Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Кафедра вычислительной техники

Отчёт

по лабораторной работе № 3

на тему:

“Управление файловой системой в Windows”

по дисциплине “Операционные системы”

Выполнил студент гр. 4306:

Табаков А.В.

Принял: Тимофеев А.В.

Санкт-Петербург  
2016

**Цель работы**: исследовать управление файловой системой с помощью Win32 API.

**Задание 3.1.** Управление дисками, каталогами и файлами.

В соответствии с заданием, в программе написано 11 функций:

1 - Вывод списка дисков

2 - Вывести информацию о диске

3 - Создать каталог

4 - Удалить каталог

5 - Создать файл

6 - Копировать файл

7 - Переместить файл

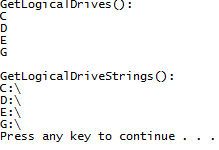
8 - Информация о файле

9 - Изменить атрибуты файла

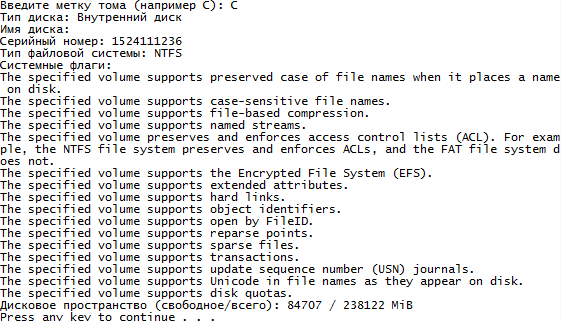
10 - Изменить время создания файла

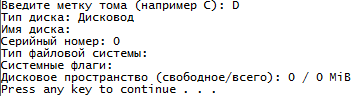
11 - Асинхронное копирование файла

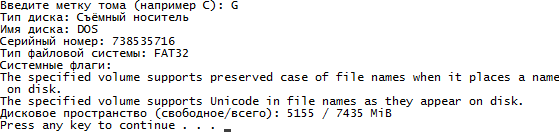
1 – Вывод списка дисков:



2- Вывести информацию о диске:







3 - Создать каталог:



4 - Удалить каталог:



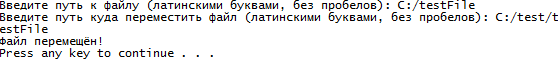
5 - Создать файл:



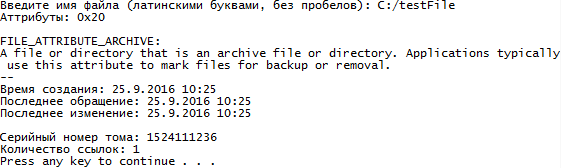
6 - Копировать файл:



7 - Переместить файл:

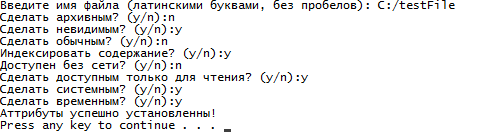


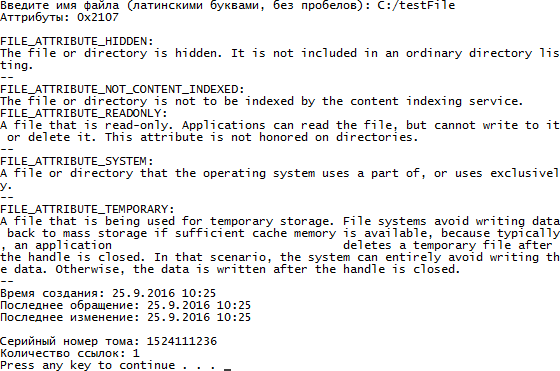
8 - Информация о файле:



9 - Изменить атрибуты файла:

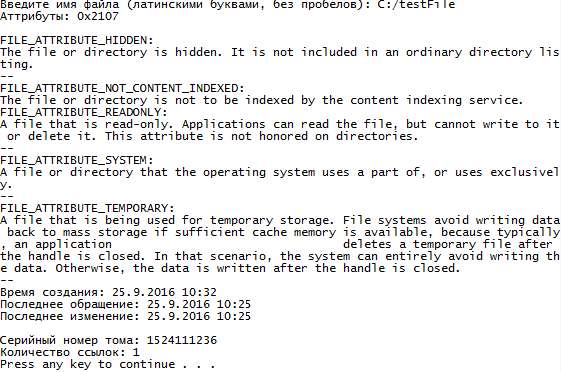
Изменим атрибуты и снова вызовем функцию информация о файле





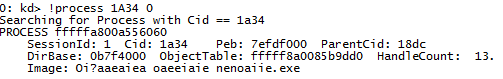
10 - Изменить время создания файла:





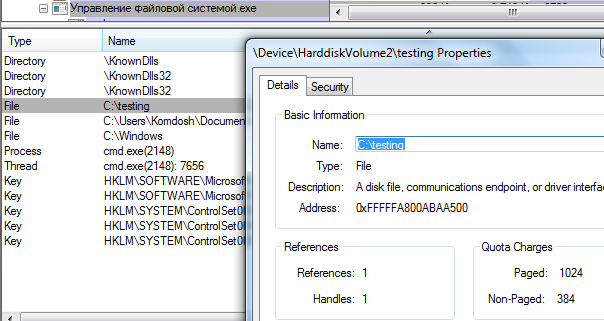
Определение процесса:

Стоит оговориться, что сейчас для исследования мы не будем вызывать СloseHandle для ново созданного файла, чтобы наблюдать изменения, по-хорошему после любой манипуляции над объектами, их всегда надо закрывать.

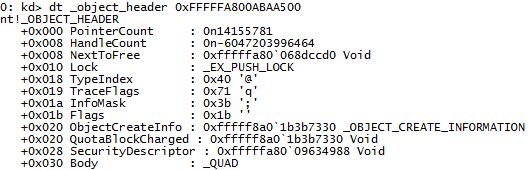


При запуске программы, у неё имеется 13 объектов.

После создания файла появляется 14 объект, обработчик нашего файла



Информация о созданном файле:

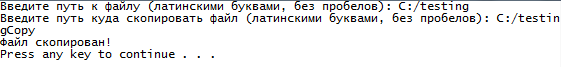




Мы не сможем проводить манипуляции над файлом, пока не закроем его обработчик:



После закрытия обработчика:



Закрытие дескриптора не влияет на работу программы, поскольку ссылки на файлы открываются заново для каждого действия, а после просто забываются.

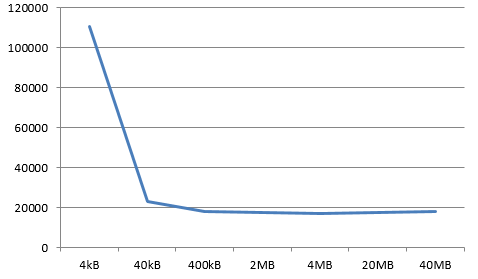
**Вывод**: Операционная система Windows позволяет разработчикам легко работать с файловой системой посредствам WinAPI.

**Задание 3.2.** Копирование файла с помощью операций перекрывающего ввода\вывода

11 - Асинхронное копирование файла

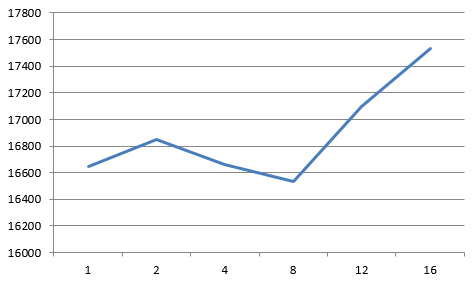
Эксперимент 1: SSD – размер файла 4765 Мб, размер кластера – 4 Кб, чтение и запись в пределах одного диска. Чем больше размер блока, тем меньше происходит операций переключения чтения\запись.

По вертикали миллисекунды, по горизонтали размер блока, самый оптимальный вариант, когда размер блока равен 4 Мб.



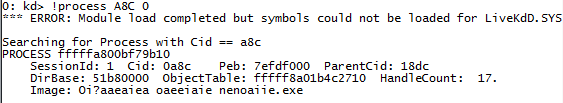
Возьмём размер блока 4 Мб и будем изменять количество перекрывающих операций:

По вертикали – миллисекунды, по горизонтали – количество перекрывающих операций

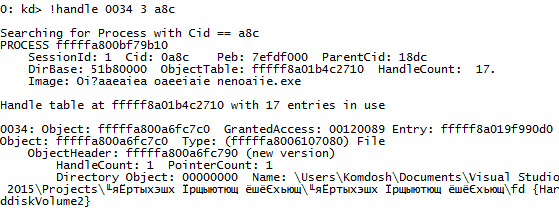


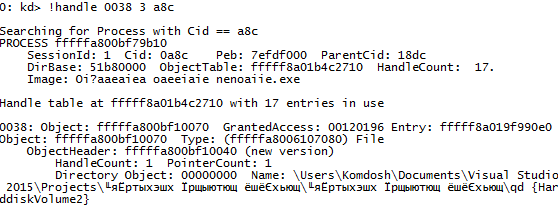
На данном графике видим, что количество перекрывающих операций не сильно улучшают время копирования. Самый оптимальный вариант 8 потоков.

Не будем закрывать обработчики копирования, чтобы исследовать процесс.



