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**

Студент Е. А. Сироткин

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной лабораторной работы является изучение и практическое применение методов управления процессами и потоками в операционной системе *Windows*, включая их порождение, завершение и изменение приоритетов. Основные задачи лабораторной работы включают в себя овладение навыками управления многозадачностью и эффективным распределением ресурсов в операционной системе.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе *Windows* является фундаментальной частью обеспечения многозадачности и эффективной работы приложений. Процесс представляет собой изолированную программу с собственным адресным пространством и ресурсами, а потоки - это единицы выполнения внутри процесса. Создание и управление процессами и потоками включает в себя функции создания новых процессов и потоков, изменения их приоритетов, а также синхронизацию выполнения приложений. *Windows* *API* предоставляет набор функций для эффективного контроля над выполнением процессов и потоков, что позволяет разработчикам создавать отзывчивые и эффективные многозадачные приложения.

Потоки представляют собой легковесные подпроцессы внутри основного процесса, которые могут выполняться параллельно и многозадачно. Они используются для выполнения асинхронных задач, таких как обработка данных, выполнение вычислений, ввод-вывод и другие операции, которые могут выполняться параллельно.

Порождение процессов и потоков в операционной системе *Windows* —это процесс создания новых выполнимых сущностей для выполнения задач. Создание новых потоков, с другой стороны, выполняется с использованием функции *CreateThread*, что позволяет приложению разделять задачи и выполнять их параллельно в рамках одного процесса. Порождение процессов и потоков дает приложениям возможность многозадачности и эффективного использования ресурсов.

Для ожидания завершения выполнения другого потока используется *WaitForSingleObject* - это функция, позволяющая потоку ожидать завершения выполнения объекта синхронизации, такого как процесс, поток, событие или мьютекс. Поток, вызвавший *WaitForSingleObject*, блокируется до завершения объекта синхронизации. Это удобно для синхронизации и координации действий между потоками и процессами в многозадачных приложениях.

Использование нескольких потоков в многопоточных приложениях может оказать значительное влияние на производительность. Многопоточные приложения способны эффективно распределять вычислительную нагрузку между несколькими потоками, что позволяет одновременно выполнять несколько задач. Это может ускорить выполнение задач, особенно тех, которые легко разделяются на независимые подзадачи. Однако использование многопоточности также может вызвать сложности, связанные с синхронизацией доступа к общим ресурсам и предотвращением гонок за данные.

Таким образом, эффективное использование многопоточности требует внимательного планирования и управления потоками, но при правильной реализации оно может значительно улучшить производительность и отзывчивость приложений.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Было создано приложение, вычисляющее значения трудоемких математических вычислений, а также показывающее скорость их выполнения. Сравнение происходит в однопоточном и многопоточном режимах. При нажатии на какую-либо из кнопок происходит инициализации потоков и дальнейшая обработка вычислений. (рисунок 1)*.*

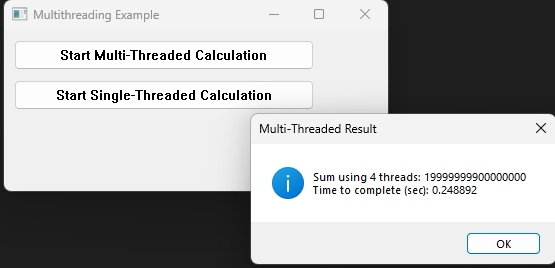


Рисунок 1 – Окно приложения

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были освоены фундаментальные аспекты управления процессами и потоками в операционной системе *Windows*. Это включало в себя навыки порождения и завершения процессов и потоков. В рамках работы над приложением была добавлена функциональность для эффективного управления выполнением процессов и потоков, что позволило оптимально использовать ресурсы системы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Безруков В. Win32 API. Программирование /учебное пособие. ─ СПб: СПбГУ ИТМО, 2009. ─ 90 с.
2. C++ потоки в Win32 [Электронный ресурс]. ─ Режим доступа: http://www.interface.ru/home.asp?artId=29606.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Lab4.cpp*

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#include <windows.h>

#include <vector>

#include <thread>

#include <iostream>

#include <string>

using namespace std;

unsigned long long results[4] = { 0,0,0,0 };

unsigned long long a, b, c, d;

unsigned long long result1 = 0, result2 = 0, result3 = 0, result4 = 0, result = 0;

const unsigned long long ARRAY\_SIZE = 200000000;

HANDLE writeCompleteEvent;

vector<unsigned long long> data1(ARRAY\_SIZE);

