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

на тему

Работа с файлами

Выполнил: студент группы 153505

Кудласевич Артур Иванович

Проверил: Сиротко Сергей Иванович

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

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

[Выводы 7](#_Toc146631501)

[Приложение А (обязательное) Листинг кода 8](#_Toc146631503)

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью лабораторной работы является изучение расширенных/дополнительных возможностей и механизмов работы с файлами и организации ввода-вывода: неблокирующие и асинхронные операции, мультиплексирование ввода-вывода, отображение файлов в память, мультиплексирование и др.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Работа процессора считается тиками — единица времени, в течение которой выполняется одна задача. Чтобы прочитать данные из двух ячеек, просуммировать их, записать получившуюся сумму в последнюю ячейку, необходимо 4 тика. Все сложные операции программирования (умножение, деление) состоят из простых. Умножение заменяется сложением числа с самим собой несколько раз.

Современные процессоры позволяют выполнять одну задачу за несколько миллисекунд. Пользователь не замечает времени выполнения, поэтому ему кажется, что все задачи решаются независимо от других. На самом деле процессор выполняет работу, затем переключается к другому действию.

Количество единиц измерения принято считать в герцах (Гц). Данная величина показывает частоту протекания процессов. Современные модели измеряются гигагерцами, включающими триллиарды герцов. Скорость обработки хватает для получения информации, решения множества задач, независимо друг от друга.

При чтении программы, процессор использует множество процессов. Затем выдаёт конечное действие на экран пользователя. Если процесс проходит линейно, каждый последующий блок кода попадал в блокировку. При скачивании фильма, компьютером нельзя пользоваться, пока 100% файла не загрузятся на жёсткий диск. Чтобы устройство могло выполнять другие функции во время чтения кода, все действия делятся на потоки. Для создания программ, позволяющих выполнять несколько действий одновременно, рассмотрим термины асинхронность и многопоточность.

Асинхронность — вид программирования, позволяющий вынести выполняемые задачи отдельными блоками кода. Применяется в сервисах, где предыдущее действие тормозит следующее. Синхронный процесс выполняется поэтапно. Пользователь совершает действие, ждёт, пока программа обработает часть кода, переходит к другому блоку. При использовании асинхронности, код убирает операцию, блокирующую следующие действия.

Отличить асинхронный код можно визуально. В программировании на C#, имеются 2 слова-индикатора, которые показывают, что задачи работают независимо. К индикаторам относятся функции async и await, упрощающие написание асинхронных блоков. Написание асинхронного кода возможно со всеми параметрами. Ограничений в программировании этого типа нет.

Примером асинхронности можно назвать работу с базами данных. Пока пользователю необходима одна информация, остальные блоки отключены. Если человеку нужно совершить новое действие — найти другие данные, код включается. Результат работы кода приходит не сразу после вызова, а при необходимости пользователю просмотреть нужную информацию. Сокращается время обработки процессором.

Термин многопоточности схож с асинхронностью. Он связан с принципом построения программного кода, при котором задачи выполняются одновременно. Многопоточность позволяет нескольким потокам работать в рамках одного процесса. При использовании приложения выделяется 1 поток — главный.

Чтобы устройство работало быстрее, главный поток должен быть свободным от задач, которые имеют большое время продолжительности. Если в компьютере находится только 1 поток, он должен выполнять все действия: отрисовка интерфейса, скачивание данных из интернета, просмотр фильмов, принятие запросов из интернета. Нагрузив систему, пользователь не может выполнять другие задачи. Главный поток выполняет только одно действие. После завершения переходит к следующему.

Используя многопоточность, можно делить ресурс процессора, выполнять несколько задач одновременно.

**3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ     РАБОТЫ**

В рамках выполнения лабораторной работы было создано приложение, которое обрабатывает содержимое файла данных, используя асинхронный ввод-вывод (чтение/запись очередных порций данных параллельно с выполнением обработки данных в памяти).