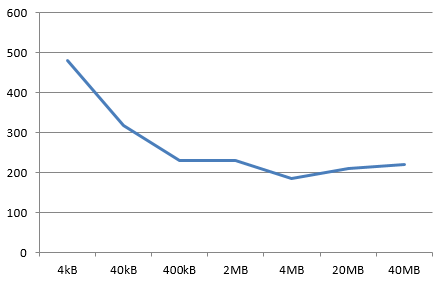
Как видим программа имеет 17 объектов после асинхронного копирования, хотя изначально она имеет только 13 объектов. Эти объекты, наши файлы (скопированный и оригинал)





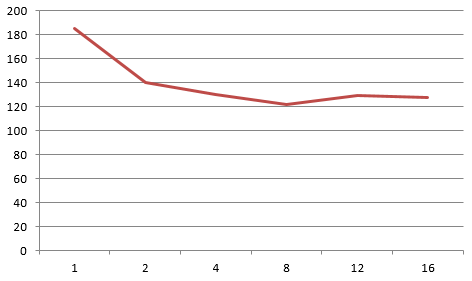
Эксперимент 2: HDD – размер файла 5139 Мб, размер кластера – 4 Кб, чтение и запись в пределах одного диска. Чем больше размер блока, тем меньше происходит операций переключения чтения\запись.

По вертикали секунды, по горизонтали размер блока, самый оптимальный вариант, когда размер блока равен 4 Мб.



Возьмём размер блока 4 Мб и будем изменять количество перекрывающих операций:

По вертикали – секунды, по горизонтали – количество перекрывающих операций



**Вывод по экспериментам**: если сравнить графики SSD и HDD, то можно заметить, что операции случайного обращения к памяти в HDD, происходит гораздо медленнее, чем в SSD, следовательно, правильные коэффициенты дают заметный прирост производительности.

**Вывод**: всё уже придумано и продумано за нас. Наша задача как разработчиков просто использовать уже написанный API, так как над его разработкой работали до нас многие программисты, они уже всё продумали и подобрали правильные параметры, а мы можем использовать неоптимальные решения.

**Исходный код программы**

**Main.cpp**

#include <iostream>

#include "FileSystemAPI.h"

#include "AsyncCopy.h"

using namespace std;

int menu();

int main() {

setlocale(0, ".1251");

int notExit;

do {

switch (notExit = menu())

{

case 1:

showDrivesList();

break;

case 2:

showDriveInfo();

break;

case 3:

createFolder();

break;

case 4:

deleteFolder();

break;

case 5:

createFile();

break;

case 6:

copyFile();

break;

case 7:

moveFile();

break;

case 8:

fileInfo();

break;

case 9:

changeFileAttributes();

break;

case 10:

changeCreationTime();

break;

case 11:

asyncCopyOfFile();

break;

case 0:

break;

default:

if(notExit)

cout << "Такого варианта нет, повторите ввод" << endl;

}

if(notExit)

system("pause");

} while (notExit);

cin.get();

return 0;

}

int menu()

{

system("cls");

int point;

do {

cin.clear();

cin.sync();

cout << "Выберите пункт меню" << endl;

cout << "1 - Вывод списка дисков" << endl;

cout << "2 - Вывести информацию о диске" << endl;

cout << "3 - Создать каталог" << endl;

cout << "4 - Удалить каталог" << endl;

cout << "5 - Создать файл" << endl;

cout << "6 - Копировать файл" << endl;

cout << "7 - Переместить файл" << endl;

cout << "8 - Информация о файле" << endl;

cout << "9 - Изменить атрибуты файла" << endl;

cout << "10 - Изменить время создания файла" << endl;

cout << "11 - Асинхронное копирование файла" << endl;

cout << "0 - Выход" << endl;

cout << "> ";

cin >> point;

if (cin.fail())

cout << "Что-то пошло не так, выберите пункт меню повторно" << endl;

} while (cin.fail());

system("cls");

return point;

}

**AsyncCopy.h**

#pragma once

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

void asyncCopyOfFile();

VOID WINAPI asyncRead(DWORD Code, DWORD nBytes, LPOVERLAPPED lpOv);

VOID WINAPI asyncWrite(DWORD Code, DWORD nBytes, LPOVERLAPPED lpOv);

**FileSystemAPI.h**

#pragma once

#define \_WIN32\_WINNT 0x501

#include <iostream>

#include <windows.h>

#include <io.h>

#include <stdio.h>

#include <tchar.h>

#include <string>

#include <sstream>

void showDrivesList();

void showDriveInfo();

void createFolder();

void deleteFolder();

void createFile();

void copyFile();

void moveFile();

void fileInfo();

void changeFileAttributes();

void changeCreationTime();

#if (\_WIN32 >= 0x0500)

WINBASEAPI BOOL WINAPI GetFileSizeEx(HANDLE, PLARGE\_INTEGER);

#endif

VOID WINAPI asyncRead(DWORD Code, DWORD nBytes, LPOVERLAPPED pOv);

VOID WINAPI asyncWrite(DWORD Code, DWORD nBytes, LPOVERLAPPED pOv);

