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»
БГУИР 6-05-0612-02 23

Выполнил студент группы 353503
СЕБЕЛЕВ Дмитрий Юрьевич

_____
(дата, подпись студента)

Проверил ассистент каф. информатики
ГРИЦЕНКО Никита Юрьевич

_____
(дата, подпись преподавателя)

Минск 2025

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Целью работы является моделирование и анализ трехступенчатого конвейера обработки данных, состоящего из независимых процессов: «Генератор», «Обработчик» и «Визуализатор». Исследование сфокусировано на технической реализации межпроцессного взаимодействия с использованием именованных каналов (Named Pipes) для передачи структурированных данных между этапами конвейера. В рамках работы производится оценка временных задержек, возникающих на каждом этапе обработки, и анализ общей производительности системы.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

2.1 Инициализация системных ресурсов и координация процессов

Этот модуль отвечает за настройку среды выполнения и управление жизненным циклом конвейера обработки данных. Инициализация начинается с установки кодовой страницы UTF-8 функцией SetConsoleOutputCP для корректного отображения кириллицы и инициализации генератора случайных чисел с помощью srand для последующего создания тестовых данных. Программа анализирует аргументы командной строки для определения своей роли в конвейере: «generator», «processor» или «visualizer». Если аргументы отсутствуют, программа переходит в режим координатора, который последовательно запускает три дочерних процесса с помощью функции CreateProcessA. Каждому процессу передается соответствующий аргумент для определения его роли, а флаг CREATE_NEW_CONSOLE обеспечивает запуск каждого этапа конвейера в отдельном консольном окне. Координатор ожидает завершения всех трех дочерних процессов, используя WaitForMultipleObjects, после чего корректно закрывает все дескрипторы процессов и потоков функцией CloseHandle, обеспечивая правильное управление системными ресурсами. Программный код представлен в приложении А [1].

2.2 Процесс-генератор

Этот модуль представляет собой первый этап конвейера, отвечающий за создание и передачу исходных данных. Процесс-генератор создает именованный канал «\.pipePipelineStage1» с помощью функции CreateNamedPipeA с параметром PIPE_ACCESS_OUTBOUND для однонаправленной передачи данных. Канал настраивается для работы в режиме сообщений (PIPE_TYPE_MESSAGE), что гарантирует атомарность операций чтения и записи для целых блоков данных. После создания канала процесс переходит в режим ожидания подключения следующего этапа конвейера с помощью функции ConnectNamedPipe. После успешного подключения генератор в цикле создает пять структурированных блоков данных DataBlock, каждый из которых инициализируется уникальным идентификатором, массивом из десяти случайных чисел и временной меткой, полученной через GetTickCount. Сформированные блоки данных последовательно записываются в канал функцией WriteFile, после чего процесс-генератор корректно закрывает дескриптор канала и завершает свою работу [2].

2.3 Процесс-обработчик

Этот модуль реализует второй, промежуточный этап конвейера, который выполняет основную обработку данных. Процесс-обработчик сначала подключается к входному каналу «\.pipePipelineStage1»как клиент с помощью

функции CreateFileA с правом доступа GENERIC_READ. Затем он создает собственный выходной именованный канал «\.pipePipelineStage2» функцией CreateNamedPipeA и ожидает подключения процесса-визуализатора. После установления всех соединений обработчик входит в основной цикл, где последовательно считывает блоки данных из входного канала функцией ReadFile. Для каждого полученного блока выполняется операция сортировки внутреннего массива данных по возрастанию с использованием std::sort. Статус блока обновляется на «sorted», после чего обработанный блок данных записывается в выходной канал функцией WriteFile для передачи на следующий этап конвейера. Цикл продолжается до тех пор, пока во входном канале не закончатся данные, после чего процесс-обработчик закрывает дескрипторы обоих каналов и завершает свою работу [3].

