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, исследование эффективности»

Выполнил:

студент гр. 153504

Князев Н.Д.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 8](#_Toc146631501)

[Список использованных источников 9](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 10](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является создание многозадачного приложения, использующего многопоточность для обработки большого объема данных в параллель.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.

Управление процессами и потоками в операционной системе Windows является важной частью разработки многозадачных приложений.

Создание нового процесса или потока выполняется с использованием функций Windows API, таких как CreateProcess для процессов и CreateThread для потоков. При порождении нового процесса происходит выделение отдельного адресного пространства памяти, файловых ресурсов и других системных ресурсов.

Процессы и потоки могут завершаться по разным причинам. Можно использовать функции, такие как ExitProcess для завершения процесса и ExitThread для завершения потока. Кроме того, процесс или поток могут быть завершены системой из-за ошибок или других обстоятельств.[1]

Windows предоставляет различные уровни приоритетов для процессов и потоков, такие как высокий, выше среднего, нормальный, ниже среднего и низкий. Функция SetPriorityClass позволяет устанавливать приоритеты для процессов, а функция SetThreadPriority для потоков. Например, существует возможность повысить приоритет критически важных задач для обеспечения их выполнения вовремя.[2]

Для изучения эффективности приложения можно использовать возможность подсчета времени работы каждого алгоритма в миллисекундах.

Управление процессами и потоками в Windows – это важный аспект разработки приложений, позволяющий создавать эффективные и отзывчивые многозадачные приложения, а также обеспечивать их стабильную работу.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. В главной программе (проекте) создаются три процесса с помощью функции CreateProcess(). Каждый процесс выполняет свою задачу, описанную в трех отдельных исполняемых файлах: Process1.exe, Process2.exe и Process3.exe.

2. Для каждого процесса создаются отдельные структуры STARTUPINFO и PROCESS\_INFORMATION, которые используются для запуска и отслеживания процессов.

3. В третьем процессе (Process3.exe) устанавливается приоритет с помощью функции SetPriorityClass(). Приоритет выставляется как ABOVE\_NORMAL\_PRIORITY\_CLASS, что означает повышенный приоритет выполнения для этого процесса.

4. Для каждого процесса ожидается его завершение с использованием функции WaitForSingleObject(). Это гарантирует, что главная программа будет ждать завершения всех трех процессов, прежде чем продолжить выполнение.

5. После завершения всех трех процессов закрываются их дескрипторы с помощью CloseHandle().

6. Каждый из трех процессов (Process1.exe, Process2.exe и Process3.exe) выполняет разные вычислительные задачи и собирает статистику о времени выполнения с использованием функций QueryPerformanceCounter() и QueryPerformanceFrequency(). Результаты и время выполнения выводятся в консоль.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано простое консольное многозадачное приложение, которое реализует два потока, которые инкрементируют и декрементируют значение, а также изменяют приоритет потока. Результат работы программы показан на рисунке 3.1.

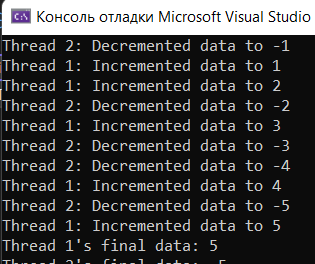


Рисунок 3.1 – Результат работы программы

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было создано простое многозадачное консольное приложение, которое реализует работу 2ух потоков и синхронизирует их с помощью semaphore и mutex.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions – Дата доступа 08.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions%20–%20Дата%20доступа%2008.10.2023)
3. [Электронный ресурс]. – Режим доступа:  [[https://club.shelek.ru/viewart.php?id=71 – Дата доступа 10.10.2023](%20https://club.shelek.ru/viewart.php?id=71%20–%20Дата%20доступа%2010.10.2023)](https://stackoverflow.com/questions/17187265/how-to-group-radio-box-buttons-using-win32-api%20–%20Дата%20доступа%2023.09.2023)

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**main.cpp**

#include <windows.h>

#include <stdio.h>

// Structure to store thread information

struct ThreadData {

int id;

int data;

HANDLE threadHandle;

HANDLE finishedEvent;

};

// Global variables

HANDLE hSemaphore;

HANDLE hMutex;

ThreadData thread1Data;

ThreadData thread2Data;

// Thread function for the first thread

DWORD WINAPI ThreadFunction1(LPVOID lpParam) {

ThreadData\* data = (ThreadData\*)lpParam;

for (int i = 0; i < 5; i++) {

WaitForSingleObject(hSemaphore, INFINITE);

WaitForSingleObject(hMutex, INFINITE);

data->data++;

printf("Thread %d: Incremented data to %d\n", data->id, data->data);

ReleaseMutex(hMutex);

ReleaseSemaphore(hSemaphore, 1, NULL);

Sleep(1000); // Simulate some work

}

SetEvent(data->finishedEvent); // Signal that the thread has finished

return 0;

}

// Thread function for the second thread

DWORD WINAPI ThreadFunction2(LPVOID lpParam) {

ThreadData\* data = (ThreadData\*)lpParam;

for (int i = 0; i < 5; i++) {

WaitForSingleObject(hSemaphore, INFINITE);

WaitForSingleObject(hMutex, INFINITE);

data->data--;

printf("Thread %d: Decremented data to %d\n", data->id, data->data);

ReleaseMutex(hMutex);

ReleaseSemaphore(hSemaphore, 1, NULL);

Sleep(1000); // Simulate some work

}

SetEvent(data->finishedEvent); // Signal that the thread has finished

return 0;

}

int main() {

// Initialize the semaphore and mutex

hSemaphore = CreateSemaphore(NULL, 1, 1, NULL);

hMutex = CreateMutex(NULL, FALSE, NULL);

if (hSemaphore == NULL || hMutex == NULL) {

printf("Failed to create synchronization objects\n");

return 1;

}

// Initialize thread data and events

thread1Data.id = 1;

thread2Data.id = 2;

thread1Data.data = 0;

thread2Data.data = 0;

thread1Data.finishedEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

thread2Data.finishedEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

// Create two threads

thread1Data.threadHandle = CreateThread(NULL, 0, ThreadFunction1, &thread1Data, 0, NULL);

thread2Data.threadHandle = CreateThread(NULL, 0, ThreadFunction2, &thread2Data, 0, NULL);

if (thread1Data.threadHandle == NULL || thread2Data.threadHandle == NULL) {

printf("Failed to create threads\n");

return 2;

}

// Wait for threads to finish

WaitForSingleObject(thread1Data.finishedEvent, INFINITE);

WaitForSingleObject(thread2Data.finishedEvent, INFINITE);

// Cleanup

CloseHandle(thread1Data.threadHandle);

CloseHandle(thread2Data.threadHandle);

CloseHandle(thread1Data.finishedEvent);

CloseHandle(thread2Data.finishedEvent);

CloseHandle(hSemaphore);

CloseHandle(hMutex);

printf("Thread 1's final data: %d\n", thread1Data.data);

printf("Thread 2's final data: %d\n", thread2Data.data);

return 0;

}