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 №3

на тему

Управление потоками и нитями

Студент П. С. Павлюткин

Преподаватель С. И. Сиротко

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**

Цель работы изучить и применить на практике знания об управлении процессами и потоками на *Windows*, научиться создавать, завершать, изменять приоритеты процессов и потоков, а также изучить организацию и контроль исследование эффективности.Для достижения цели будет создано приложение осуществляющее запуск нескольких процессов и демонстрирующий приоритеты потоков с оценкой их производительности.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе *Windows* является важной частью организации работы приложений и ресурсов компьютера.

Вычислительные потоки. Вычислительный поток как системный объект, объект *Thread* в *Windows*. Атрибуты потока, в т.ч. приоритеты. Состояния потока. Разделяемые адресное пространство и пространство дескрипторов. Создание (порождение) потоков, главный поток. Диспетчирование потоков. Приостановка и возобновление потоков. Завершение потоков. Код завершения. Нити (*Fiber*), особенности их выполнения. Взаимосвязь нитей и потоков. Создание, завершение, переключение нитей. Программный интерфейс (*API*) для базового управления потоками и нитями.

Процесс в операционной системе *Windows* представляет собой независимую программу, выполняющуюся в собственном адресном пространстве памяти. Процесс обладает собственными системными ресурсами, такими как память и дескрипторы файлов. Поток легковесный исполняемый агент внутри процесса. Потоки внутри одного процесса могут совместно использовать ресурсы этого процесса, включая память и файлы. Один процесс может содержать несколько потоков, которые выполняются параллельно.

Для порождения новых процессов в *Windows* можно использовать функции, такие как *CreateProcess*. Это позволяет запускать новые исполняемые файлы в отдельных процессах. Для порождения новых потоков внутри процесса можно использовать функции, такие как *CreateThread*. Это позволяет создавать параллельные задачи внутри процесса. Завершение процессов и потоков может осуществляться с помощью функций *ExitProcess* и *ExitThread*. Эти функции завершают выполнение соответствующих процессов и потоков.

В *Windows* можно изменять приоритеты процессов и потоков для управления их доступом к ресурсам *CPU*. Высший приоритет обеспечивает больше времени *CPU* для выполнения задачи. Приоритеты процессов можно устанавливать с помощью функции *SetPriorityClass*, а приоритеты потоков - с помощью функции *SetThreadPriority*.

Для оценки эффективности работы процессов и потоков в *Windows* можно использовать различные инструменты мониторинга и профилирования, такие как *Windows* *Performance* *Monitor* и *Visual* *Studio* *Profiler*. Эти инструменты позволяют анализировать использование *CPU*, памяти и других ресурсов, а также выявлять узкие места в процессах и потоках для оптимизации производительности.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение осуществляющее запуск нескольких процессов и демонстрирующий приоритеты потоков с оценкой их производительности включая старт процесса, окончание процесса, и число отведенных ресурсов для каждого из них (Рисунок 1).

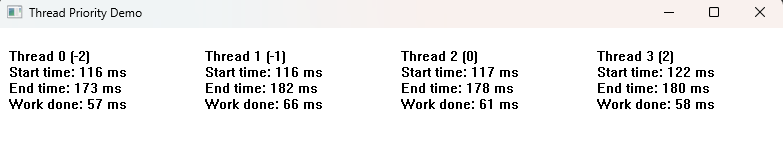


Рисунок 1 – Окно приложения с отображением производительности потоков

В нашей программе вычисляется произведение матриц с фиксированным вводом потоков, в нашем случае 4 потока и матрица размером 200 для большей нагруженности нашей программы. Сейчас чтобы продемонстрировать работу программы изменим размер матрицы с двухсот до ста и увидим разницу выполнения (Рисунок 2).

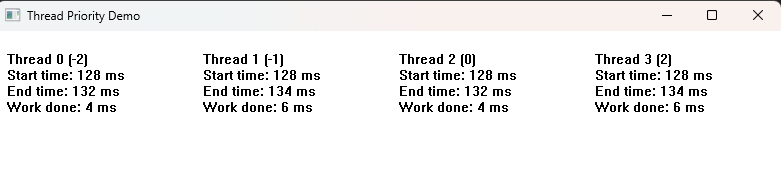


Рисунок 2– Выход за игровое поле

В квадратных скобках мы показываем какой поток приоритетнее: 0 максимальный приоритет, -2 соответственно низший. Также мы отображаем время начала и конец выполнения, а также количество проделанной работы каждым процессом по отдельности.