2.4 Процесс-визуализатор и анализ результатов

Этот модуль является завершающим этапом конвейера, ответственным за получение и отображение конечных результатов. Процесс-визуализатор подключается к каналу «\.pipePipelineStage2» функцией CreateFileA с правом доступа GENERIC_READ и ожидает поступления обработанных данных. В цикле он считывает отсортированные блоки данных функцией ReadFile. Для каждого полученного блока вычисляется общее время обработки как разница между текущим временем (GetTickCount) и временной меткой, установленной генератором. Результаты выводятся в консоль в виде форматированной таблицы, содержащей идентификатор блока, его статус, отсортированные данные и вычисленное время обработки в миллисекундах. После получения и отображения всех блоков данных программа выводит итоговую статистику, включая общее количество обработанных блоков и среднее время прохождения одного блока через весь конвейер. Завершение работы модуля включает корректное закрытие дескриптора канала с помощью CloseHandle [4].

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Пример выполнения задания

На рисунке 3.1 представлен результат работы процесса-генератора.

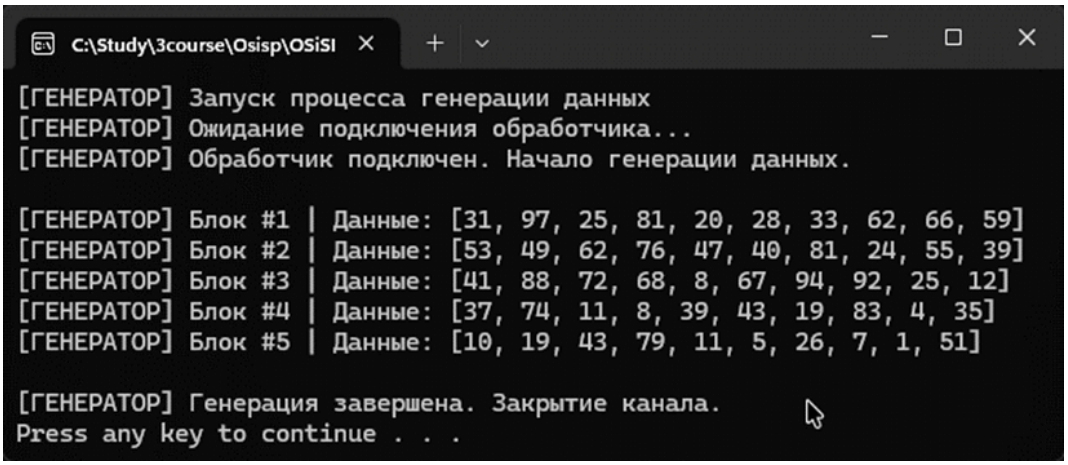

Рисунок 3.1 – Вывод процесса-генератора

Результат работы процесса-обработчика представлен на рисунке 3.2.

