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.253505 Косяков М.М.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178093096)

[2 Краткие теоритические сведения 4](#_Toc178093097)

[3 Описание функций программы 5](#_Toc178093098)

[3.1 Выбор файла для чтения и количества потоков 5](#_Toc178093099)

[3.2 Обработка данных файла и анализ временных затрат 6](#_Toc178093100)

[Заключение 7](#_Toc178093101)

[Список использованных источников 8](#_Toc178093102)

[Приложение А (обязательное) Исходный код программы 9](#_Toc178093103)

1 ФОРМУЛИРОВКА ЗАДАЧИ

В рамках этой лабораторной работы ставлю перед собой цель углубить и закрепить навыки программирования приложений для операционной системы Windows. В ходе выполнения работы необходимо изучить концепции вычислительных процессов, потоков и нитей, а также их реализацию в среде Windows. Важными аспектами работы будут основные этапы жизненного цикла процессов и потоков, включая их порождение, завершение, управление состоянием и взаимодействие между ними. Также требуется освоить принципы многозадачности и многопоточности, что играет ключевую роль в разработке производительных приложений.

Для выполнения лабораторной работы с многопоточной обработкой файла в Windows необходимо реализовать приложение, которое будет считывать файл с использованием нескольких потоков, собирать результаты и подсчитывать время выполнения. Программа будет написана на языке программирования C, с использованием WinAPI.

В качестве задачи необходимо выполнить многопоточную обработку файла, а для этого:

– выбрать файла для чтения;

– выбрать количество потоков для считывания файла;

– прочитать файл несколькими потоками;

– собрать результат из нескольких потоков;

– подсчитать время, которое понадобилось на его обработку;

В результате выполнения этой лабораторной работы будут не только получены теоретические знания работы с многопоточностью, но и практический опыт в разработке приложений, которые могут эффективно использовать ресурсы компьютера. Понимание концепций потоков и процессов, а также их управления, станет необходимым для создания высокопроизводительных программных решений.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Процесс представляет собой выполняющуюся программу и является основным объектом выполнения в операционных системах [1]. Каждый процесс может состоять из одного или нескольких потоков, которые выполняются в его контексте. Поток, в свою очередь, является наименьшей единицей выполнения, которая может быть запущена и управляется операционной системой. Он способен выполнять любую часть кода, находящегося в пределах процесса, включая те участки, которые в данный момент обрабатываются другим потоком. Это обеспечивает гибкость и позволяет реализовывать многозадачность, что особенно актуально в современных приложениях, требующих параллельной обработки данных.

Объект задания предоставляет возможность управлять группами процессов как единым целым [2]. Эти объекты представляют собой именуемые, защищаемые и общие ресурсы, которые регулируют атрибуты процессов, связанных с данным объектом задания. С помощью операций, выполняемых с объектом задания, можно влиять на все процессы, которые ассоциированы с ним, что позволяет оптимизировать управление ресурсами и упрощает мониторинг их состояния. Например, можно одновременно приостановить или завершить выполнение всех процессов, связанных с конкретным объектом задания, что существенно упрощает управление многими процессами.

Пул потоков — это структура, состоящая из коллекции рабочих потоков, которые предназначены для эффективного выполнения асинхронных обратных вызовов от имени приложения. Основная цель пула потоков заключается в снижении общего количества активных потоков в приложении и в обеспечении более качественного управления рабочими потоками. Это достигается путем повторного использования потоков, что позволяет избежать накладных расходов, связанных с созданием и уничтожением потоков на каждом этапе выполнения. Пул потоков значительно улучшает производительность, особенно в приложениях, требующих частых и быстрых операций.

Нить является более легковесной единицей выполнения, которую приложение должно запланировать вручную. Нити функционируют в контексте потоков, которые, в свою очередь, осуществляют планирование и управление их выполнением. Это позволяет эффективно распределять задачи и оптимизировать использование процессорного времени, что является критически важным в сценариях, где необходимо быстрое реагирование на события или обработка больших объемов данных.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы [3]:

– выбор файла для запуска (с обработкой ошибок при выборе некорректного файла);

– выбор количества потоков для запуска (с обработкой ошибок при выборе некорректного количества);

– подсчёт времени для считывания файла;

– завершение работы программы;

**3.1 Выбор файла для чтения и количества потоков**

Для начала обработки следует выбрать файл в консольной строке (рисунок 3.1).

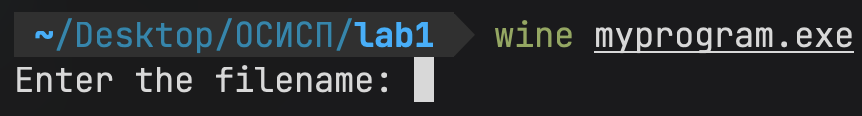


Рисунок 3.1– Выбор файла для обработки

При попытке открыть некорректный файл появится всплывающее окно с сообщением об ошибке (рисунок 3.2).

