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 Севрюков С.И.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178690658)

[2 Краткие теоритические сведения 4](#_Toc178690659)

[3 Описание функций программы 5](#_Toc178690660)

[3.1 Ввод количества потоков для выполнения программы 5](#_Toc178690661)

[3.2 Отображение статусов потоков 6](#_Toc178690662)

[Заключение 8](#_Toc178690663)

[Список использованных источников 9](#_Toc178690664)

[Приложение А (обязательное) Исходный код программы 10](#_Toc178690665)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является возобновление, закрепление и развитие навыков программирования приложений Windows. В ходе выполнения работы необходимо изучить концепции вычислительных процессов, потоков, нитей и их реализацию в Windows; основные этапы жизненного цикла процессов (потоков) и элементарное управление ими: порождение, завершение, получение и изменение состояния; типичное (простое) использование многозадачности и многопоточности.

В качестве задачи необходимо запустить несколько (можно фиксированное количество) потоков с разными приоритетами (можно задать заранее) и оценить их производительность. Желательно отображение времени работы потока – в течение его выполнения и итоговое после завершения, и объема выполненной потоком работы – в простейшем случае счетчик итераций. «Содержимое» потока – произвольная задача с достаточной вычислительной сложностью (сортировка, умножение матриц и т.п.). Дополнение: анализ поведения при наличии высокоприоритетных потоков.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Вычислительный поток (программный поток), Thread – объект, связанный с выполнением программы процессором. Поток выполняется параллельно с другими потоками всех процессов и делит адресное пространство с другими потоками своего процесса. Потоки более экономичны по сравнению с процессами и имеют возможность более эффективного и менее затратного взаимодействия друг с другом (в рамках одного процесса). Благодаря этому концепция потоков поддерживается всеми основными современными «универсальными» ОС. Однако реализация может идти различными путями: MS Windows (Win 32 и Win 64)[1] – изначально предусмотренный объект ядра; Unix и Unix-подобные системы – как правило, «облегченный» процесс.

В многопоточной системе планировщик управляет именно потоками, поэтому представление о состояниях переносится с процессов на потоки.

Пул потоков — это коллекция рабочих потоков, которые эффективно выполняют асинхронные обратные вызовы от имени приложения. Пул потоков в основном используется для уменьшения количества потоков приложения и обеспечения управления рабочими потоками.

Нить — это единица выполнения, которую приложение должно запланировать вручную. Нити выполняются в контексте потоков, которые планируют их.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– ввод количества потоков для запуска математических вычислений;

– случайное распределение приоритетов для потоков;

– вывод каждой итерации потоков с общим временем выполнения;

– вывод конечного времени выполнения потока и его приоритета с вычисленной производительностью и запись полученных данных в файл.

## **3.1 Ввод количества потоков для выполнения программы**

Для ограничения ввода количества потоков выводится текстовое сообщение, предупреждающее о том, что граничное количество потоков, при котором программа еще выполняется, составляет 64, а при 65 программа уже не запустится. Сообщение представлено на рисунке 3.1.

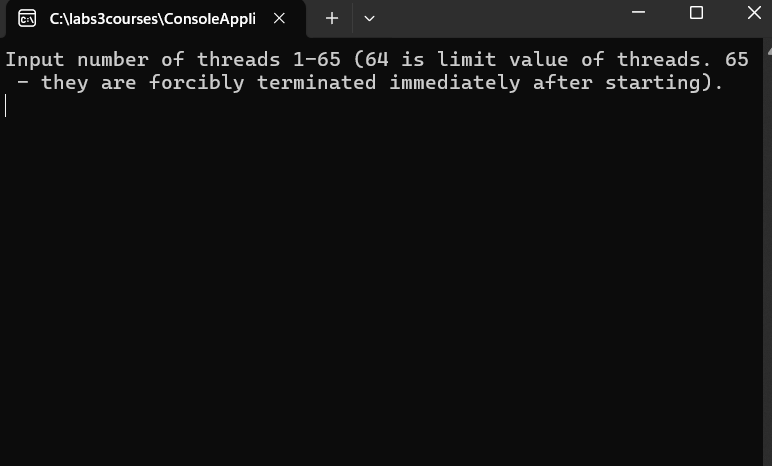


Рисунок 3.1 – Консольное окно с предупреждающим сообщением

Далее после вывода сообщения программа ожидает ввод количества потоков.

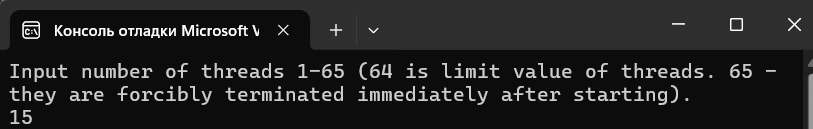


Рисунок 3.2 – Ввод количества потоков

При успешном вводе потоков программа запускается с введенным количеством потоков.

При попытке ввести количество потоков, превышающее число 65 или не превосходящее число 1, программа просит ввести число потоков заново (рисунок 3.3).