**FileSystemAPI.cpp**

#include "FileSystemAPI.h"

using namespace std;

void showDrivesList() {

int n;

char driveLetter;

DWORD dr = GetLogicalDrives();

cout << "GetLogicalDrives():\n";

for (int i = 0; i < 26; i++) {

n = ((dr >> i) & 0x1);

if (n == 1) {

driveLetter = char(65 + i);

cout << driveLetter << endl;

}

}

cout << "\nGetLogicalDriveStrings():\n";

wchar\_t drives[256];

wchar\_t \*drive;

DWORD sizebuf = 256;

GetLogicalDriveStrings(sizebuf, drives);

drive = drives;

while (\*drive) {

wprintf(L"%s\n", drive); //L:\ 3 символа

drive = drive + wcslen(drive) + 1;

}

}

void showDriveInfo() {

char driveLetter[100];

wchar\_t driveLetterWchar[100];

cout << "Введите метку тома (например C): ";

cin >> driveLetter;

driveLetter[1] = ':';

driveLetter[2] = '\\';

driveLetter[3] = 0;

mbstowcs(driveLetterWchar, driveLetter, 2);

driveLetterWchar[2] = 0;

unsigned int driveType = GetDriveType(driveLetterWchar);

cout << "Тип диска: ";

switch (driveType) {

case DRIVE\_UNKNOWN:

cout << "Неизвестный тип\n";

return;

case DRIVE\_NO\_ROOT\_DIR:

cout << "Диска с такой меткой не существует\n";

return;

case DRIVE\_REMOVABLE:

cout << "Съёмный носитель";

break;

case DRIVE\_FIXED:

cout << "Внутренний диск";

break;

case DRIVE\_REMOTE:

cout << "Удалённый диск";

break;

case DRIVE\_CDROM:

cout << "Дисковод";

break;

case DRIVE\_RAMDISK:

cout << "RAM диск";

break;

}

cout << endl;

char volumeNameBuffer[100];

volumeNameBuffer[0] = 0;

char fileSystemNameBuffer[100];

fileSystemNameBuffer[0] = 0;

DWORD maxComponentLength = 0, systemFlags = 0; //fs - системные флаги

unsigned long drive\_sn = 0;

GetVolumeInformationA(driveLetter, volumeNameBuffer, 100, &drive\_sn, &maxComponentLength, &systemFlags, fileSystemNameBuffer, 100); //ANSI

cout << "Имя диска: " << volumeNameBuffer << endl <<

"Серийный номер: " << drive\_sn << endl <<

"Тип файловой системы: " << fileSystemNameBuffer << endl <<

"Системные флаги: " <<endl;

string TSV = "The specified volume";

string TSVS = TSV +" supports";

if (systemFlags & FILE\_CASE\_PRESERVED\_NAMES)

cout<<TSVS+" preserved case of file names when it places a name on disk.\n";

if (systemFlags & FILE\_CASE\_SENSITIVE\_SEARCH)

cout<< TSVS + " case-sensitive file names.\n";

if (systemFlags & FILE\_FILE\_COMPRESSION)

cout << TSVS + " file-based compression.\n";

if (systemFlags & FILE\_NAMED\_STREAMS)

cout << TSVS + " named streams.\n";

if (systemFlags & FILE\_PERSISTENT\_ACLS)

cout << TSV + " preserves and enforces access control lists (ACL). For example, the NTFS file system preserves and enforces ACLs, and the FAT file system does not.\n";

if (systemFlags & FILE\_READ\_ONLY\_VOLUME)

cout << TSV + " is read-only.\n";

if (systemFlags & FILE\_SEQUENTIAL\_WRITE\_ONCE)

cout << TSVS + " a single sequential write.\n";

if (systemFlags & FILE\_SUPPORTS\_ENCRYPTION)

cout << TSVS + " the Encrypted File System (EFS).\n";

if (systemFlags & FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES)

cout << TSVS + " extended attributes.\n";

if (systemFlags & FILE\_SUPPORTS\_HARD\_LINKS)

cout << TSVS + " hard links. \n";

if (systemFlags & FILE\_SUPPORTS\_OBJECT\_IDS)

cout << TSVS + " object identifiers.\n";

if (systemFlags & FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID)

cout << TSVS + " open by FileID.\n";

if (systemFlags & FILE\_SUPPORTS\_REPARSE\_POINTS)

cout << TSVS + " reparse points.\n";

if (systemFlags & FILE\_SUPPORTS\_SPARSE\_FILES)

cout << TSVS + " sparse files.\n";

if (systemFlags & FILE\_SUPPORTS\_TRANSACTIONS)

cout << TSVS + " transactions.\n";

if (systemFlags & FILE\_SUPPORTS\_USN\_JOURNAL)

cout << TSVS + " update sequence number (USN) journals.\n";

if (systemFlags & FILE\_UNICODE\_ON\_DISK)

cout << TSVS + " Unicode in file names as they appear on disk.\n";

if (systemFlags & FILE\_VOLUME\_IS\_COMPRESSED)

cout << TSV + " is a compressed volume, for example, a DoubleSpace volume.\n";

if (systemFlags & FILE\_VOLUME\_QUOTAS)

cout << TSVS + " disk quotas.\n";