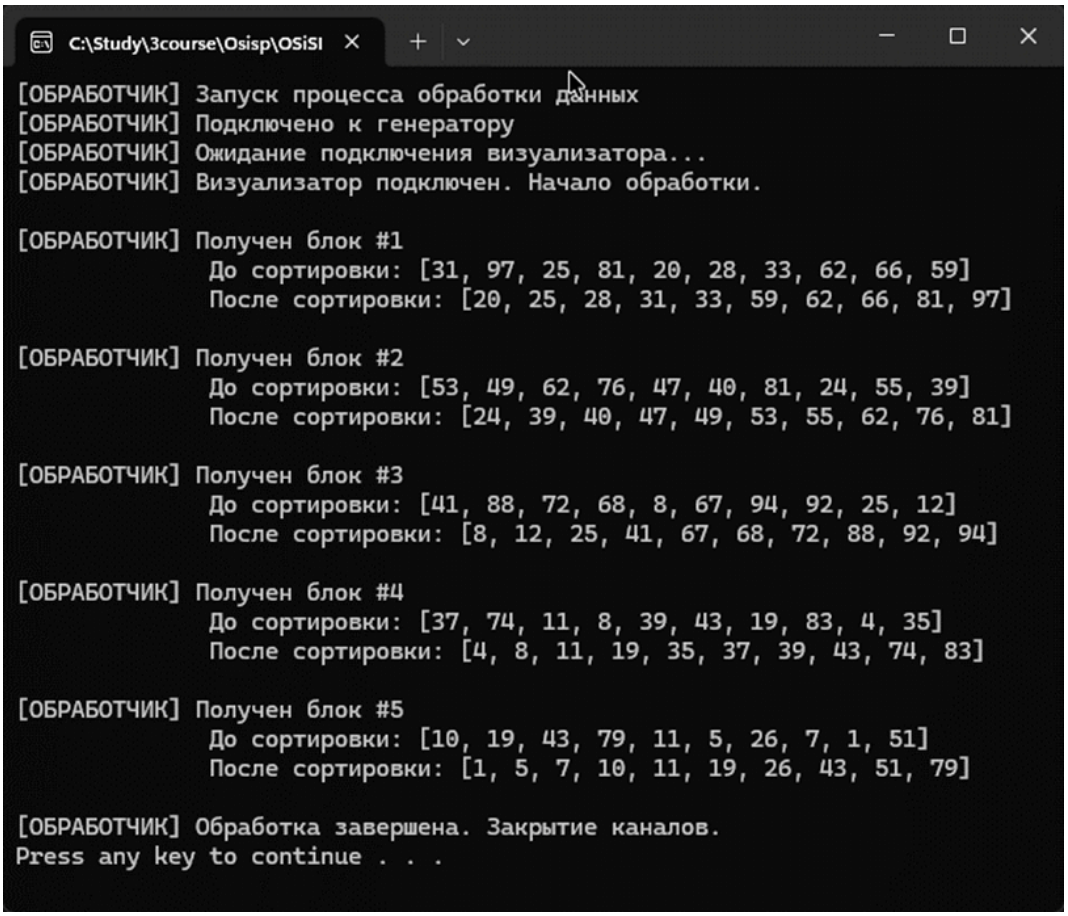

Рисунок 3.2 – Результат работы процесса-обработчика

Результат работы процесса-визуализатора представлен на рисунке 3.3.