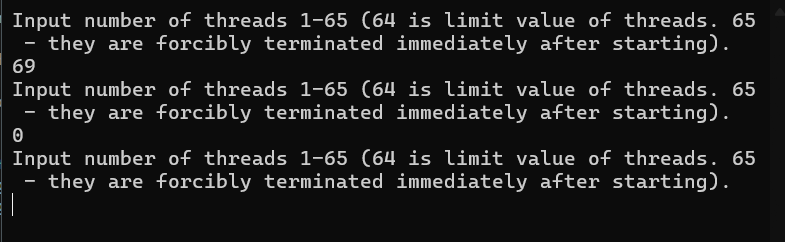


Рисунок 3.3 – Повторный ввод количества потоков

## **3.2 Отображение статусов потоков**

Создание потоков осуществляется с помощью функции CreateThread() [2]. При выполнении потоками арифметических действий программа отображает идентификатор потока, итерацию и его общее время выполнения (рисунок 3.4)

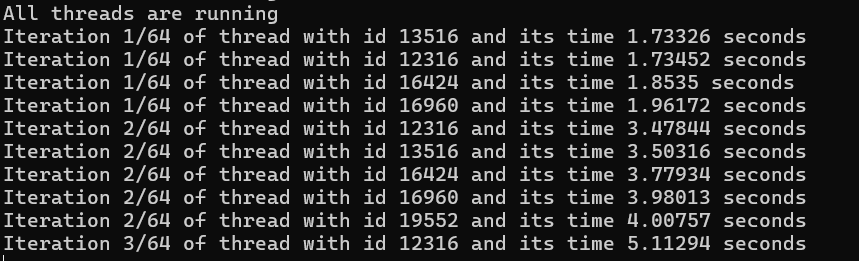


Рисунок 3.4 – Отображение информации о потоках

При выполнении потоками заданных арифметических действий программа отображает время выполнения заданного задания потоком и его производительность и записывает в файл (рисунок 3.5)

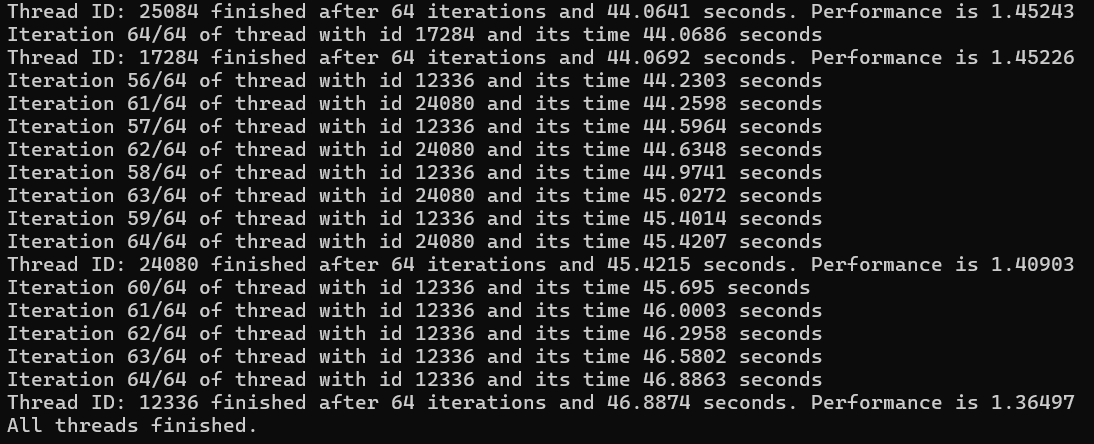


Рисунок 3.5 – Отображение информации о завершенном потоке

После завершения работы всех потоков программа прекращается.

Вся информация о выполнившихся потоках отображается в файле с названием thread\_performance.log (рисунок 3.6)

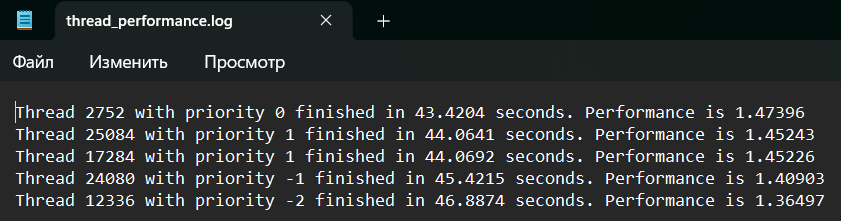


Рисунок 3.6 – Отображение информации о завершенных потоках в файлах

Выставление потоку приоритета 2 не будет являться гарантией, что он выполнится раньше всех, поэтому поток с приоритетом 1 может выполниться раньше потока с приоритетом 2.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были достигнуты поставленные цели, включающие возобновление и развитие навыков программирования с использованием многопоточности. В процессе работы изучены ключевые концепции потоков, их жизненные циклы, а также механизмы управления приоритетами потоков в операционной системе Windows.

Реализованная программа позволяет запускать несколько потоков с различными приоритетами, оценивать их производительность по времени выполнения и количеству выполненных итераций, а также анализировать поведение потоков в зависимости от их приоритета. Для наглядности выполнена регистрация данных по работе каждого потока в консоль и файл, что способствует более детальному анализу и сравнению потоков.

