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 №1

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ, ПОТОКАМИ, НИТЯМИ**.

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 ФОРМУЛИРОВКА ЗАДАЧИ 3](#_Toc177991217)

[2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc177991218)

[3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ 5](#_Toc177991219)

[3.1 Функция ThreadFunction 5](#_Toc177991220)

[3.2 Функция MultiplyMatrices 5](#_Toc177991221)

[3.3 Функция UpdateProgressBar 5](#_Toc177991222)

[3.4 Функция main 5](#_Toc177991223)

[4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ 6](#_Toc177991224)

[4.1 Запуск программы и процесс выполнения 6](#_Toc177991225)

[4.2 Описание работы и результатов 6](#_Toc177991226)

[ВЫВОД 7](#_Toc177991227)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc177991228)

[ПРИЛОЖЕНИЕ А (обязательное) 9](#_Toc177991229)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является закрепление и развитие навыков программирования приложений для операционной системы Windows с использованием многопоточности. В ходе выполнения работы необходимо изучить основные этапы жизненного цикла потоков: их порождение, завершение, получение и изменение состояния. Также важно понять, как приоритеты потоков влияют на их выполнение и производительность системы.

Задача лабораторной работы заключается в разработке многопоточного приложения, которое демонстрирует основные принципы управления потоками, а также оценку их производительности в зависимости от установленных приоритетов. Программа должна выполнять следующие функции:

1 Создание потоков с разными приоритетами: запуск нескольких потоков с заранее заданными приоритетами (CRITICAL, HIGHEST, NORMAL, BELOW\_NORMAL, LOWEST, IDLE).

2 Выполнение вычислительной задачи: каждый поток должен выполнять вычислительную сложную задачу (умножение матриц), что обеспечит достаточную нагрузку для демонстрации работы потоков.

3 Отображение процесса выполнения: в консоли приложения необходимо в реальном времени отображать прогресс выполнения каждого потока с помощью прогресс-бара.

4 Измерение времени выполнения: для каждого потока необходимо измерять время выполнения задачи и выводить его после завершения потока, чтобы оценить влияние приоритетов на производительность.

5 Закрытие потоков и завершение программы: приложение должно завершать все потоки, корректно закрывая их дескрипторы, и выводить итоговое сообщение о завершении работы всех потоков.

Приложение должно демонстрировать влияние приоритетов на производительность потоков, наглядно показывая, как операционная система распределяет ресурсы между потоками с различными приоритетами.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Приложение может состоять из одного или нескольких процессов. Процесс — это выполнение программы, включающее в себя системные ресурсы и выделенную область памяти. Процессы изолированы друг от друга, что предотвращает прямое взаимодействие и возможные ошибки. В рамках одного процесса могут функционировать несколько потоков. Поток является основной единицей исполнения кода в процессе, которой операционная система выделяет процессорное время. Он может исполнять любой участок кода процесса, в том числе те, которые уже используются другими потоками.

Используя объекты заданий, можно управлять группами процессов как единым целым. Эти объекты — именуемые, защищаемые и могут делиться между процессами, управляя атрибутами связанных с ними процессов. Взаимодействие с объектом задания влияет на все процессы, ассоциированные с этим объектом.

Пул потоков представляет собой коллекцию рабочих потоков, которые осуществляют асинхронные вызовы от имени приложения. Это средство позволяет сократить количество создаваемых потоков и облегчить управление ими, повышая тем самым эффективность приложения. Пул потоков используется в основном для уменьшения нагрузки на процессор и улучшения общей производительности.

Нить — это единица выполнения, которую приложение должно запланировать вручную. Нити выполняются в контексте потоков, которые планируют их, и обычно используются для задач, требующих быстрого отклика системы.

Критические секции — это механизм синхронизации, который обеспечивает эксклюзивный доступ к коду или ресурсам, предотвращая одновременное исполнение участка кода несколькими потоками. Они важны для предотвращения условий гонки и обеспечения целостности данных в многопоточных приложениях. Критическая секция гарантирует, что только один поток может выполнять защищённый участок кода в любой момент времени, что предотвращает возникновение ошибок и несогласованности данных, которые могут произойти при одновременном доступе к общим данным.[1]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– ThreadFunction;

– MultiplyMatrices;

– UpdateProgressBar;

– main.

3.1 Функция ThreadFunction

Данная функция принимает указатель на индекс потока. Функция совершает следующие действия:

1 Устанавливает приоритет потока;

2 Выполняет инициализацию и запускает матричное умножение;

3 Измеряет время выполнения операции;

4 Выводит в консоль результаты выполнения, включая время и приоритет потока.