DWORD sectorsPerCluster, bytesPerSector, freeClusters, totalClusters;

GetDiskFreeSpaceA(driveLetter, &sectorsPerCluster, &bytesPerSector, &freeClusters, &totalClusters);

unsigned \_\_int64 free = freeClusters \* sectorsPerCluster / 1024 \* bytesPerSector / 1024;

unsigned \_\_int64 total = totalClusters\* sectorsPerCluster / 1024 \* bytesPerSector / 1024;

cout << "Дисковое пространство (свободное/всего): " << free << " / " << total << " MiB\n";

}

bool isDirectoryExists(const wchar\_t \*filename)

{

DWORD dwFileAttributes = GetFileAttributes(filename);

if (dwFileAttributes == 0xFFFFFFFF)

return false;

return dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY;

}

void createFolder() {

wchar\_t directoryName[250];

char directoryCharName[250];

cout << "Введите название папки (латинскими буквами, без пробелов): ";

cin >> directoryCharName;

mbstowcs(directoryName, directoryCharName, 250);

if (CreateDirectory(directoryName, NULL) != 0)

cout << "Папка создана\n";

else

cout << "Ошибка, папка не создалась!\n";

}

void deleteFolder() {

wchar\_t directoryName[250];

char directoryCharName[250];

cout << "Введите название папки (латинскими буквами, без пробелов): ";

cin >> directoryCharName;

mbstowcs(directoryName, directoryCharName, 250);

if (isDirectoryExists(directoryName)) {

if (RemoveDirectory(directoryName) != 0)

cout << "Папка удалена\n";

else

cout << "Ошибка, папка не была удалена!\n";

}

else

cout << "Такой папки не существует!\n";

}

void createFile() {

wchar\_t fileName[250];

char fileCharName[250];

cout << "Введите имя файла (латинскими буквами, без пробелов): ";

cin >> fileCharName;

mbstowcs(fileName, fileCharName, 250);

HANDLE hFile = CreateFile(fileName, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != 0)

cout << "Файл создан!\n";

else

cout << "Ошибка, файл не был создан!\n";

CloseHandle(hFile);

}

void copyFile() {

wchar\_t source[250], destination[250];

char sourceChar[250], destinationChar[250];

cout << "Введите путь к файлу (латинскими буквами, без пробелов): ";

cin >> sourceChar;

cout << "Введите путь куда скопировать файл (латинскими буквами, без пробелов): ";

cin >> destinationChar;

mbstowcs(source, sourceChar, 250);

mbstowcs(destination, destinationChar, 250);

if (CopyFile(source, destination, false) != 0)

cout << "Файл скопирован!\n";

else

cout << "Произошла ошибка, файл не был скопирован!\n";

}

void moveFile() {

wchar\_t source[250], destination[250];

char sourceChar[250], destinationChar[250];

cout << "Введите путь к файлу (латинскими буквами, без пробелов): ";

cin >> sourceChar;

cout << "Введите путь куда переместить файл (латинскими буквами, без пробелов): ";

cin >> destinationChar;

mbstowcs(source, sourceChar, 250);

mbstowcs(destination, destinationChar, 250);

if (MoveFile(source, destination) != 0)

cout << "Файл перемещён!\n";

else

cout << "Произошла ошибка, файл не был перемещён!\n";

}

void fileInfo() {

wchar\_t fileName[250];

char fileCharName[250];

cout << "Введите имя файла (латинскими буквами, без пробелов): ";

cin >> fileCharName;

mbstowcs(fileName, fileCharName, 250);

ostringstream tempStringStream;

DWORD fileAttributes;

fileAttributes = GetFileAttributes(fileName);

tempStringStream << "0x";

tempStringStream << hex << fileAttributes << "\n";

cout << "Аттрибуты: " << tempStringStream.str() << endl;

if (fileAttributes & FILE\_ATTRIBUTE\_ARCHIVE)

cout<<"FILE\_ATTRIBUTE\_ARCHIVE:\nA file or directory that is an archive file or directory. Applications typically use this attribute to mark files for backup or removal.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_COMPRESSED)

cout<<"FILE\_ATTRIBUTE\_COMPRESSED:\nA file or directory that is compressed. For a file, all of the data in the file is compressed. For a directory, compression is the default for newly created files and subdirectories.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_DEVICE)

cout<<"FILE\_ATTRIBUTE\_DEVICE:\nThis value is reserved for system use.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

