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 № 4

на тему «Управление процессами и потоками (Windows). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности.»

Выполнил:

студент гр. 153504

Михалевич M.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цели работы 3](#_Toc147863804)

[2 Kраткие теоретические сведения 4](#_Toc147863805)

[3 Результаты выполнения лабораторной работы 5](#_Toc147863806)

[Вывод 6](#_Toc147863807)

[Список использованных источников 7](#_Toc147863808)

[Приложение А 8](#_Toc147863809)

## 1 ЦЕЛИ РАБОТЫ

Целью данной лабораторной работы является разработка многозадачного приложения, способного эффективно обрабатывать большие объемы данных с использованием многопоточности. В процессе выполнения задачи, необходимо также реализовать асинхронные механизмы обработки данных, что позволит увеличить производительность и отзывчивость приложения. Дополнительно, целью является настройка приоритетов потоков.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Управление процессами и потоками в операционной системе Windows - это важный аспект обеспечения эффективности и многозадачности в работе с приложениями. Процесс представляет собой изолированное пространство выполнения, которое включает в себя один или несколько потоков. Потоки, в свою очередь, представляют собой независимые выполнительные единицы внутри процесса, которые могут выполняться параллельно.

Порождение и завершение процессов и потоков - это ключевые операции в управлении ими. Для создания нового процесса или потока в Windows используются функции, такие как CreateProcess и CreateThread. После завершения выполнения задачи процесс или поток могут быть завершены с помощью функций ExitProcess и ExitThread, соответственно. Эффективное управление порождением и завершением процессов и потоков позволяет оптимизировать использование системных ресурсов и уменьшить нагрузку на центральный процессор.[1]

Изменение приоритетов процессов и потоков также играет важную роль в управлении ресурсами. В Windows существует несколько уровней приоритетов, от самого низкого до самого высокого. Эти приоритеты влияют на то, как ОС распределяет процессорное время между процессами и потоками. Высокий приоритет позволяет приложению получить больше процессорного времени и, таким образом, повысить отзывчивость системы. Однако неправильное управление приоритетами может привести к деградации производительности системы.

В процессе выполнения данной лабораторной работы, были применены следующие теоретические сведения:

1. Порождение процессов и потоков: В коде использованы функции CreateThread для создания трех потоков, которые выполняют фиктивную обработку данных. Эта операция аналогична созданию потоков в рамках процесса.

2. Завершение процессов и потоков: В коде используется функция ExitThread для завершения выполнения каждого потока. Это позволяет корректно завершить выполнение каждого потока после выполнения задачи.

3. Изменение приоритетов процессов и потоков: В коде присутствуют выпадающие списки, которые позволяют пользователю выбрать приоритет для каждого потока. Используя функцию SetThreadPriority, приоритет каждого потока устанавливается в соответствии с выбранным пользователем значением (низкий, обычный, высокий).

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В рамках проведения лабораторной работы было создано приложение, способное эффективно обрабатывать большие объемы данных с использованием многопоточности. Пользователю предоставлена возможность выбора приоритета для каждого потока. После нажатия кнопки "Старт", пользователь видит результат выполнения каждого потока. Подробности работы программы представлены на рисунке 3.1.

