваются и подготавливаются для последующей передачи. Затем происходит сборка, объединяющая отсортированные данные в единый результат, который на финальном этапе выводится, завершая обработку.

Таким образом, лабораторная работа позволит углубить понимание принципов конвейерной обработки и механизмов межпроцессного взаимодействия для построения эффективных цепочек обработки данных.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Объект синхронизации – это объект, дескриптор которого можно указать в одной из функций ожидания для координации выполнения нескольких потоков. Несколько процессов могут иметь дескриптор одного и того же объекта синхронизации, что делает возможной синхронизацию между процессами [1].

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений [2].

Операционная система Windows предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включенные этими механизмами, называются межпроцессными коммуникациями (IPC). Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети.

Как правило, приложения могут использовать IPC, классифицированные как клиенты или серверы. Клиент – это приложение или процесс, который запрашивает службу от другого приложения или процесса. Сервер – это приложение или процесс, который отвечает на запрос клиента. Многие приложения действуют как клиент, так и сервер в зависимости от ситуации. Например, приложение обработки слов может выступать в качестве клиента при запросе сводной таблицы производственных затрат от приложения электронной таблицы, выступающего в качестве сервера. В свою очередь, приложение электронной таблицы может выступать в качестве клиента при запросе последних уровней инвентаризации из приложения автоматического управления инвентаризацией [3].

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Разработанная программа представляет собой моделирование конвейерной обработки данных с использованием нескольких процессов, где каждый процесс отвечает за выполнение определенной функции в цепочке: генерацию данных, сортировку, сборку и вывод. Программа реализована на языке C++ и состоит из нескольких файлов, включая GenerateData.cpp для генерации данных, SortData.cpp для их сортировки, OutputData.cpp для итогового вывода, и основной файл Lab3.cpp, координирующий работу конвейера.

На этапе генерации данных процесс создает блок данных, который передается следующему процессу. Этот этап реализован в файле GenerateData.cpp, который формирует начальные данные и передает их на следующий этап обработки.

Далее происходит сортировка данных, выполняемая процессом, описанным в файле SortData.cpp. Здесь данные упорядочиваются и подготавливаются для последующей передачи в процесс сборки.

Процесс сборки объединяет отсортированные данные в единый результат, а финальный этап вывода выполняется в файле OutputData.cpp, где обработанные данные выводятся на экран или в файл.

Основной файл Lab3.cpp управляет запуском и взаимодействием этих процессов, а также применяет различные методы межпроцессного взаимодействия (IPC), такие как именованные и неименованные каналы, для передачи данных между этапами конвейера. Результат работы программы представлен на рисунке 3.1.

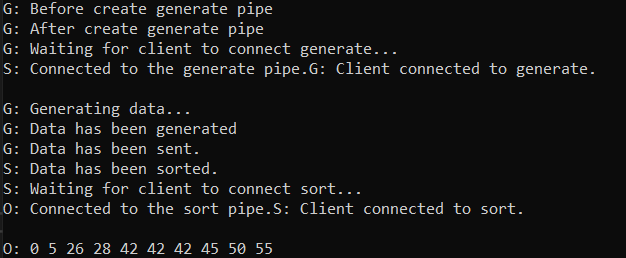


Рисунок 3.1 – Результат работы программы

Так, в результате работы программы достигается конвейерная обработка данных, обеспечивающая поэтапное взаимодействие между процессами в режиме реального времени. Это позволяет передавать и обрабатывать данные последовательно, демонстрируя основные принципы межпроцессного взаимодействия и эффективности распределенной обработки данных.

# ЗАКЛЮЧЕНИЕ

Таким образов, в рамках данной лабораторной работы была создана программа для моделирования конвейерной обработки данных, которая состоит из нескольких взаимосвязанных этапов. Основная цель работы заключалась в исследовании и применении различных подходов, системных объектов и функций, необходимых для эффективной передачи данных между взаимодействующими процессами. Такой метод конвейерной обработки был выбран для изучения преимуществ поэтапного распределения задач, что позволяет разбить обработку различных данных на четко определенные последовательные стадии.

Разработанная программа представляет собой программную модель системы массового обслуживания, где каждый процесс в цепочке выполняет определенный этап обработки и передает результаты следующему этапу. На первом этапе генерируется блок данных, который затем передается процессу сортировки. Сортировка данных, проводимая на втором этапе, позволяет упорядочить полученные значения, подготавливая их для дальнейших манипуляций. На следующем этапе отсортированные данные объединяются, создавая цельный результат, и, наконец, последний этап выводит итоговую информацию.