cout<<"FILE\_ATTRIBUTE\_DIRECTORY:\nThe handle that identifies a directory.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_ENCRYPTED)

cout<<"FILE\_ATTRIBUTE\_ENCRYPTED:\nA file or directory that is encrypted. For a file, all data streams in the file are encrypted. For a directory, encryption is the default for newly created files and subdirectories.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_HIDDEN)

cout<<"FILE\_ATTRIBUTE\_HIDDEN:\nThe file or directory is hidden. It is not included in an ordinary directory listing.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_INTEGRITY\_STREAM)

cout<<"FILE\_ATTRIBUTE\_INTEGRITY\_STREAM:\nThe directory or user data stream is configured with integrity (only supported on ReFS volumes). It is not included in an ordinary directory listing. The integrity setting persists \

with the file if it's renamed. If a file is copied the destination file will have integrity set if either the source file or destination directory have integrity set.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_NORMAL)

cout<<"FILE\_ATTRIBUTE\_NORMAL:\nA file that does not have other attributes set. This attribute is valid only when used alone.\n";

if (fileAttributes & FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED)

cout<<"FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED:\nThe file or directory is not to be indexed by the content indexing service.\n";

if (fileAttributes & FILE\_ATTRIBUTE\_NO\_SCRUB\_DATA)

cout<<"FILE\_ATTRIBUTE\_NO\_SCRUB\_DATA:\nThe user data stream not to be read by the background data integrity scanner (AKA scrubber). When set on a directory it only provides inheritance. This flag is only supported on\

Storage Spaces and ReFS volumes. It is not included in an ordinary directory listing.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_OFFLINE)

cout<<"FILE\_ATTRIBUTE\_OFFLINE:\nThe data of a file is not available immediately. This attribute indicates that the file data is physically moved to offline storage. This attribute is used by Remote Storage,\

which is the hierarchical storage management software. Applications should not arbitrarily change this attribute.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_READONLY)

cout<<"FILE\_ATTRIBUTE\_READONLY:\nA file that is read-only. Applications can read the file, but cannot write to it or delete it. This attribute is not honored on directories.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_REPARSE\_POINT)

cout<<"FILE\_ATTRIBUTE\_REPARSE\_POINT:\nA file or directory that has an associated reparse point, or a file that is a symbolic link.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_SPARSE\_FILE)

cout<<"FILE\_ATTRIBUTE\_SPARSE\_FILE:\nA file that is a sparse file.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_SYSTEM)

cout<<"FILE\_ATTRIBUTE\_SYSTEM:\nA file or directory that the operating system uses a part of, or uses exclusively.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_TEMPORARY)

cout<<"FILE\_ATTRIBUTE\_TEMPORARY:\nA file that is being used for temporary storage. File systems avoid writing data back to mass storage if sufficient cache memory is available, because typically, an application\

deletes a temporary file after the handle is closed. In that scenario, the system can entirely avoid writing the data. Otherwise, the data is written after the handle is closed.\n--\n";

if (fileAttributes & FILE\_ATTRIBUTE\_VIRTUAL)

cout<<"FILE\_ATTRIBUTE\_VIRTUAL:\nThis value is reserved for system use.\n--\n";

FILE\* pfile = fopen(fileCharName, "r");

HANDLE hFile = (HANDLE)\_get\_osfhandle(\_fileno(pfile));

if (hFile == NULL){

cout << "Невозможно получить обработчик файла!\n";

return;

}

FILETIME fileCreatedTime;

SYSTEMTIME fileCreatedSystemTime;

wchar\_t createdLocalDate[255];

wchar\_t createdLocalTime[255];

FILETIME fileAccessedTime;

SYSTEMTIME fileAccessedSystemTime;

wchar\_t accessedLocalDate[255];

wchar\_t accessedLocalTime[255];

FILETIME fileWritedTime;

SYSTEMTIME fileWritedSystemTime;

wchar\_t writedLocalDate[255];

wchar\_t writedLocalTime[255];

if (GetFileTime(hFile, &fileCreatedTime, &fileAccessedTime, &fileWritedTime) != 0) {

FileTimeToLocalFileTime(&fileCreatedTime, &fileCreatedTime);

FileTimeToLocalFileTime(&fileAccessedTime, &fileAccessedTime);

FileTimeToLocalFileTime(&fileWritedTime, &fileWritedTime);

FileTimeToSystemTime(&fileCreatedTime, &fileCreatedSystemTime);

FileTimeToSystemTime(&fileAccessedTime, &fileAccessedSystemTime);

FileTimeToSystemTime(&fileWritedTime, &fileWritedSystemTime);

GetDateFormat(LOCALE\_USER\_DEFAULT, DATE\_LONGDATE, &fileCreatedSystemTime, NULL, createdLocalDate, 255);