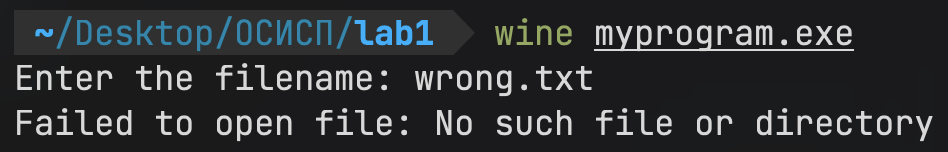


Рисунок 3.2 – Сообщение об ошибке при открытии некорректного файла

Также необходимо выбрать количество потоков, которые понадобятся для обработки файла (рисунок 3.3).

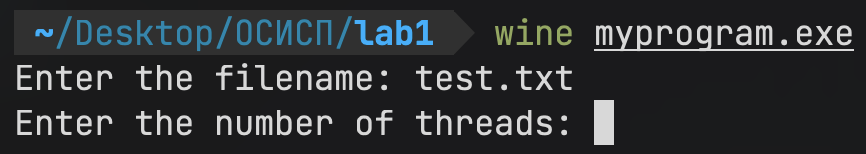


Рисунок 3.3 – Выбор количества потоков

При попытке выбрать некорректное число потоков появится всплывающее окно с сообщением об ошибке (рисунок 3.4).

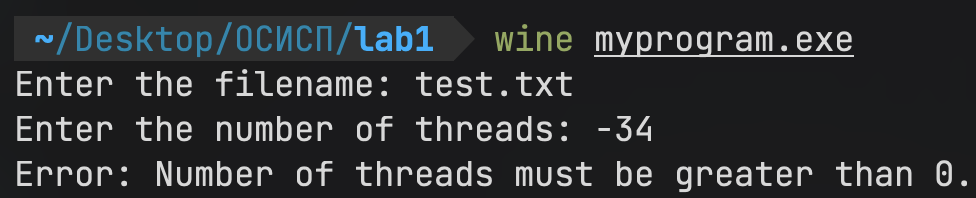


Рисунок 3.4 – Сообщение об ошибке при вводе некорректного числа потоков

**3.2 Обработка данных файла и анализ временных затрат**

Программа считывает размер файла, делит его на сегменты, создает потоки и каждому потоку отводит под обработку свой сегмент, далее выводит информацию о том, как потоки обработали сегменты, а также время необходимое для обработки всего файла. Сборка результата обработки записывается в файл output.bin (рисунок 3.5).

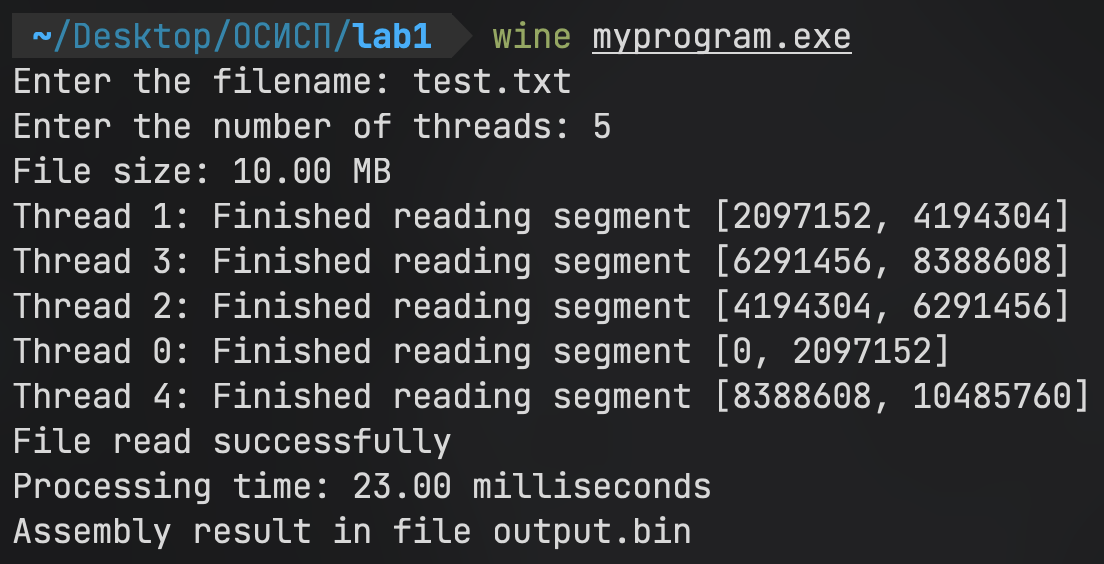


Рисунок 3.5 – Сообщение об обработке файла

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены и закреплены навыки программирования приложений для операционной системы Windows с использованием многопоточности. Мы погрузились в концепции вычислительных процессов, потоков и нитей, и научились их реализации в среде Windows с помощью WinAPI.

