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 №2

на тему

**РАБОТА С ФАЙЛАМИ**

Студент: Сенько Н.С.

Преподаватель: Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc179805612)

[2 Описание функций программы 4](#_Toc179805613)

[2.1 Асинхронное чтение 4](#_Toc179805614)

[2.2 Обработка прочитанных данных 4](#_Toc179805615)

[2.3 Запись данных в файл 4](#_Toc179805616)

[Заключение 5](#_Toc179805617)

[Список использованных источников 6](#_Toc179805618)

[Приложение А (обязательное) Исходный код программы 7](#_Toc179805619)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Реализация обработки содержимого файла данных, используя асинхронный ввод-вывод (чтение/запись очередных порций данных параллельно с выполнением обработки данных в памяти). Варьирование количества параллельно инициированных операций ввода вывода, влияние их на общею производительность. Поиск «узкого места» в программной реализации. Оценка эффективности (производительности) по сравнению с традиционным подходом (чтение → обработка → выгрузка). Опционально – то же по сравнению с многопоточной реализацией. Возможные варианты обработки: сортировка, статистическая обработка, криптография и т.д. Объем данных следует выбирать достаточно большим, чтобы длительность обработки была заметна и хорошо поддавалась измерению. [1]

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

## **2.1 Асинхронное чтение**

При запуске программы открывается заданный текстовый файл и из него несколько потоков асинхронно считывают данные. [2] (рисунок 2.1).

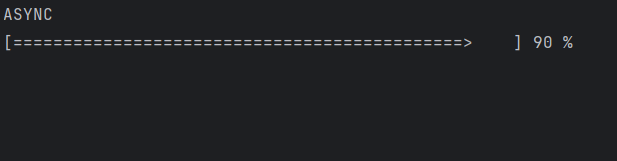
****

Рисунок 2.1 – Процесс асинхронного чтения из файла

## **2.2** **Обработка прочитанных данных**

После чтения файла, где окончание чтения проверяется так, что все потоки чтения будут освобождены, в общий буфер производится подсчет символов, код которых является четным.

## **2.3 Запись данных в файл**

После обработки данные асинхронно записываются в заданный файл. В заранее заданный файл записывается результат. Создаются потоки для записи в файл. [3] Также записывается время всех операций, что изображено на рисунке 2.2. И количество четных символов для проверки.



Рисунок 2.2 – Результат записи данных в файл

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе была реализована обработка содержимого файла данных с использованием асинхронного ввода-вывода (I/O). В ходе эксперимента была выполнена параллельная инициированная загрузка данных и их обработка в памяти, что позволило оптимизировать производительность программы по сравнению с традиционным последовательным подходом (чтение → обработка → выгрузка).

Было проведено варьирование количества одновременно выполняемых операций ввода-вывода для анализа влияния параллелизма на общую производительность системы. Результаты показали, что при увеличении числа параллельных операций достигается ускорение процесса обработки данных, однако существует порог, после которого дальнейшее увеличение количества параллельно выполняемых операций не приводит к значительному улучшению производительности. Это связано с наличием "узкого места" в программной реализации, которое проявляется на уровне ограниченной пропускной способности ввода-вывода или из-за особенностей архитектуры системы, таких как ограниченные ресурсы процессора или памяти.

Сравнительная оценка эффективности асинхронного подхода по сравнению с традиционным показала, что асинхронная реализация значительно быстрее в сценариях, где операции ввода-вывода занимают существенную часть времени. [4]

Таким образом, использование асинхронного ввода-вывода является эффективным методом повышения производительности при обработке больших объемов данных, особенно в случаях, когда операции ввода-вывода занимают продолжительное время.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] CreateFile function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea.

[2] ReadFile function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-readfile.

[3] WriteFile function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-writefile.

[4] Overlapped structure [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-overlapped.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

#include <chrono>

#include <thread>

int oddcnt = 0;

void processData(std::vector<char> &buffer, DWORD bytesRead) {

for (DWORD i = 0; i < bytesRead; ++i) {

buffer[i] = toupper(buffer[i]);

oddcnt += (buffer[i] % 2);

}

}

void displayProgressBar(double progress) {

int barWidth = 50;

std::cout << "[";

int pos = static\_cast<int>(barWidth \* progress);

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

if (i < pos) std::cout << "=";

else if (i == pos) std::cout << ">";

else std::cout << " ";

}

std::cout << "] " << int(progress \* 100.0) << " %\r";

std::cout.flush();

}

HANDLE openFile(const std::string &filename, DWORD desiredAccess, DWORD creationDisposition, DWORD flags) {

HANDLE fileHandle = CreateFile(filename.c\_str(), desiredAccess, 0, nullptr, creationDisposition, flags, nullptr);

if (fileHandle == INVALID\_HANDLE\_VALUE) {

std::cerr << "Error opening file: " << GetLastError() << std::endl;

}

return fileHandle;

}

bool getFileSize(HANDLE fileHandle, LARGE\_INTEGER &fileSize) {

if (!GetFileSizeEx(fileHandle, &fileSize)) {

std::cerr << "Error getting file size: " << GetLastError() << std::endl;

return false;

}

return true;

}

void closeFile(HANDLE fileHandle) {

CloseHandle(fileHandle);

}

void readAndProcessFile(const std::string &inputFilename, const std::string &outputFilename) {