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

void CreateThreads();

void CalculateSingleThread();

double GetTimeElapsed(LARGE\_INTEGER startTime, LARGE\_INTEGER endTime);

double PCFreq;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

for (unsigned long long i = 0; i < ARRAY\_SIZE; ++i) {

data1[i] = i;

}

a = 0;

b = ARRAY\_SIZE / 4;

c = ARRAY\_SIZE / 2;

d = 3 \* (ARRAY\_SIZE / 4);

LARGE\_INTEGER frequency;

QueryPerformanceFrequency(&frequency);

PCFreq = double(frequency.QuadPart) / 1000.0;

WNDCLASS wc = { 0 };

wc.hbrBackground = (HBRUSH)COLOR\_WINDOW;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hInstance = hInst;

wc.lpszClassName = L"MultithreadingApp";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc)) {

return -1;

}

CreateWindowW(L"MultithreadingApp", L"Multithreading Example", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 400, 200, NULL, NULL, NULL, NULL);

MSG msg = { 0 };

while (GetMessage(&msg, NULL, NULL, NULL)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

if (wp == 1) {

CreateThreads();

}

else if (wp == 2) {

CalculateSingleThread();

}

break;

case WM\_CREATE:

CreateWindow(L"button", L"Start Multi-Threaded Calculation", WS\_VISIBLE | WS\_CHILD, 10, 10, 300, 30, hWnd, (HMENU)1, NULL, NULL);

CreateWindow(L"button", L"Start Single-Threaded Calculation", WS\_VISIBLE | WS\_CHILD, 10, 50, 300, 30, hWnd, (HMENU)2, NULL, NULL);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

return 0;

}

DWORD WINAPI ThreadFunction1(LPVOID aboba) {

unsigned long long result = 0;

for (unsigned long long i = a; i < b; ++i) {

result += data1[i];

}

results[0] = result;

return result;

}

DWORD WINAPI ThreadFunction2(LPVOID aboba) {

unsigned long long result = 0;

for (unsigned long long i = b; i < c; ++i) {

result += data1[i];

}

results[1] = result;

return result;

}

DWORD WINAPI ThreadFunction3(LPVOID aboba) {

unsigned long long result = 0;

for (unsigned long long i = c; i < d; ++i) {

result += data1[i];

}

results[2] = result;

return result;

}

DWORD WINAPI ThreadFunction4(LPVOID aboba) {

unsigned long long result = 0;

for (unsigned long long i = d; i < ARRAY\_SIZE; ++i) {

result += data1[i];

}

results[3] = result;

return result;

}

void CreateThreads() {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

HANDLE threads[4];

threads[0] = CreateThread(NULL, 0, ThreadFunction1, NULL, 0, NULL);

threads[1] = CreateThread(NULL, 0, ThreadFunction2, NULL, 0, NULL);

threads[2] = CreateThread(NULL, 0, ThreadFunction3, NULL, 0, NULL);

threads[3] = CreateThread(NULL, 0, ThreadFunction4, NULL, 0, NULL);

for (int i = 0; i < 4; i++) {

WaitForSingleObject(threads[i], INFINITE);

GetExitCodeThread(threads[i], (LPDWORD)(&results[i]));

}

unsigned long long totalSum = 0;

for (int i = 0; i < 4; i++) {

totalSum += results[i];

}

QueryPerformanceCounter(&endTime);

double timeElapsed = GetTimeElapsed(startTime, endTime);

wstring resultText = L"Sum using 4 threads: " + to\_wstring(totalSum) + L"\nTime to complete (sec): " + to\_wstring(timeElapsed);

MessageBoxW(NULL, resultText.c\_str(), L"Multi-Threaded Result", MB\_OK | MB\_ICONINFORMATION);

}

void CalculateSingleThread() {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

unsigned long long result = 0;

for (unsigned long long i = 0; i < ARRAY\_SIZE; ++i) {

result += data1[i];

}

QueryPerformanceCounter(&endTime);

double timeElapsed = GetTimeElapsed(startTime, endTime);

wstring resultText = L"Sum using singular thread: " + to\_wstring(result) + L"\nTime to complete (sec): " + to\_wstring(timeElapsed);

MessageBoxW(NULL, resultText.c\_str(), L"Single-Threaded Result", MB\_OK | MB\_ICONINFORMATION);

}

double GetTimeElapsed(LARGE\_INTEGER startTime, LARGE\_INTEGER endTime) {

return (double(endTime.QuadPart - startTime.QuadPart) / PCFreq) / 1000.0;

}