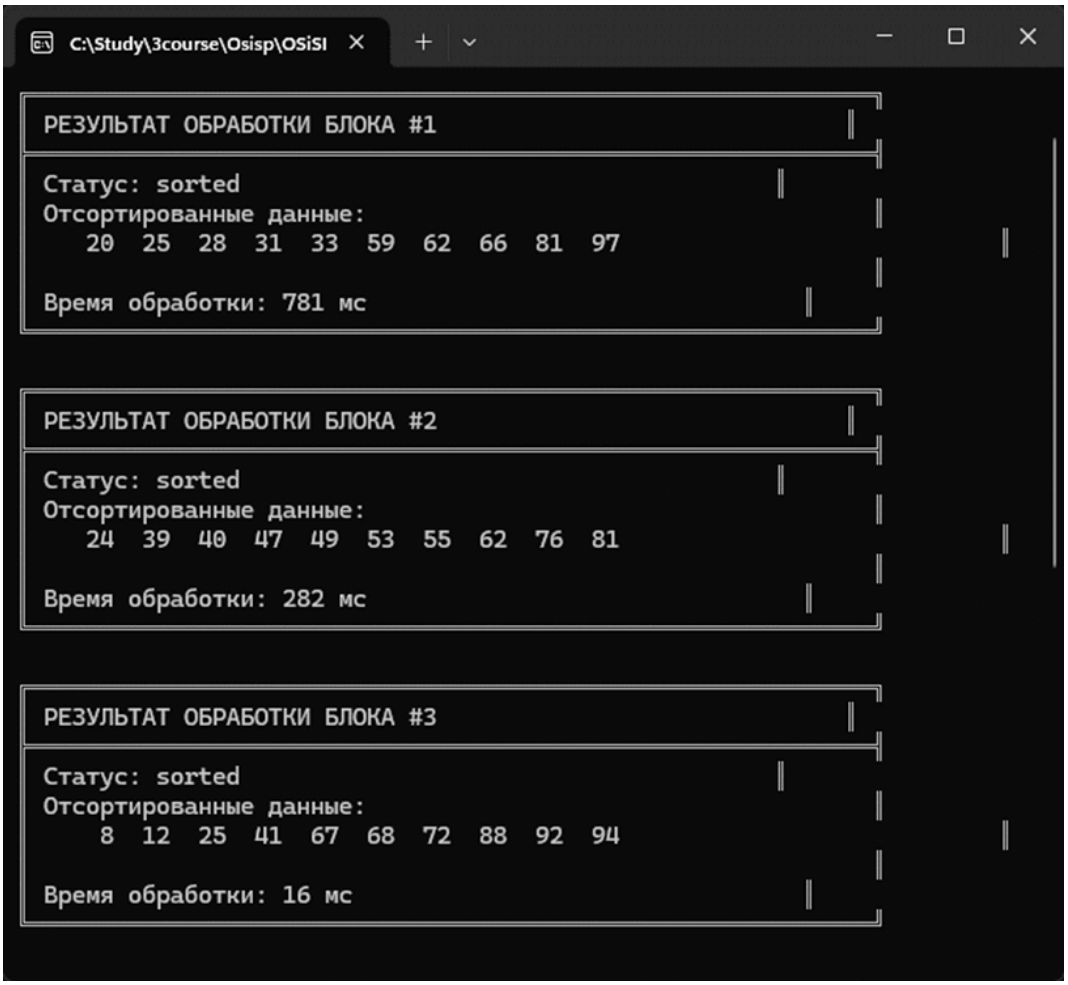

Рисунок 3.3 – Результат работы процесса-визуализатора

ВЫВОД

В ходе выполнения лабораторной работы были освоены практические навыки реализации и анализа систем межпроцессного взаимодействия (IPC) в операционной системе Windows с использованием WinAPI. Была разработана программная модель, имитирующая трехступенчатый конвейер обработки данных, состоящий из трех независимых, но взаимодействующих друг с другом процессов: «Генератор», «Обработчик» и «Визуализатор».

Программа демонстрирует организацию последовательной передачи структурированных данных между процессами с использованием именованных каналов (Named Pipes). Процесс-генератор отвечает за создание исходных блоков данных, каждый из которых содержит уникальный идентификатор, массив случайных чисел и временную метку создания, полученную с помощью функции GetTickCount. Эти блоки передаются по первому именованному каналу на следующий этап.

Процесс-обработчик реализует промежуточный этап конвейера, считывая данные из входного канала, выполняя над ними операцию сортировки внутреннего массива и передавая результат в выходной канал. Этот этап демонстрирует модель «фильтра», который получает данные, модифицирует их и отправляет дальше по цепочке, что является классическим сценарием применения конвейерной обработки.

Процесс-визуализатор выступает в роли конечного потребителя данных, считывая обработанные блоки и производя анализ производительности системы. Ключевой метрикой является вычисление общего времени прохождения каждого блока данных через конвейер, что позволяет оценить задержки, вносимые операциями передачи по каналам и обработкой на каждом этапе.

Для организации взаимодействия использовались функции WinAPI для работы с именованными каналами: CreateNamedPipeA для создания серверной части канала, ConnectNamedPipe для ожидания подключения клиента и CreateFileA для подключения к существующему каналу. Передача данных осуществлялась функциями WriteFile и ReadFile, работающими в режиме сообщений (PIPE_TYPE_MESSAGE), что гарантирует атомарность передачи целых структур DataBlock.

Управление жизненным циклом конвейера реализовано в координирующем процессе, который использует CreateProcessA для запуска каждого из трех дочерних процессов в отдельных консольных окнах. Синхронизация завершения работы всего конвейера обеспечивается функцией WaitForMultipleObjects, которая ожидает завершения всех дочерних процессов перед завершением основной программы.

Полученные навыки работы с механизмами межпроцессного взаимодействия и организации многопроцессных систем являются фундаментальными для разработки сложных распределенных приложений. Реализованная модель конвейерной обработки наглядно демонстрирует принципы разделения задач, асинхронной передачи данных и анализа

производительности, которые широко применяются в серверных приложениях, системах пакетной обработки и модульных программных комплексах. Результаты работы позволяют количественно оценить накладные расходы на сериализацию и передачу данных между процессами, что критически важно при проектировании высокопроизводительных систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] API Win32 documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/. – Дата доступа: 04.11.2025.

