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. 253503  Царук В.А. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc183045248)

[2 Теоретические сведения 4](#_Toc183045249)

[2.1 Процесс, потоки, нити 4](#_Toc183045250)

[3 Ход выполнения программы 5](#_Toc183045251)

[3.1 Примеры выполнения задания 5](#_Toc183045252)

[Вывод 6](#_Toc183045253)

[Список использованных источников 7](#_Toc183045254)

[Приложение А (справочное) Исходный код 8](#_Toc183045255)

# 

# 1 ПОСТАНОВКА ЗАДАЧИ

## 

Целью лабораторной работы является освоение навыков работы с потоками. Была поставлена следующая задача: запуск нескольких потоков с разными приоритетами и оценка их производительности. Количество потоков и их приоритеты можно задать заранее. Отобразить время работы потока и объем выполненной работы.

# 

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Процесс, потоки, нити

Вычислительный процесс – это программа, выполняемая на процессоре. В операционной системе Windows каждый процесс рассматривается как системный объект, который представляет собой выполнение программы с набором ресурсов, включая память, дескрипторы и процессорное время. Состояния процесса включают: запуск (start), выполнение (running), ожидание (waiting) и завершение (terminated) [1].

Вычислительный поток – это наименьшая единица выполнения кода внутри процесса. В Windows каждый поток является системным объектом, создающимся операционной системой и управляемым планировщиком. Потоки разделяют адресное пространство и пространство дескрипторов процесса, к которому принадлежат, что позволяет им совместно работать над данными. Существует несколько состояний потоков: выполнение, готовность, ожидание, завершение и приостановка. Поток может быть приостановлен и возобновлен в любой момент [2].

Нити – это выполнения кода, похожие на потоки, но без прямой поддержки со стороны операционной системы. Их переключение происходит вручную в контексте одного потока. Нити работают в рамках одного потока и разделяют его ресурсы и переключение между ними осуществляется программно, что дает больше контроля над их выполнением [3].

# 3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

## 

## 3.1 Примеры выполнения задания

На рисунке 3.1 представлен результат выполнения программы при запуске программы с тремя потоками и заданными приоритетами: -2, 0, 2.

## 

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Рисунок 3.1 - Результат выполнения программы

# 

# ВЫВОД

В ходе выполнения лабораторной работы были освоены навыки работы с многопоточностью и приоритетами потоков в операционной системе Windows. Была реализована программа, которая создает три потока с разными приоритетами и выполняет в каждом из них вычисления, связанные с умножением матриц. Программа также измеряет время работы каждой задачи для оценки эффективности работы потоков. У каждого потока устанавливается свой приоритет с помощью функции *SetThreadPriority*, что позволяет изучить, как приоритеты влияют на распределение ресурсов процессора между потоками. После завершения всех потоков выводится информация о том, сколько времени заняла их работа.

В процессе работы было обнаружено, что потоки с высоким приоритетом получают больше процессорного времени и завершают свою работу быстрее, а низкоприоритетные наоборот. Также можно отметить, что при выполнении ресурсоемких вычислений многопоточность позволяет эффективно распределить нагрузку на ядра процессора, однако при большом количестве потоков или высокой нагрузке использование многопоточности может не давать значительного преимущества, поскольку расходы на переключение контекста между потоками могут превышать выигрыш от параллельного выполнения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Класс Win32\_Process – Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/cimwin32prov/win32-process – Дата доступа: 14.09.2024

[2] A complete introduction to Windows OS processes, threads and related resources [Электронный ресурс]. – Режим доступа: https://www.tenouk.com/ModuleT.html – Дата доступа: 14.09.2024

[3] Волокна – Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/procthread/fibers – Дата доступа: 14.09.2024

# ПРИЛОЖЕНИЕ А (справочное) Исходный код

#include <windows.h>

#include <iostream>

#include <vector>

#include <chrono>

#define THREAD\_COUNT 3

struct ThreadData {

int threadID;

int priority;

unsigned long long iterations;

double executionTime;

};

void MatrixMultiplication(int size) {

std::vector<std::vector<int>> matrixA(size, std::vector<int>(size, 15));

std::vector<std::vector<int>> matrixB(size, std::vector<int>(size, 20));

std::vector<std::vector<int>> result(size, std::vector<int>(size, 25));

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

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

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

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

}

}

}

}

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

ThreadData\* data = static\_cast<ThreadData\*>(lpParam);

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

for (unsigned long long i = 0; i < data->iterations; ++i) {

MatrixMultiplication(100); // Умножение матриц размером 100x100

if (i % 10 == 0) {

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

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

std::cout << "Thread " << data->threadID << " is running. Time elapsed: "

<< elapsed.count() << " seconds." << std::endl;

}

}

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

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

data->executionTime = elapsed.count();

return 0;

}

int main() {

HANDLE threads[THREAD\_COUNT];

ThreadData threadData[THREAD\_COUNT];

int priorities[THREAD\_COUNT] = { THREAD\_PRIORITY\_LOWEST, THREAD\_PRIORITY\_NORMAL, THREAD\_PRIORITY\_HIGHEST };

for (int i = 0; i < THREAD\_COUNT; ++i) {

threadData[i].threadID = i + 1;

threadData[i].priority = priorities[i];

threadData[i].iterations = 100;

threadData[i].executionTime = 0.0;

threads[i] = CreateThread(

nullptr,

0,

ThreadFunction,

&threadData[i],

0,

nullptr

);

if (threads[i] == nullptr) {

std::cerr << "Error in creating thread " << i + 1 << std::endl;

return 1;

}

SetThreadPriority(threads[i], priorities[i]);

}

WaitForMultipleObjects(THREAD\_COUNT, threads, TRUE, INFINITE);

for (int i = 0; i < THREAD\_COUNT; ++i) {

std::cout << "Thread " << threadData[i].threadID

<< " with priority " << threadData[i].priority

<< " ended. Time of work: "

<< threadData[i].executionTime << " seconds." << std::endl;

CloseHandle(threads[i]);

}

return 0;

}