HANDLE hInput = openFile(inputFilename, GENERIC\_READ, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL);

if (hInput == INVALID\_HANDLE\_VALUE) return;

HANDLE hOutput = openFile(outputFilename, GENERIC\_WRITE, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL);

if (hOutput == INVALID\_HANDLE\_VALUE) {

closeFile(hInput);

return;

}

LARGE\_INTEGER fileSize;

if (!getFileSize(hInput, fileSize)) {

closeFile(hInput);

closeFile(hOutput);

return;

}

std::vector<char> buffer(fileSize.QuadPart);

DWORD bytesRead = 0;

DWORD bytesWritten = 0;

LONGLONG totalBytesRead = 0;

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

while (ReadFile(hInput, buffer.data(), fileSize.QuadPart, &bytesRead, nullptr) && bytesRead > 0) {

totalBytesRead += bytesRead;

displayProgressBar(static\_cast<double>(totalBytesRead) / fileSize.QuadPart);

processData(buffer, bytesRead);

if (!WriteFile(hOutput, buffer.data(), bytesRead, &bytesWritten, nullptr)) {

std::cerr << "Error writing file: " << GetLastError() << std::endl;

break;

}

}

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

std::chrono::duration<double> elapsedTime = endTime - startTime;

std::cout << "\nElapsed time: " << elapsedTime.count() << " seconds\n";

closeFile(hInput);

closeFile(hOutput);

}

void readAndProcessFileAsync(const std::string &inputFilename, const std::string &outputFilename, int numStreams) {

HANDLE hInput = openFile(inputFilename, GENERIC\_READ, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED);

if (hInput == INVALID\_HANDLE\_VALUE) return;

HANDLE hOutput = openFile(outputFilename, GENERIC\_WRITE, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED);

if (hOutput == INVALID\_HANDLE\_VALUE) {

closeFile(hInput);

return;

}

LARGE\_INTEGER fileSize;

if (!getFileSize(hInput, fileSize)) {

closeFile(hInput);

closeFile(hOutput);

return;

}

int chunkSize = fileSize.QuadPart / numStreams;

std::vector<OVERLAPPED> olRead(numStreams);

std::vector<OVERLAPPED> olWrite(numStreams);

std::vector<HANDLE> events(numStreams);

std::vector<HANDLE> wevents(numStreams);

std::vector<std::vector<char> > buffers(numStreams, std::vector<char>(chunkSize));

std::vector<DWORD> bytesRead(numStreams, 0);

std::vector<DWORD> bytesWritten(numStreams, 0);

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

events[i] = CreateEvent(nullptr, TRUE, FALSE, nullptr);

olRead[i].hEvent = events[i];

olRead[i].Offset = i \* chunkSize;

}

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

wevents[i] = CreateEvent(nullptr, TRUE, FALSE, nullptr);

olWrite[i].hEvent = events[i];

olWrite[i].Offset = i \* chunkSize;

}

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

LONGLONG totalBytesRead = 0;

bool moreData = true;

while (moreData) {

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

ResetEvent(events[i]);

if (!ReadFile(hInput, buffers[i].data(), chunkSize, nullptr, &olRead[i]) && GetLastError() !=

ERROR\_IO\_PENDING) {

std::cerr << "Error reading file: " << GetLastError() << std::endl;

moreData = false;

break;

}

}

WaitForMultipleObjects(numStreams, events.data(), TRUE, INFINITE);

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

if (!GetOverlappedResult(hInput, &olRead[i], &bytesRead[i], FALSE) || bytesRead[i] == 0) {

moreData = false;

break;

}

totalBytesRead += bytesRead[i];

displayProgressBar(static\_cast<double>(totalBytesRead) / fileSize.QuadPart);

processData(buffers[i], bytesRead[i]);

ResetEvent(events[i]);

if (!WriteFile(hOutput, buffers[i].data(), bytesRead[i], nullptr, &olWrite[i]) && GetLastError() !=

ERROR\_IO\_PENDING) {

std::cerr << "Error writing file: " << GetLastError() << std::endl;

moreData = false;

break;

}

// WaitForSingleObject(olWrite[i].hEvent, INFINITE);

// GetOverlappedResult(hOutput, &olWrite[i], &bytesWritten[i], TRUE);

olRead[i].Offset += numStreams \* chunkSize;

}

WaitForMultipleObjects(numStreams, wevents.data(), TRUE, INFINITE);

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

GetOverlappedResult(hInput, &olWrite[i], &bytesWritten[i], FALSE);

}

}

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

std::chrono::duration<double> elapsedTime = endTime - startTime;

std::cout << "\nElapsed time: " << elapsedTime.count() << " seconds\n";

for (HANDLE event: events) {

CloseHandle(event);

}

closeFile(hInput);

closeFile(hOutput);

}

int main() {

std::string inputFilename = "input.txt";

std::string outputFilename = "output.txt";

int numThreads = 1024;

std::cout << "ASYNC" << std::endl;

readAndProcessFileAsync(inputFilename, outputFilename, numThreads);

std::cout << "ODDS: " << oddcnt << std::endl << std::endl;

oddcnt = 0;

std::cout << "NO ASYNC" << std::endl;

readAndProcessFile(inputFilename, outputFilename);

std::cout << "ODDS: " << oddcnt << std::endl;

getchar();

return 0;

}