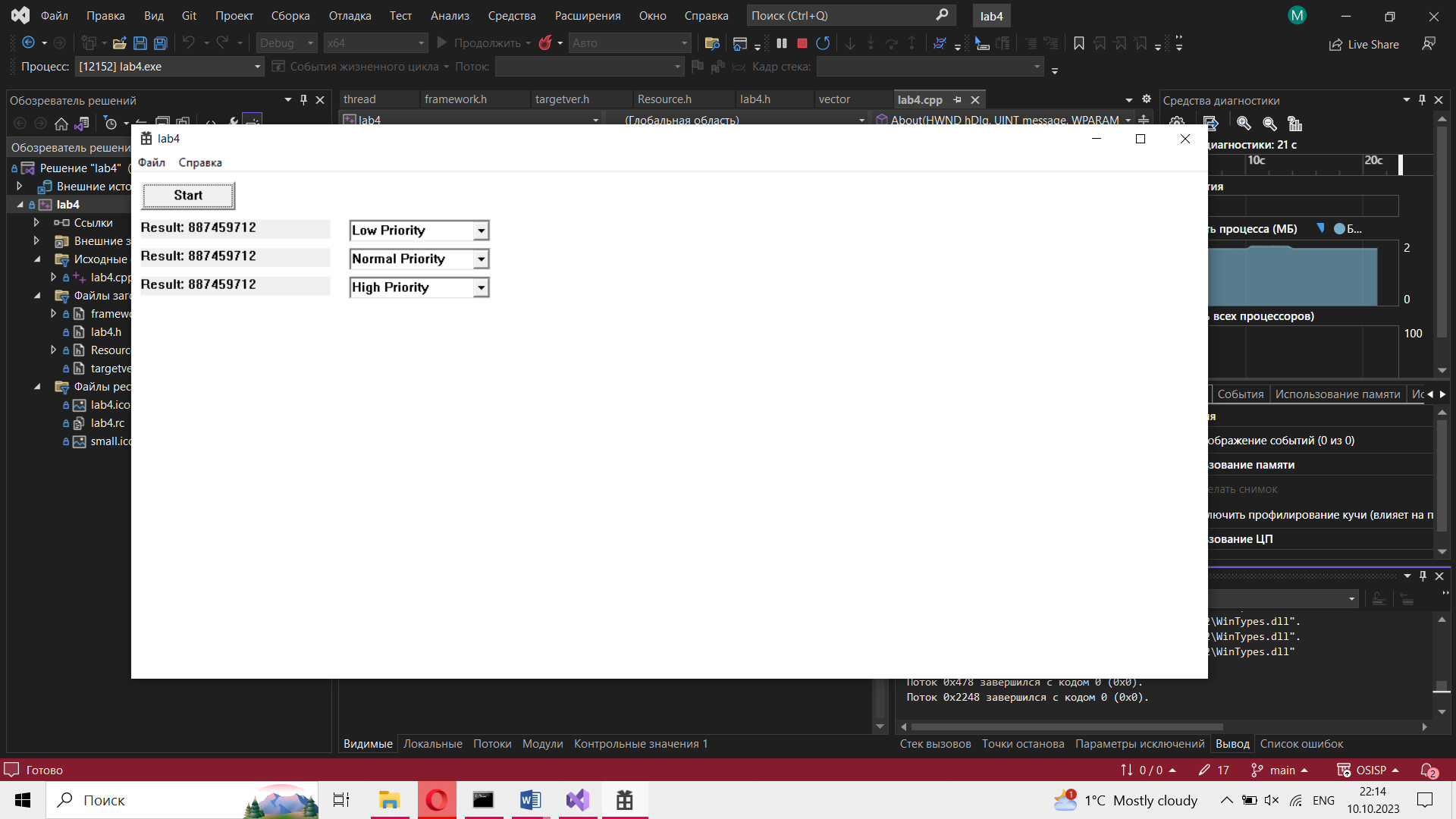


Рисунок 3.1 – Результат работы программы

## ВЫВОД

В ходе выполнения данной лабораторной работы было разработано приложение, предназначенное для эффективной обработки объемных данных, используя принципы многопоточности. Пользователь получает возможность настраивать приоритеты для каждого создаваемого потока, что способствует более оптимальному выполнению задачи в различных сценариях. После запуска процесса обработки данных, пользователь может получить информацию о результате выполнения каждого потока. На основе этой информации можно оценить эффективность и состояние каждого потока в процессе работы программы.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/api – Дата доступа 09.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[3] [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170 – Дата доступа 09.10.2023](https://stackoverflow.com/questions/17187265/how-to-group-radio-box-buttons-using-win32-api%20–%20Дата%20доступа%2023.09.2023)

## ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода**

// lab4.cpp : Определяет точку входа для приложения.

//

https://learn.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170

#include "framework.h"

#include "lab4.h"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

HWND hStatusLabel = nullptr;

HWND hCombo1 = nullptr;

HWND hCombo2 = nullptr;

HWND hCombo3 = nullptr;

HWND hResultLabel1 = nullptr;

HWND hResultLabel2 = nullptr;

HWND hResultLabel3 = nullptr;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

#include <vector>

#include <thread>

std::vector<std::thread> threads;

int result1 = 0;

int result2 = 0;

int result3 = 0;

#include <cmath> // для std::sin

void processData(int threadPriority, int\* result)

{

SetThreadPriority(GetCurrentThread(), threadPriority);

// Фиктивная обработка данных

for (int i = 0; i < 100000000; ++i) {

\*result += i;

volatile double tmp = std::sin(i);

//Sleep(10);

}

if (result == &result1)

PostMessage(GetParent(hResultLabel1), WM\_USER + 1, (WPARAM)&result1, (LPARAM)hResultLabel1);

else if (result == &result2)

PostMessage(GetParent(hResultLabel2), WM\_USER + 1, (WPARAM)&result2, (LPARAM)hResultLabel2);

else

PostMessage(GetParent(hResultLabel3), WM\_USER + 1, (WPARAM)&result3, (LPARAM)hResultLabel3);

}

void SetResultLabel(HWND label, int value)

{

wchar\_t resultText[100];

swprintf\_s(resultText, L"Result: %d", value);

SetWindowText(label, resultText);

}

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_LAB4, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB4));

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LAB4));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB4);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

// Создать кнопки "Start" и "Stop"

CreateWindow(L"BUTTON", L"Start", WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD,

10, 10, 100, 30, hWnd, (HMENU)IDC\_START\_BUTTON, hInst, NULL);