3.2 Функция MultiplyMatrices

Функция принимает матрицы для умножения и индекс потока. Функция совершает следующие действия:

1 Производит матричное умножение;

2 Регулярно обновляет прогресс выполнения с помощью функции UpdateProgressBar.

3.3 Функция UpdateProgressBar

Данная функция принимает индекс потока и текущий прогресс выполнения. Функция совершает следующие действия:

1 Блокирует вывод в консоль для синхронизации;

2 Обновляет строку прогресса в консоли;

3 Снимает блокировку вывода после обновления.

3.4 Функция main

Главная функция программы. Функция совершает следующие действия:

1 Инициализирует необходимые механизмы синхронизации;

2 Создает потоки для выполнения задачи умножения матриц;

3 Ожидает завершение всех потоков;

4 Закрывает все потоки и выводит сообщение о завершении работы.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

На рисунке 4.1 показана консоль после запуска программы и завершения всех потоков. Каждый поток отображает свой прогресс выполнения и окончательные результаты, включая время выполнения и приоритет.

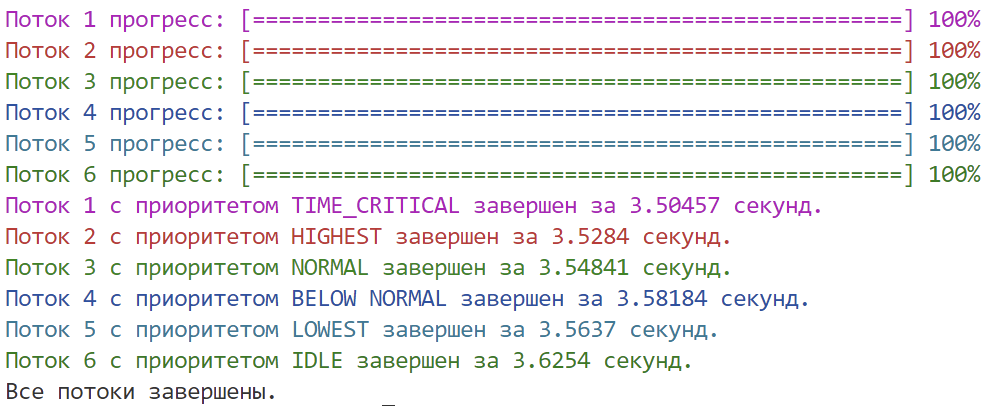
****

Рисунок 4.1 – Консоль с результатами выполнения потоков

4.2 Описание работы и результатов

В процессе выполнения, программа инициализировала шесть потоков, каждому из которых был задан различный уровень приоритета. Эти потоки выполняли матричные вычисления, результаты которых отображены на скриншоте. Как видно из данных, время выполнения потоков различается в зависимости от их приоритета, что демонстрирует влияние приоритета на производительность обработки данных. Приоритет TIME\_CRITICAL позволил завершить работу быстрее других, в то время как IDLE — медленнее всех.

# ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана многопоточная консольная программа на языке C++, предназначенная для демонстрации управления потоками, нитями и их приоритетами в среде Windows. Программа эффективно реализует процесс умножения матриц с использованием нескольких потоков, каждый из которых имеет различные уровни приоритета. Это позволило оценить влияние приоритетов потоков на производительность вычислений.

Результаты экспериментов показали, что потоки с более высоким приоритетом завершают задачи быстрее, чем потоки с более низким приоритетом, подтверждая теоретические сведения о механизме планирования и управления потоками в операционных системах. Реализация прогресс-бара для каждого потока в консоли также обеспечила наглядное отображение процесса выполнения задач, что добавило практическую значимость и образовательную ценность к работе.

Таким образом, лабораторная работа не только способствовала закреплению теоретических знаний по управлению процессами и потоками в операционной системе Windows, но и позволила получить практический опыт разработки многопоточных приложений, что является важным навыком в области системного программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/. – Дата доступа: 23.09.2024.

[2] Разработка приложений с помощью WinAPI [Электронный ресурс]. – Режим доступа: https://shorturl.at/BDJW8. – Дата доступа: 23.09.2024.