[2] WinAPI: CreateProcessA [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa. – Дата доступа: 04.11.2025.

[3] WinAPI: ConnectNamedPipe [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-connectnamedpipe. – Дата доступа: 04.11.2025.

[4] WinAPI: CreateNamedPipeA [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winbase/nf-winbase-createnamedpipea. – Дата доступа: 04.11.2025.

ПРИЛОЖЕНИЕ А
(справочное)
Исходный код программы

```
#include <windows.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <ctime>
#include <string>
#include <sstream>

struct DataBlock {
    int id;
    int data[10];
    DWORD timestamp;
    char status[32];
};

// ПРОЦЕСС 1: ГЕНЕРАТОР ДАННЫХ
void GeneratorProcess() {
    std::cout << "[ГЕНЕРАТОР] Запуск процесса генерации данных\n";

    HANDLE hPipe = CreateNamedPipeA(
        "\\\\.\\pipe\\PipelineStage1",
        PIPE_ACCESS_OUTBOUND,
        PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT,
        1,
        sizeof(DataBlock) * 10,
        sizeof(DataBlock) * 10,
        0,
        NULL
    );

    if (hPipe == INVALID_HANDLE_VALUE) {
        std::cerr << "[ГЕНЕРАТОР] Ошибка создания канала: " <<
GetLastError() << "\n";
        return;
    }

    std::cout << "[ГЕНЕРАТОР] Ожидание подключения обработчика...\n";

    if (!ConnectNamedPipe(hPipe, NULL)) {
        std::cerr << "[ГЕНЕРАТОР] Ошибка подключения: " << GetLastError()
<< "\n";
        CloseHandle(hPipe);
        return;
    }

    std::cout << "[ГЕНЕРАТОР] Обработчик подключен. Начало генерации
данных.\n\n";

    int totalBlocks = 5;
    for (int blockId = 1; blockId <= totalBlocks; blockId++) {
        DataBlock block;
        block.id = blockId;
        block.timestamp = GetTickCount();
        strcpy_s(block.status, "generated");

        std::cout << "[ГЕНЕРАТОР] Блок #" << blockId << " | Данные: [";
        for (int i = 0; i < 10; i++) {
            block.data[i] = rand() % 100;
            std::cout << block.data[i];
            if (i < 9) std::cout << ", ";
        }
        std::cout << "]\n";

        DWORD written;
```

```
            if (!WriteFile(hPipe, &block, sizeof(DataBlock), &written, NULL))
{
                std::cerr << "[ГЕНЕРАТОР] Ошибка записи в канал\n";
                break;
            }

            Sleep(1000);
        }

        std::cout << "\n[ГЕНЕРАТОР] Генерация завершена. Закрытие канала.\n";
        CloseHandle(hPipe);
}

// ПРОЦЕСС 2: СОРТИРОВКА
void ProcessorProcess() {
        std::cout << "[ОБРАБОТЧИК] Запуск процесса обработки данных\n";

        Sleep(500);

        HANDLE hPipeIn = CreateFileA(
            "\\\\.\\pipe\\PipelineStage1",
            GENERIC_READ,
            0,
            NULL,
            OPEN_EXISTING,
            0,
            NULL
        );

        if (hPipeIn == INVALID_HANDLE_VALUE) {
            std::cerr << "[ОБРАБОТЧИК] Ошибка подключения к входному каналу:
" << GetLastError() << "\n";
            return;
        }

        std::cout << "[ОБРАБОТЧИК] Подключено к генератору\n";

        HANDLE hPipeOut = CreateNamedPipeA(
            "\\\\.\\pipe\\PipelineStage2",
            PIPE_ACCESS_OUTBOUND,
            PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT,
            1,
            sizeof(DataBlock) * 10,
            sizeof(DataBlock) * 10,
            0,
            NULL
        );

        if (hPipeOut == INVALID_HANDLE_VALUE) {