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены и применены важные аспекты управления процессами и потоками в операционной системе *Windows* с использованием *Win32* *API*

Данное приложение представляет собой проект, который наглядно демонстрирует применение указанных аспектов программирования в контексте разработки многозадачных и многопоточных приложений под операционной системой *Windows*. Это позволяет эффективно управлять процессами и потоками, обеспечивая отзывчивость и производительность приложения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. *Win32 API*. Разработка приложений для *Windows*. ─ СПБ: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для *Windows* с использованием *API Win32* [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab3.cpp

#include <windows.h>

#include <tchar.h>

#include <iostream>

#include <vector>

#include <thread>

#include <ctime>

#include <iomanip>

#include <sstream>

const int numThreads = 4;

const int matrixSize = 200;

struct ThreadInfo {

std::thread thread;

int priority;

clock\_t start\_time;

clock\_t end\_time;

long long work\_done;

HWND hwnd;

};

std::vector<ThreadInfo> threads;

RECT columns[numThreads];

int columnWidth;

void MultiplyMatrices(int threadIndex) {

int matrixA[matrixSize][matrixSize];

int matrixB[matrixSize][matrixSize];

int resultMatrix[matrixSize][matrixSize];

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

for (int j = 0; j < matrixSize; ++j) {

matrixA[i][j] = rand() % 100;

matrixB[i][j] = rand() % 100;

}

}

clock\_t start\_time = clock();

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

for (int j = 0; j < matrixSize; ++j) {

resultMatrix[i][j] = 0;

for (int k = 0; k < matrixSize; ++k) {

resultMatrix[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

clock\_t end\_time = clock();

long long work\_done = (end\_time - start\_time);

threads[threadIndex].start\_time = start\_time;

threads[threadIndex].end\_time = end\_time;

threads[threadIndex].work\_done = work\_done;

PostMessage(threads[threadIndex].hwnd, WM\_USER, 0, threadIndex);

}

void DisplayInfo(HWND hwnd, int threadIndex) {

std::wstringstream wss;

wss << L"Thread " << threadIndex << L" (" << threads[threadIndex].priority << L")\n";

wss << L"Start time: " << threads[threadIndex].start\_time << L" ms\n";

wss << L"End time: " << threads[threadIndex].end\_time << L" ms\n";

wss << L"Work done: " << threads[threadIndex].work\_done << L" ms";

RECT columnRect = columns[threadIndex];

columnRect.top += 20;

columnRect.left += 10;

columnRect.right -= 10;

HDC hdc = GetDC(hwnd);

SetBkMode(hdc, TRANSPARENT);

DrawText(hdc, wss.str().c\_str(), -1, &columnRect, DT\_LEFT | DT\_TOP);

ReleaseDC(hwnd, hdc);

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

if (msg == WM\_CREATE) {

srand(static\_cast<unsigned>(time(nullptr)));

RECT clientRect;

GetClientRect(hwnd, &clientRect);

columnWidth = (clientRect.right - clientRect.left) / numThreads;

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

columns[i] = { i \* columnWidth, 0, (i + 1) \* columnWidth, clientRect.bottom };

threads.push\_back({ std::thread(MultiplyMatrices, i), THREAD\_PRIORITY\_NORMAL, 0, 0, 0, hwnd });

}

threads[0].priority = THREAD\_PRIORITY\_LOWEST;

threads[1].priority = THREAD\_PRIORITY\_BELOW\_NORMAL;

threads[2].priority = THREAD\_PRIORITY\_NORMAL;

threads[3].priority = THREAD\_PRIORITY\_HIGHEST;

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

SetThreadPriority(threads[i].thread.native\_handle(), threads[i].priority);

}

return 0;

}

if (msg == WM\_CLOSE) {

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

threads[i].thread.join();

}

PostQuitMessage(0);

return 0;

}

if (msg == WM\_USER) {

int threadIndex = lParam;

DisplayInfo(hwnd, threadIndex);

return 0;

}

return DefWindowProc(hwnd, msg, wParam, lParam);

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASSEX wc = { 0 };

wc.cbSize = sizeof(WNDCLASSEX);

wc.style = 0;

wc.lpfnWndProc = WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hInstance;

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

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

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

wc.lpszMenuName = NULL;

wc.lpszClassName = \_T("ThreadPriorityDemo");

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

if (!RegisterClassEx(&wc)) {

MessageBox(NULL, \_T("Failed to register the window class"), \_T("Error"), MB\_ICONERROR);

return -1;

}

HWND hwnd = CreateWindow(\_T("ThreadPriorityDemo"), \_T("Thread Priority Demo"), WS\_OVERLAPPEDWINDOW, 0, 0, 800, 600, NULL, NULL, hInstance, NULL);

if (!hwnd) {

MessageBox(NULL, \_T("Failed to create the window"), \_T("Error"), MB\_ICONERROR);

return -1;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}