//Создать выпадающий список для установки приоритета потока

hCombo1 = CreateWindow(L"COMBOBOX", L"", CBS\_DROPDOWN | WS\_CHILD | WS\_VISIBLE,

230, 50, 150, 100, hWnd, (HMENU)IDC\_PRIORITY\_COMBO1, hInst, NULL);

hCombo2 = CreateWindow(L"COMBOBOX", L"", CBS\_DROPDOWN | WS\_CHILD | WS\_VISIBLE,

230, 80, 150, 100, hWnd, (HMENU)IDC\_PRIORITY\_COMBO2, hInst, NULL);

hCombo3 = CreateWindow(L"COMBOBOX", L"", CBS\_DROPDOWN | WS\_CHILD | WS\_VISIBLE,

230, 110, 150, 100, hWnd, (HMENU)IDC\_PRIORITY\_COMBO3, hInst, NULL);

SendMessage(hCombo1, CB\_ADDSTRING, 0, (LPARAM)L"Low Priority");

SendMessage(hCombo1, CB\_ADDSTRING, 0, (LPARAM)L"Normal Priority");

SendMessage(hCombo1, CB\_ADDSTRING, 0, (LPARAM)L"High Priority");

SendMessage(hCombo1, CB\_SETCURSEL, 1, 0);

SendMessage(hCombo2, CB\_ADDSTRING, 0, (LPARAM)L"Low Priority");

SendMessage(hCombo2, CB\_ADDSTRING, 0, (LPARAM)L"Normal Priority");

SendMessage(hCombo2, CB\_ADDSTRING, 0, (LPARAM)L"High Priority");

SendMessage(hCombo2, CB\_SETCURSEL, 1, 0);

SendMessage(hCombo3, CB\_ADDSTRING, 0, (LPARAM)L"Low Priority");

SendMessage(hCombo3, CB\_ADDSTRING, 0, (LPARAM)L"Normal Priority");

SendMessage(hCombo3, CB\_ADDSTRING, 0, (LPARAM)L"High Priority");

SendMessage(hCombo3, CB\_SETCURSEL, 1, 0);

hResultLabel1 = CreateWindow(L"STATIC", L"Result 1: 0", WS\_VISIBLE | WS\_CHILD,

10, 50, 200, 20, hWnd, (HMENU)IDC\_RESULT\_LABEL1, hInst, NULL);

hResultLabel2 = CreateWindow(L"STATIC", L"Result 2: 0", WS\_VISIBLE | WS\_CHILD,

10, 80, 200, 20, hWnd, (HMENU)IDC\_RESULT\_LABEL2, hInst, NULL);

hResultLabel3 = CreateWindow(L"STATIC", L"Result 3: 0", WS\_VISIBLE | WS\_CHILD,

10, 110, 200, 20, hWnd, (HMENU)IDC\_RESULT\_LABEL3, hInst, NULL);

}

break;

case WM\_USER + 1:

{

int\* pResult = (int\*)wParam;

HWND hResultLabel = (HWND)lParam;

if (pResult && hResultLabel)

{

wchar\_t resultText[100];

swprintf\_s(resultText, L"Result: %d", \*pResult);

SetWindowText(hResultLabel, resultText);

}

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDC\_START\_BUTTON:

{

int selectedPriority1 = SendMessage(hCombo1, CB\_GETCURSEL, 0, 0);

int selectedPriority2 = SendMessage(hCombo2, CB\_GETCURSEL, 0, 0);

int selectedPriority3 = SendMessage(hCombo3, CB\_GETCURSEL, 0, 0);

int threadPriority1 = THREAD\_PRIORITY\_NORMAL;

int threadPriority2 = THREAD\_PRIORITY\_NORMAL;

int threadPriority3 = THREAD\_PRIORITY\_NORMAL;

if (selectedPriority1 == 0) {

threadPriority1 = THREAD\_PRIORITY\_LOWEST;

}

else if (selectedPriority1 == 2) {

threadPriority1 = THREAD\_PRIORITY\_HIGHEST;

}

if (selectedPriority2 == 0) {

threadPriority2 = THREAD\_PRIORITY\_LOWEST;

}

else if (selectedPriority2 == 2) {

threadPriority2 = THREAD\_PRIORITY\_HIGHEST;

}

if (selectedPriority3 == 0) {

threadPriority3 = THREAD\_PRIORITY\_LOWEST;

}

else if (selectedPriority3 == 2) {

threadPriority3 = THREAD\_PRIORITY\_HIGHEST;

}

// Запустить потоки для обработки данных с выбранными приоритетами

std::thread thread1(processData, threadPriority1, &result1);

std::thread thread2(processData, threadPriority2, &result2);

std::thread thread3(processData, threadPriority3, &result3);

thread1.detach();

thread2.detach();

thread3.detach();

}

break;

case IDM\_EXIT:

{

DestroyWindow(hWnd);

break;

}

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}