GetDateFormat(LOCALE\_USER\_DEFAULT, DATE\_LONGDATE, &fileAccessedSystemTime, NULL, accessedLocalDate, 255);

GetDateFormat(LOCALE\_USER\_DEFAULT, DATE\_LONGDATE, &fileWritedSystemTime, NULL, writedLocalDate, 255);

GetTimeFormat(LOCALE\_USER\_DEFAULT, 0, &fileCreatedSystemTime, NULL, createdLocalTime, 255);

GetTimeFormat(LOCALE\_USER\_DEFAULT, 0, &fileAccessedSystemTime, NULL, accessedLocalTime, 255);

GetTimeFormat(LOCALE\_USER\_DEFAULT, 0, &fileWritedSystemTime, NULL, writedLocalTime, 255);

cout << "Время создания: " << fileCreatedSystemTime.wDay << "." << fileCreatedSystemTime.wMonth << "."

<< fileCreatedSystemTime.wYear << " " << fileCreatedSystemTime.wHour << ":"

<< fileCreatedSystemTime.wMinute << "\n";

cout << "Последнее обращение: " << fileAccessedSystemTime.wDay << "." << fileAccessedSystemTime.wMonth << "."

<< fileAccessedSystemTime.wYear << " " << fileAccessedSystemTime.wHour << ":"

<< fileAccessedSystemTime.wMinute << "\n";

cout << "Последнее изменение: " << fileWritedSystemTime.wDay << "." << fileWritedSystemTime.wMonth << "."

<< fileWritedSystemTime.wYear << " " << fileWritedSystemTime.wHour << ":"

<< fileWritedSystemTime.wMinute << "\n";

}

BY\_HANDLE\_FILE\_INFORMATION fileinfo;

if (GetFileInformationByHandle(hFile, &fileinfo)){

cout << "\nСерийный номер тома: " << fileinfo.dwVolumeSerialNumber << endl

<< "Количество ссылок: " << fileinfo.nNumberOfLinks << endl;

}

fclose(pfile);

}

void changeFileAttributes() {

char fileName[250];

cout << "Введите имя файла (латинскими буквами, без пробелов): ";

cin >> fileName;

DWORD attrs = GetFileAttributesA(fileName);

char answer;

cout<<"Сделать архивным? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_ARCHIVE;

else

attrs &= ~FILE\_ATTRIBUTE\_ARCHIVE;

cout<<"Сделать невидимым? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_HIDDEN;

else

attrs &= ~FILE\_ATTRIBUTE\_HIDDEN;

cout<<"Сделать обычным? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_NORMAL;

else

attrs &= ~FILE\_ATTRIBUTE\_NORMAL;

cout<<"Индексировать содержание? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED;

else

attrs &= ~FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED;

cout<<"Доступен без сети? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_OFFLINE;

else

attrs &= ~FILE\_ATTRIBUTE\_OFFLINE;

cout<<"Сделать доступным только для чтения? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_READONLY;

else

attrs &= ~FILE\_ATTRIBUTE\_READONLY;

cout<<"Сделать системным? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_SYSTEM;

else

attrs &= ~FILE\_ATTRIBUTE\_SYSTEM;

cout<<"Сделать временным? (y/n):";

cin >> answer;

if (answer == 'y')

attrs |= FILE\_ATTRIBUTE\_TEMPORARY;

else

attrs &= ~FILE\_ATTRIBUTE\_TEMPORARY;

if (SetFileAttributesA(fileName, attrs))

cout<<"Аттрибуты успешно установленны!\n";

else

cout<<"Произошла ошибка, аттрибуты не были установлены!\n";

}

void changeCreationTime() {

wchar\_t filename[250];

char fileCharName[250];

cout << "Введите имя файла: ";

cin >> fileCharName;

mbstowcs(filename, fileCharName, 250);

HANDLE hFile = CreateFile(filename, FILE\_WRITE\_ATTRIBUTES, 0, NULL, OPEN\_EXISTING, 0, NULL);

FILETIME fileTime;

SYSTEMTIME systemTimeNow;

GetSystemTime(&systemTimeNow);

SystemTimeToFileTime(&systemTimeNow, &fileTime);

if (SetFileTime(hFile, &fileTime, NULL, NULL))

cout << "Время успешно установленно\n"<<systemTimeNow.wDay<<"." << systemTimeNow.wMonth << "."

