Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №7

по дисциплине «Основы операционных систем»

на тему «Работа с файлами в асинхронном режиме»

Выполнили студенты группы 22ВВП1

Демин М. С.

Беляев Д. И.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Название**

Работа с файлами в асинхронном режиме

**Цель работы**

Изучение способов работы с файлами в Win32 API. Изучение асинхронного режима работы с файлами.

**Лабораторное задание**

Разработать программу, копирующую файл из одного каталога в другой. Операции копирования файла должны выполняться одной нитью в асинхронном режиме.

**Описание данных**

1. BUFFER\_SIZE: Константа, определяющая размер буфера для чтения и записи файлов. В данном случае размер равен 4096 байт.

2. CopyFileParams: Структура, содержащая два поля:

• srcPath: Указатель на строку типа WCHAR, представляющую путь к исходному файлу, который нужно скопировать.

• dstPath: Указатель на строку типа WCHAR, представляющую путь к целевому файлу, куда будет осуществляться копирование.

3. OVERLAPPED: Структуры OVERLAPPED (olRead и olWrite) используются для асинхронного ввода-вывода. Каждая из них содержит информацию о состоянии операции ввода-вывода, включая указатель на событие (hEvent), который используется для синхронизации.

4. BYTE buffer[BUFFER\_SIZE]: Массив байтов, используемый для временного хранения данных, считываемых из исходного файла перед записью их в целевой файл.

5. DWORD bytesRead, bytesWritten: Переменные, хранящие количество байт, считанных из исходного файла и записанных в целевой файл соответственно.

6. bool isReading: Логическая переменная, указывающая, продолжается ли процесс чтения данных из исходного файла.

**Описание структуры программы**

1. Функция CopyFileThread:

• Эта функция запускается в отдельном потоке и отвечает за копирование файла из srcPath в dstPath.

• Сначала она открывает исходный файл для чтения и целевой файл для записи. Если не удается открыть один из файлов, выводится сообщение об ошибке, и функция завершает свою работу.

• Создаются события для асинхронных операций чтения и записи.

• В цикле происходит чтение данных из исходного файла и запись их в целевой файл. Если операции чтения или записи выполняются асинхронно, поток ожидает завершения этих операций с помощью WaitForSingleObject.

• После успешного чтения данных из исходного файла и их записи в целевой файл обновляются смещения (Offset) для последующих операций.

• После завершения всех операций закрываются дескрипторы файлов и событий, а также освобождается выделенная память для параметров.

2. Функция main:

• В функции main задаются пути к исходному и целевому файлам.

• Выделяется память для структуры CopyFileParams, в которую записываются пути к файлам.

• Создается новый поток с помощью CreateThread, который запускает функцию CopyFileThread.

• Если создание потока прошло успешно, основной поток ожидает его завершения с помощью WaitForSingleObject. В противном случае выводится сообщение об ошибке.

• После завершения работы потока дескриптор потока закрывается.

**Текст программы**

**main.c [code]**

#include <windows.h>

#include <stdio.h>

#include <stdbool.h>

#include <time.h>

#define BUFFER\_SIZE 67108864 // Размер буфера для чтения и записи (64 MB)

void swapBuffers(char\*\* buffer1, char\*\* buffer2) {

char\* temp = \*buffer1;

\*buffer1 = \*buffer2;

\*buffer2 = temp;

}

void CopyFileAsync(const WCHAR\* srcPath, const WCHAR\* destPath) {

HANDLE hSrcFile = CreateFileW(srcPath, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED | FILE\_FLAG\_NO\_BUFFERING, NULL);

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

fwprintf(stderr, L"Failed to open source file. Error: %lu\n", GetLastError());

return;

}

HANDLE hDestFile = CreateFileW(destPath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED | FILE\_FLAG\_NO\_BUFFERING, NULL);

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

fwprintf(stderr, L"Failed to open destination file. Error: %lu\n", GetLastError());

CloseHandle(hSrcFile);

return;

}

OVERLAPPED overlappedRead = { 0 };

OVERLAPPED overlappedWrite = { 0 };

overlappedRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

overlappedWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

if (overlappedRead.hEvent == NULL || overlappedWrite.hEvent == NULL) {

fwprintf(stderr, L"Failed to create events. Error: %lu\n", GetLastError());

CloseHandle(hSrcFile);

CloseHandle(hDestFile);

return;

}

char\* buffer1 = (char\*)\_aligned\_malloc(BUFFER\_SIZE, BUFFER\_SIZE);

char\* buffer2 = (char\*)\_aligned\_malloc(BUFFER\_SIZE, BUFFER\_SIZE);

if (buffer1 == NULL || buffer2 == NULL) {

fwprintf(stderr, L"Failed to allocate aligned buffers.\n");