Дополнительно был проведен анализ поведения системы в условиях наличия потоков с разными приоритетами, что позволило увидеть влияние приоритетов на выполнение задач. В результате выполнения работы были закреплены навыки создания и управления потоками, а также проведен анализ многозадачности в операционной системе Windows, что имеет важное значение для дальнейшего изучения и разработки эффективных многопоточных приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] Процессы и потоки – Win32 apps | Microsoft learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/\_processthreadsapi/

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <windows.h>

#include <chrono>

#include <fstream>

//функция, которая будет выполняться потоком (умножение матриц)

DWORD WINAPI FunctionByThread(LPVOID lpParam) {

auto start = std::chrono::high\_resolution\_clock::now();

DWORD ThreadId = GetCurrentThreadId();

int iteration = 0;

int A[4][4] = {

{135, 2344324, 3312312, 4312312},

{5312312, 3123126, 3123127, 3545348},

{5345349, 15345340, 1543531, 12534534},

{15345343, 15345344, 15543534, 16534534}

};

int B[4][4] = {

{16, 15, 14, 13},

{12, 11, 10, 9},

{8, 7, 6, 5},

{4, 3, 2, 1}

};

int C[4][4] = { 0 }; // Матрица результата

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

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

C[i][j] = 0; // Инициализация элемента матрицы результата

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

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

iteration++;

// Симуляция вычислительной нагрузки

for (int l = 0; l < 50000; l++)

{

for (int o = 0; o < 10000; o++)

volatile int temp = o \* l; // Дополнительные вычисления

}

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

std::cout << "Iteration " << iteration << "/64 of thread with id " << ThreadId <<" and its time "<<elapsed.count()<<" seconds"<< std::endl;

}

}

}

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

std::cout << "Thread ID: " << ThreadId << " finished after " << iteration << " iterations and " <<elapsed.count()<<" seconds. Performance is " <<iteration/elapsed.count()<<std::endl;

std::ofstream logFile("thread\_performance.log", std::ios\_base::app); // Открытие файла в режиме добавления

if (logFile.is\_open()) {

logFile << "Thread "<<ThreadId<<" with priority " << GetThreadPriority(GetCurrentThread())

<< " finished in " << elapsed.count() << " seconds. Performance is " << iteration / elapsed.count()<<"\n";

logFile.close();

}

else {

std::cerr << "Error opening file for writing\n";

}

return 0;

}

int main()

{

const int priorities[] = {

THREAD\_PRIORITY\_LOWEST,

THREAD\_PRIORITY\_BELOW\_NORMAL,

THREAD\_PRIORITY\_NORMAL,

THREAD\_PRIORITY\_ABOVE\_NORMAL,

THREAD\_PRIORITY\_HIGHEST

};

const int numPriorities = sizeof(priorities) / sizeof(priorities[0]);

//рандом зависит от времени

srand(static\_cast<unsigned int>(time(NULL)));

//очистка файла для сохранения результата

std::ofstream clearFile("thread\_performance.log", std::ios\_base::trunc);

clearFile.close();

int randomPriority = 0;

int number = -1;

//ввод количества потоков

while (number <= 0 || number>65) {

std::cout << "Input number of threads 1-65 (64 is limit value of threads. 65 - they are forcibly terminated immediately after starting).\n";

std::cin >> number;

}

//создание массива потоков

HANDLE\* Threads = new HANDLE[number];

for (int i = 0; i < number; i++)

{

Threads[i] = CreateThread(

NULL, //атрибут безопасности (стандарт-нулл)

0, //размер стека ( 0 - стандарт)

FunctionByThread, //выполняемая стеком функ

0, //передаваемый параметр

CREATE\_SUSPENDED, //флаг создания (в данном случае в

// приост состоянии) (0 - запуск сразу)

0); //указатель на переменную для айди потока

if (Threads[i] == NULL)

{

std::cout << "Error in creating " << GetThreadId(Threads[i]) << " thread\n";

}

}

//выставление приоритетов для потоков (случайным образом)

for (int k = 0; k < number; k++)

{

randomPriority = priorities[rand() % numPriorities];

if (!SetThreadPriority(Threads[k], randomPriority))

{

std::cout << "Error in setting priority of " << GetThreadId(Threads[k]) << "thread\n";

}

std::cout << "Thread " << GetThreadId(Threads[k]) << " started with " << randomPriority << "\n";

}

//возобновление потоков после выставления приоритетов

for (int i = 0; i < number; i++)

{

if (!ResumeThread(Threads[i]))

{

std::cout << "Error in resuming thread with id " << GetThreadId(Threads[i]) << " \n";

}

std::cout << "Thread " << GetThreadId(Threads[i]) << " is running\n";

}

std::cout << "All threads are running\n";

//ожидание выполнения потоков

WaitForMultipleObjects(number, Threads, TRUE, INFINITE);

// Закрытие самих потоков

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

CloseHandle(Threads[i]);

}

std::cout << "All threads finished." << std::endl;

}