<< systemTimeNow.wYear << " " << systemTimeNow.wHour+3 << ":" << systemTimeNow.wMinute << "\n";

else

cout << "Произошла ошибка, время установить не удалось\n";

CloseHandle(hFile);

}

**AsyncCopy.cpp**

#include "AsyncCopy.h"

int streamCount;

int numbersOfByteToWrite;

int bufferMultiplier;

CHAR \*\*buffersArray;

OVERLAPPED \*overlapIn, \*overlapOut;

HANDLE original, copyFile;

LARGE\_INTEGER fileSize, endOfFile;

LONGLONG doneCount, recordCount;

void asyncCopyOfFile() {

int overlapOperationsCount = 0;

numbersOfByteToWrite = 4096;

bufferMultiplier = 1;

char sourceCharFile[250], destinationCharFile[250];

cout << "Введите путь к файлу (латинскими буквами, без пробелов): ";

cin >> sourceCharFile;

cout << "Введите путь куда скопировать файл (латинскими буквами, без пробелов): ";

cin >> destinationCharFile;

cout << "Количество перекрывающих операций: ";

cin >> overlapOperationsCount;

cout << "Множитель буфера: ";

cin >> bufferMultiplier;

numbersOfByteToWrite \*= bufferMultiplier;

cout << "Один блок = " << numbersOfByteToWrite << " б." << endl;

buffersArray = new CHAR\*[overlapOperationsCount];

for (int i = 0; i<overlapOperationsCount; i++)

buffersArray[i] = new CHAR[numbersOfByteToWrite];

overlapIn = new OVERLAPPED[overlapOperationsCount];

overlapOut = new OVERLAPPED[overlapOperationsCount];

LARGE\_INTEGER curPosIn;

wchar\_t source[250], destination[250];

mbstowcs(source, sourceCharFile, 250);

mbstowcs(destination, destinationCharFile, 250);

original = CreateFile(source, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED | FILE\_FLAG\_NO\_BUFFERING, NULL);

copyFile = CreateFile(destination, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED | FILE\_FLAG\_NO\_BUFFERING, NULL);

GetFileSizeEx(original, &fileSize);

cout << "Размер файла = " << fileSize.QuadPart << " б." << endl;

recordCount = fileSize.QuadPart / numbersOfByteToWrite;

if ((fileSize.QuadPart % numbersOfByteToWrite) != 0)

++recordCount;

cout << "Количество блоков = " << recordCount << endl;

DWORD startCopyTime, endCopyTime;

startCopyTime = GetTickCount();

curPosIn.QuadPart = 0;

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

overlapIn[i].hEvent = (HANDLE)i;

overlapOut[i].hEvent = (HANDLE)i;

overlapIn[i].Offset = curPosIn.LowPart;

overlapIn[i].OffsetHigh = curPosIn.HighPart;

if (curPosIn.QuadPart < fileSize.QuadPart)

ReadFileEx(original, buffersArray[i], numbersOfByteToWrite, &overlapIn[i], asyncRead);

curPosIn.QuadPart += (LONGLONG)numbersOfByteToWrite;

}

doneCount = 0;

while (doneCount < 2 \* recordCount)

SleepEx(INFINITE, true);

cout << "Копирование успешно завершено!" << endl;

delete[] overlapIn;

delete[] overlapOut;

for (int i = 0; i < overlapOperationsCount; ++i)

delete[]buffersArray[i];

delete[] buffersArray;

endOfFile.QuadPart = fileSize.QuadPart;

endOfFile.HighPart = fileSize.HighPart;

SetFilePointerEx(copyFile, endOfFile, 0, FILE\_BEGIN);

SetEndOfFile(copyFile);

CloseHandle(original);

CloseHandle(copyFile);

endCopyTime = GetTickCount();

cout << "Время потрачено: " << endCopyTime - startCopyTime << " мс" << endl;

}

VOID WINAPI asyncRead(DWORD Code, DWORD nBytes, LPOVERLAPPED lpOv) {

++doneCount;

LARGE\_INTEGER curPosIn, curPosOut;

DWORD i = (DWORD)(lpOv->hEvent);

curPosIn.LowPart = overlapIn[i].Offset;

curPosIn.HighPart = overlapIn[i].OffsetHigh;

curPosOut.QuadPart = curPosIn.QuadPart;

overlapOut[i].Offset = curPosOut.LowPart;

overlapOut[i].OffsetHigh = curPosOut.HighPart;

WriteFileEx(copyFile, buffersArray[i], numbersOfByteToWrite, &overlapOut[i], asyncWrite);

curPosIn.QuadPart += numbersOfByteToWrite \* (LONGLONG)(streamCount);

overlapIn[i].Offset = curPosIn.LowPart;

overlapIn[i].OffsetHigh = curPosIn.HighPart;

}

VOID WINAPI asyncWrite(DWORD Code, DWORD nBytes, LPOVERLAPPED lpOv){

++doneCount;

LARGE\_INTEGER curPosIn;

DWORD i = (DWORD)(lpOv->hEvent);

curPosIn.LowPart = overlapIn[i].Offset;

curPosIn.HighPart = overlapIn[i].OffsetHigh;

if (curPosIn.QuadPart < fileSize.QuadPart){

ReadFileEx(original, buffersArray[i], numbersOfByteToWrite, &overlapIn[i], asyncRead);

}

}