Основными аспектами, которыми мы ознакомились, стали этапы жизненного цикла процессов и потоков, включая порождение, завершение, управление состоянием и взаимодействие между ними. Мы также освоили принципы многозадачности и многопоточности, что сыграет важную роль в разработке производительных приложений.

В ходе работы над многопоточной обработкой файла мы выбрали файл для чтения, определили количество потоков для считывания, реализовали чтение файла несколькими потоками, собрали результаты и подсчитали время выполнения операции.

Эта лабораторная работа не только позволила углубить наши теоретические знания в области многопоточности, но и предоставила ценный практический опыт в разработке эффективных приложений, способных эффективно использовать ресурсы компьютера. Понимание концепций потоков и процессов, а также навыки их управления, стали ключевыми в создании высокопроизводительных программных решений.

Выполнение этой лабораторной работы позволило нам не только расширить нашу базу знаний, но и приобрести практические навыки, которые будут полезны в дальнейшей разработке программного обеспечения для Windows.

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] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка приложений с помощью WinAPI. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

// Define a struct to hold thread-specific data

typedef struct

{

char \*buffer; // Pointer to the data buffer

size\_t start; // Start index for processing

size\_t end; // End index for processing

size\_t thread\_id; // Thread id

} ThreadData;

// Function executed by each thread to process a chunk of data

DWORD WINAPI process\_chunk(LPVOID arg)

{

// Cast the argument back to ThreadData pointer

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

// Process the data chunk by XORing each byte with 0xFF

for (size\_t i = data->start; i < data->end; i++)

{

// XOR operation on data

data->buffer[i] ^= 0xFF;

}

printf("Thread %d: Finished reading segment [%d, %d]\n", data->thread\_id, data->start, data->end);

return 0;

}

int main()

{

int num\_threads;

char filename[256] = "test.txt";

printf("Enter the filename: ");

scanf("%255s", filename);

// Open the file for reading

FILE \*file = fopen(filename, "r");

if (!file)

{

perror("Failed to open file");

return EXIT\_FAILURE;

}

printf("Enter the number of threads: ");

scanf("%d", &num\_threads);

if (num\_threads <= 0)

{

fprintf(stderr, "Number of threads must be greater than 0.\n");

fclose(file);

return EXIT\_FAILURE;

}

// Determine the size of the file

fseek(file, 0, SEEK\_END);

size\_t file\_size = ftell(file);

double file\_size\_mb = (double)file\_size / (1024 \* 1024);

printf("File size: %.2f MB\n", file\_size\_mb);

fseek(file, 0, SEEK\_SET);

// Allocate memory to store the file contents

char \*buffer = (char \*)malloc(file\_size);

if (!buffer)

{

perror("Failed to allocate memory");

fclose(file);

return EXIT\_FAILURE;

}

// Read the file contents into the buffer

fread(buffer, 1, file\_size, file);

fclose(file);

// Calculate the chunk size for each thread

size\_t chunk\_size = file\_size / num\_threads;

// Allocate memory for thread handles and thread data

HANDLE \*threads = (HANDLE \*)malloc(num\_threads \* sizeof(HANDLE));

ThreadData \*thread\_data = (ThreadData \*)malloc(num\_threads \* sizeof(ThreadData));

// Start time

clock\_t start\_time = clock();

// Create threads to process different chunks of data

for (int i = 0; i < num\_threads; i++)

{

// Assign data to each thread

thread\_data[i].buffer = buffer;

thread\_data[i].start = i \* chunk\_size;

thread\_data[i].end = (i == num\_threads - 1) ? file\_size : (i + 1) \* chunk\_size;

thread\_data[i].thread\_id = i;

// Create a thread for processing the chunk

threads[i] = CreateThread(NULL, 0, process\_chunk, &thread\_data[i], 0, NULL);

}

// Wait for all threads to finish

WaitForMultipleObjects(num\_threads, threads, TRUE, INFINITE);

printf("File read successfully\n");

// End time

clock\_t end\_time = clock();

double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC \* 1000;

printf("Processing time: %.2f milliseconds\n", elapsed\_time);

// Assembly result

file = fopen("output.bin", "wb");

if (!file)

{

perror("Failed to open output file");

free(buffer);

return EXIT\_FAILURE;

}

fwrite(buffer, 1, file\_size, file);

fclose(file);

printf("Assembly result in file output.bin\n");

// Free allocated memory

free(buffer);

free(threads);

free(thread\_data);

return EXIT\_SUCCESS;

}