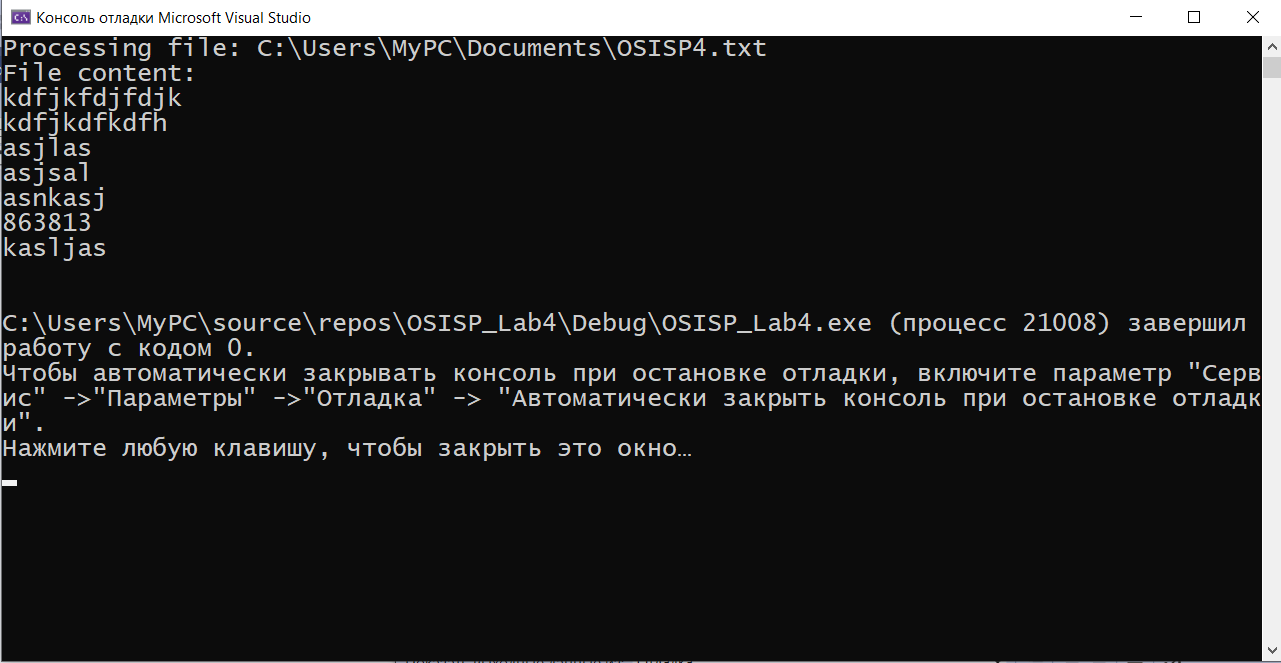


Рисунок 1 – Результат работы программы

**ВЫВОДЫ**

В результате выполнения данной лабораторной работы была разработана программа с использованием C++, которая обрабатывает содержимое файла, используя асинхронные операции.

**ПРИЛОЖЕНИЕ А**

## **(обязательное)**

**Листинг кода**

#include <iostream>

#include <fstream>

#include <string>

#include <future>

std::string read\_file\_content(const std::string& file\_path) {

std::ifstream file(file\_path);

if (!file) {

throw std::runtime\_error("Failed to open the file");

}

std::string content;

std::string line;

while (std::getline(file, line)) {

content += line + "\n";

}

return content;

}

void process\_file\_content(const std::string& file\_path) {

std::cout << "Processing file: " << file\_path << std::endl;

try {

// Асинхронно читаем содержимое файла

auto future = std::async(std::launch::async, read\_file\_content, file\_path);

// Другие асинхронные операции могут быть выполнены в это время

// Получаем результат из будущего (future)

std::string content = future.get();

// Обрабатываем содержимое файла (в данном примере просто выводим его)

std::cout << "File content:\n" << content << std::endl;

}

catch (const std::exception& ex) {

std::cerr << "Error: " << ex.what() << std::endl;

}

}

int main() {

std::string file\_path("C:\\Users\\MyPC\\Documents\\OSISP4.txt");

process\_file\_content(file\_path);

return 0;

}