CloseHandle(hSrcFile);

CloseHandle(hDestFile);

return;

}

char\* currentBuffer = buffer1;

char\* processingBuffer = buffer2;

DWORD bytesRead = 0, bytesWritten = 0, bytesReadBefore = 0;

LARGE\_INTEGER fileSize;

GetFileSizeEx(hSrcFile, &fileSize);

LONGLONG remainingBytes = fileSize.QuadPart;

wprintf(L"Starting file copy...\n");

clock\_t startTime = clock();

for (int i = 0; remainingBytes > 0; i++) {

if (!ReadFile(hSrcFile, currentBuffer, BUFFER\_SIZE, &bytesRead, &overlappedRead)) {

DWORD error = GetLastError();

if (error != ERROR\_IO\_PENDING && error != ERROR\_HANDLE\_EOF) {

fwprintf(stderr, L"ReadFile failed. Error: %lu\n", error);

break;

}

}

if (i == 0) {

WaitForSingleObject(overlappedRead.hEvent, INFINITE);

if (!GetOverlappedResult(hSrcFile, &overlappedRead, &bytesRead, FALSE)) {

fwprintf(stderr, L"GetOverlappedResult failed for ReadFile. Error: %lu\n", GetLastError());

break;

}

if (bytesRead == 0) {

break; // достигнут конец файла

}

swapBuffers(&currentBuffer, &processingBuffer);

overlappedRead.Offset += bytesRead;

bytesReadBefore = bytesRead;

continue;

}

if (!WriteFile(hDestFile, processingBuffer, bytesReadBefore, &bytesWritten, &overlappedWrite)) {

if (GetLastError() != ERROR\_IO\_PENDING) {

fwprintf(stderr, L"WriteFile failed. Error: %lu\n", GetLastError());

break;

}

}

WaitForSingleObject(overlappedWrite.hEvent, INFINITE);

if (!GetOverlappedResult(hDestFile, &overlappedWrite, &bytesWritten, FALSE)) {

fwprintf(stderr, L"GetOverlappedResult failed for WriteFile. Error: %lu\n", GetLastError());

break;

}

WaitForSingleObject(overlappedRead.hEvent, INFINITE);

if (!GetOverlappedResult(hSrcFile, &overlappedRead, &bytesRead, FALSE)) {

DWORD error = GetLastError();

if (error == ERROR\_HANDLE\_EOF) {

break; // достигнут конец файла

}

fwprintf(stderr, L"GetOverlappedResult failed for ReadFile. Error: %lu\n", error);

break;

}

if (bytesRead == 0) {

break; // достигнут конец файла

}

bytesReadBefore = bytesRead;

overlappedRead.Offset += bytesRead;

overlappedWrite.Offset += bytesWritten;

swapBuffers(&currentBuffer, &processingBuffer);

remainingBytes -= bytesRead;

}

clock\_t endTime = clock();

double elapsed = ((double)(endTime - startTime)) / CLOCKS\_PER\_SEC;

wprintf(L"File copy completed.\n");

wprintf(L"Time taken: %.2f seconds.\n", elapsed);

CloseHandle(hSrcFile);

CloseHandle(hDestFile);

CloseHandle(overlappedRead.hEvent);

CloseHandle(overlappedWrite.hEvent);

\_aligned\_free(buffer1);

\_aligned\_free(buffer2);

}

int wmain() {

const WCHAR\* srcPath = L"E:\\Study\\InstituteRep\\semester\_5\\OOS\\lab7\\from\\temp.txt";

const WCHAR\* dstPath = L"E:\\Study\\InstituteRep\\semester\_5\\OOS\\lab7\\to\\temp.txt";

CopyFileAsync(srcPath, dstPath);

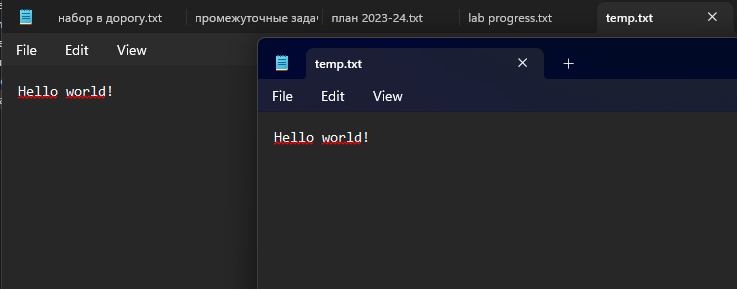
wprintf(L"Press any key to exit...");

getchar(); // Ожидание ввода символа

return 0;

}

**Результат работы программы**



**Вывод**

Изучили способы работы с файлами в Win32 API. Изучили асинхронный режим работы с файлами