            std::cerr << "[ОБРАБОТЧИК] Ошибка создания выходного канала: " <<
GetLastError() << "\n";
            CloseHandle(hPipeIn);
            return;
        }

        std::cout << "[ОБРАБОТЧИК] Ожидание подключения визуализатора...\n";

        if (!ConnectNamedPipe(hPipeOut, NULL)) {
            std::cerr << "[ОБРАБОТЧИК] Ошибка подключения визуализатора: " <<
GetLastError() << "\n";
            CloseHandle(hPipeIn);
            CloseHandle(hPipeOut);
            return;
        }

        std::cout << "[ОБРАБОТЧИК] Визуализатор подключен. Начало обработки.
\n\n";

        while (true) {
            DataBlock block;
            DWORD read;
```

```
        if (!ReadFile(hPipeIn, &block, sizeof(DataBlock), &read, NULL) ||
read == 0) {
            break;
        }

        std::cout << "[ОБРАБОТЧИК] Получен блок #" << block.id << "\n";
        std::cout << "                До сортировки: [";
        for (int i = 0; i < 10; i++) {
            std::cout << block.data[i];
            if (i < 9) std::cout << ", ";
        }
        std::cout << "]\n";

        std::sort(block.data, block.data + 10);
        strcpy_s(block.status, "sorted");

        std::cout << "                После сортировки: [";
        for (int i = 0; i < 10; i++) {
            std::cout << block.data[i];
            if (i < 9) std::cout << ", ";
        }
        std::cout << "]\n\n";

        DWORD written;
        if (!WriteFile(hPipeOut, &block, sizeof(DataBlock), &written,
NULL)) {
            std::cerr << "[ОБРАБОТЧИК] Ошибка записи в выходной
канал\n";
            break;
        }

        Sleep(500);
    }

    std::cout << "[ОБРАБОТЧИК] Обработка завершена. Закрытие каналов.\n";
    CloseHandle(hPipeIn);
    CloseHandle(hPipeOut);
}

// ПРОЦЕСС 3: ВИЗУАЛИЗАТОР
void VisualizerProcess() {
    std::cout << "[ВИЗУАЛИЗАТОР] Запуск процесса визуализации\n";

    Sleep(1000);

    HANDLE hPipe = CreateFileA(
        "\\\\.\\pipe\\PipelineStage2",
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );

    if (hPipe == INVALID_HANDLE_VALUE) {
        std::cerr << "[ВИЗУАЛИЗАТОР] Ошибка подключения к каналу: " <<
GetLastError() << "\n";
        return;
    }

    std::cout << "[ВИЗУАЛИЗАТОР] Подключено к обработчику. Ожидание
данных.\n\n";

    int totalBlocks = 0;
    DWORD totalTime = 0;

    while (true) {
        DataBlock block;
        DWORD read;
```

```
        if (!ReadFile(hPipe, &block, sizeof(DataBlock), &read, NULL) ||
read == 0) {
            break;
        }

        totalBlocks++;
        DWORD processingTime = GetTickCount() - block.timestamp;
        totalTime += processingTime;

        std::cout <<
"╔═══════════════════════════════════════════════════════════════╗\n";
        std::cout << "║ РЕЗУЛЬТАТ ОБРАБОТКИ БЛОКА #" << block.id << "
║\n";
        std::cout <<
"╠═══════════════════════════════════════════════════════════════╣\n";
        std::cout << "║ Статус: " << block.status << "
║\n";
        std::cout << "║ Отсортированные данные:
║\n";
        std::cout << "║   ";

        for (int i = 0; i < 10; i++) {
            printf("%3d ", block.data[i]);
        }
        std::cout << "                              ║\n";

        std::cout << "║
║\n";
        std::cout << "║ Время обработки: " << processingTime << " мс";
        for (int i = 0; i < 34 -
std::to_string(processingTime).length(); i++) std::cout << " ";
        std::cout << "║\n";
        std::cout <<