Для организации передачи данных между процессами и создания цепочки взаимодействий использовались функции и объекты межпроцессного взаимодействия (IPC), такие как именованные каналы WinAPI. Эти каналы обеспечивают надежную передачу информации между процессами и гарантируют, что данные будут переданы в требуемом формате и в нужной последовательности. В частности, функции создания и работы с каналами, такие как CreateNamedPipe, ConnectNamedPipe, ReadFile и WriteFile, позволяют процессам обмениваться информацией в реальном времени и поддерживать синхронизацию выполнения каждого этапа конвейера.

Такой подход моделирования конвейерной обработки, кроме реализации базовой функциональности, позволил детально изучить и понять типичные проблемы, возникающие при организации межпроцессного взаимодействия, включая синхронизацию процессов, обеспечение целостности данных и управление доступом к ресурсам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft "Synchronization objects" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

[2] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.

[3] Microsoft "Взаимодействие процессов" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

int main()

{

// Создаем три процесса для каждого этапа обработки

STARTUPINFO si[3];

PROCESS\_INFORMATION pi[3];

std::wstring commands[3] =

{

L"GenerateData.exe",

L"SortData.exe",

L"OutputData.exe"

};

for (int i = 0; i < 3; ++i) {

ZeroMemory(&si[i], sizeof(si[i]));

si[i].cb = sizeof(si[i]);

ZeroMemory(&pi[i], sizeof(pi[i]));

// Запуск процессов

if (!CreateProcess(NULL, const\_cast<LPWSTR>(commands[i].c\_str()), NULL, NULL, FALSE, 0, NULL, NULL, &si[i], &pi[i]))

{

std::cerr << "CreateProcess failed: " << GetLastError() << " " << i << '\n';

return 1;

}

}

for (int i = 0; i < 3; ++i)

{

CloseHandle(pi[i].hProcess);

CloseHandle(pi[i].hThread);

}

return 0;

}

#include <windows.h>

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#define BUFFER\_SIZE 512

void GenerateData()

{

HANDLE hPipeGenerate;

DWORD dwWritten;

int dataCount = 10;

std::vector<int> data(dataCount);

std::cout << "G: Before create generate pipe" << '\n';

hPipeGenerate = CreateNamedPipe(

TEXT("\\\\.\\pipe\\GenerateDataPipe"),

PIPE\_ACCESS\_DUPLEX,

PIPE\_TYPE\_BYTE | PIPE\_READMODE\_BYTE | PIPE\_WAIT,

1,

BUFFER\_SIZE,

BUFFER\_SIZE,

0,

NULL);

if (hPipeGenerate == INVALID\_HANDLE\_VALUE)

{

std::cerr << "G: CreateNamedPipe generate failed: " << GetLastError() << '\n';

return;

}

std::cout << "G: After create generate pipe" << '\n';

// Ожидание подключения клиента

std::cout << "G: Waiting for client to connect generate..." << '\n';

if (!ConnectNamedPipe(hPipeGenerate, NULL))

{

std::cerr << "G: ConnectNamedPipe generate failed: " << GetLastError() << '\n';

CloseHandle(hPipeGenerate);

return;

}

std::cout << "G: Client connected to generate." << '\n';

// Генерация данных

std::cout << "G: Generating data..." << '\n';

std::srand(static\_cast<unsigned int>(std::time(0)));

for (int& num : data)

{

num = std::rand() % 100;

}

std::cout << "G: Data has been generated" << '\n';

if (!WriteFile(hPipeGenerate, data.data(), data.size() \* sizeof(int), &dwWritten, NULL))

{

std::cerr << "G: WriteFile failed: " << GetLastError() << '\n';

}

else

{

std::cout << "G: Data has been sent." << '\n';

}

CloseHandle(hPipeGenerate);

}

//int main() {

// GenerateData();

// return 0;

//}

#include <windows.h>

#include <iostream>

#include <vector>

void OutputData()

{

HANDLE hPipe;

DWORD dwRead;

std::vector<int> data(10);

// Подключение к каналу

while (true)

{

hPipe = CreateFile(

TEXT("\\\\.\\pipe\\SortDataPipe"),

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

0,

NULL);

if (hPipe == INVALID\_HANDLE\_VALUE)

{

std::cerr << "O: CreateFile failed: " << GetLastError() << '\n';

}

else

{

std::cout << "O: Connected to the sort pipe." << '\n';

break;

}

Sleep(100);

}

// Чтение данных

if (!ReadFile(hPipe, data.data(), data.size() \* sizeof(int), &dwRead, NULL))

{

std::cerr << "O: ReadFile sort failed: " << GetLastError() << '\n';

CloseHandle(hPipe);

return;

}

// Вывод данных

std::cout << "O: ";

for (int num : data)

{

std::cout << num << " ";

}

std::cout << '\n';

CloseHandle(hPipe);

}

//int main() {

// OutputData();

// return 0;

//}