[3] Multithreading Tutorial - CodeProject [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.codeproject.com/Articles/14746/Multithreading-Tutorial. – Дата доступа: 23.09.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

#include <bits/stdc++.h>

#include <windows.h>

#define ll long long

#define ld long double

#define vi vector<ll>

#define vd vector<ld>

#define vvi vector<vector<ll>>

#define vvd vector<vector<ld>>

using namespace std;

using namespace chrono;

CRITICAL\_SECTION coutCS;

HANDLE hConsole;

const ll numThreads = 6;

ll threadPriorities[numThreads] = {

    THREAD\_PRIORITY\_TIME\_CRITICAL,

    THREAD\_PRIORITY\_HIGHEST,

    THREAD\_PRIORITY\_NORMAL,

    THREAD\_PRIORITY\_BELOW\_NORMAL,

    THREAD\_PRIORITY\_LOWEST,

    THREAD\_PRIORITY\_IDLE};

string priorityNames[numThreads] = {

    "TIME\_CRITICAL",

    "HIGHEST",

    "NORMAL",

    "BELOW NORMAL",

    "LOWEST",

    "IDLE"};

WORD threadColors[numThreads] = {

    FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY,

    FOREGROUND\_RED | FOREGROUND\_INTENSITY,

    FOREGROUND\_GREEN | FOREGROUND\_INTENSITY,

    FOREGROUND\_BLUE | FOREGROUND\_INTENSITY,

    FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY,

    FOREGROUND\_GREEN};

void UpdateProgressBar(ll index, ll progress)

{

    EnterCriticalSection(&coutCS);

    SetConsoleTextAttribute(hConsole, threadColors[index]);

    COORD cursorPosition;

    cursorPosition.X = 0;

    cursorPosition.Y = (SHORT)(index);

    SetConsoleCursorPosition(hConsole, cursorPosition);

    string progressBar = "Поток " + to\_string(index + 1) + " прогресс: [";

    ll barWidth = 50;

    ll pos = barWidth \* progress / 100;

    for (ll i = 0; i < barWidth; ++i)

    {

        if (i < pos)

            progressBar += "=";

        else

            progressBar += " ";

    }

    progressBar += "] " + to\_string(progress) + "%";

    cout << progressBar;

    SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

    LeaveCriticalSection(&coutCS);

}

void MultiplyMatrices(const vvd &A, const vvd &B, vvd &C, ll threadIndex)

{

    ll N = A.size();

    for (ll i = 0; i < N; ++i)

    {

        for (ll j = 0; j < N; ++j)

        {

            C[i][j] = 0;

            for (ll k = 0; k < N; ++k)

            {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

        ll progress = (100 \* (i + 1)) / N;

        if (progress > 100)

            progress = 100;

        UpdateProgressBar(threadIndex, progress);

    }

}

DWORD WINAPI ThreadFunction(LPVOID lpParam)

{

    ll threadIndex = \*(ll \*)lpParam;

    SetThreadPriority(GetCurrentThread(), threadPriorities[threadIndex]);

    auto startTime = high\_resolution\_clock::now();

    vvd A(500, vector<ld>(500, 1.0));

    vvd B(500, vector<ld>(500, 1.0));

    vvd C(500, vector<ld>(500, 0.0));

    MultiplyMatrices(A, B, C, threadIndex);

    auto endTime = high\_resolution\_clock::now();

    duration<ld> elapsed = endTime - startTime;

    EnterCriticalSection(&coutCS);

    SetConsoleTextAttribute(hConsole, threadColors[threadIndex]);

    COORD cursorPosition;

    cursorPosition.X = 0;

    cursorPosition.Y = (SHORT)(numThreads + threadIndex);

    SetConsoleCursorPosition(hConsole, cursorPosition);

    cout << "Поток " << threadIndex + 1 << " с приоритетом " << priorityNames[threadIndex]

         << " завершен за " << elapsed.count() << " секунд." << endl;

    SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

    LeaveCriticalSection(&coutCS);

    return 0;

}

int main()

{

    InitializeCriticalSection(&coutCS);

    hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    system("cls");

    HANDLE hThreads[numThreads];

    DWORD threadIDs[numThreads];

    ll threadIndices[numThreads];

    for (ll i = 0; i < numThreads; ++i)

    {

        threadIndices[i] = i;

        hThreads[i] = CreateThread(NULL, 0, ThreadFunction, &threadIndices[i], 0, &threadIDs[i]);

    }

    WaitForMultipleObjects(numThreads, hThreads, TRUE, INFINITE);

    for (ll i = 0; i < numThreads; ++i)

    {

        CloseHandle(hThreads[i]);

    }

    DeleteCriticalSection(&coutCS);

    COORD cursorPosition;

    cursorPosition.X = 0;

    cursorPosition.Y = (SHORT)(12);

    SetConsoleCursorPosition(hConsole, cursorPosition);

    cout << "Все потоки завершены." << endl;

    return 0;

}