"╚═══════════════════════════════════════════════════════════════╝\n\n";

        Sleep(300);
    }

    std::cout << "\n[ВИЗУАЛИЗАТОР] Обработано блоков: " << totalBlocks <<
"\n";
    std::cout << "[ВИЗУАЛИЗАТОР] Среднее время обработки: " <<
(totalBlocks > 0 ? totalTime / totalBlocks : 0) << " мс\n";

    CloseHandle(hPipe);
}


int main(int argc, char* argv[]) {
    SetConsoleOutputCP(CP_UTF8);
    srand(static_cast<unsigned int>(time(NULL)));

    // Определение режима работы по аргументам командной строки
    if (argc > 1) {
        std::string mode = argv[1];

        if (mode == "generator") {
            GeneratorProcess();
        }
        else if (mode == "processor") {
            ProcessorProcess();
        }
        else if (mode == "visualizer") {
            VisualizerProcess();
        }

        system("pause");
        return 0;
    }

    std::cout <<
```

```
"═══════════════════════════════════════════════════════════════\n";
    std::cout << "  СИСТЕМА КОНВЕЙЕРНОЙ ОБРАБОТКИ ДАННЫХ (WinAPI)\n";
    std::cout <<
"═══════════════════════════════════════════════════════════════\n\n";
    std::cout << "Архитектура конвейера:\n";
    std::cout << "  ГЕНЕРАТОР → [канал 1] → ОБРАБОТЧИК → [канал 2] →
ВИЗУАЛИЗАТОР\n\n";
    std::cout << "Запуск процессов конвейера...\n\n";

    char exePath[MAX_PATH];
    GetModuleFileNameA(NULL, exePath, MAX_PATH);

    STARTUPINFOA si1 = { sizeof(si1) }, si2 = { sizeof(si2) }, si3 =
{ sizeof(si3) };
    PROCESS_INFORMATION pi1, pi2, pi3;

    // Запуск процесса генератора
    std::string cmdGenerator = std::string(exePath) + " generator";
    if (!CreateProcessA(NULL, (LPSTR)cmdGenerator.c_str(), NULL, NULL,
FALSE,
                        CREATE_NEW_CONSOLE, NULL, NULL, &si1, &pi1)) {
        std::cerr << "Ошибка запуска генератора: " << GetLastError() <<
"\n";
        return 1;
    }
    std::cout << "[КООРДИНАТОР] Генератор запущен (PID: " <<
pi1.dwProcessId << ")\n";

    Sleep(300);

    std::string cmdProcessor = std::string(exePath) + " processor";
    if (!CreateProcessA(NULL, (LPSTR)cmdProcessor.c_str(), NULL, NULL,
FALSE,
                        CREATE_NEW_CONSOLE, NULL, NULL, &si2, &pi2)) {
        std::cerr << "Ошибка запуска обработчика: " << GetLastError() <<
"\n";
        return 1;
    }
    std::cout << "[КООРДИНАТОР] Обработчик запущен (PID: " <<
pi2.dwProcessId << ")\n";

    Sleep(300);

    std::string cmdVisualizer = std::string(exePath) + " visualizer";
    if (!CreateProcessA(NULL, (LPSTR)cmdVisualizer.c_str(), NULL, NULL,
FALSE,
                        CREATE_NEW_CONSOLE, NULL, NULL, &si3, &pi3)) {
        std::cerr << "Ошибка запуска визуализатора: " << GetLastError()
<< "\n";
        return 1;
    }
    std::cout << "[КООРДИНАТОР] Визуализатор запущен (PID: " <<
pi3.dwProcessId << ")\n\n";

    std::cout << "Все процессы запущены. Ожидание завершения...\n\n";

    HANDLE processes[] = { pi1.hProcess, pi2.hProcess, pi3.hProcess };
    WaitForMultipleObjects(3, processes, TRUE, INFINITE);

    std::cout << "\n[КООРДИНАТОР] Все процессы конвейера завершены.\n";
    std::cout <<
"═══════════════════════════════════════════════════════════════\n";

    CloseHandle(pi1.hProcess);
    CloseHandle(pi1.hThread);
    CloseHandle(pi2.hProcess);
    CloseHandle(pi2.hThread);
    CloseHandle(pi3.hProcess);
    CloseHandle(pi3.hThread);

    system("pause");
```

```
    return 0;
}
```