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:

```
GetLogicalDrives():
C
D
E
G

GetLogicalDriveStrings():
C:\
D:\
E:\
G:\
Press any key to continue . . .
```

2- Вывести информацию о диске:

```
Введите метку тома (например C): C
Тип диска: Внутренний диск
Имя диска:
Серийный номер: 1524111236
Тип файловой системы: NTFS
Системные флаги:
The specified volume supports preserved case of file names when it places a name
 on disk.
The specified volume supports case-sensitive file names.
The specified volume supports file-based compression.
The specified volume supports named streams.
The specified volume preserves and enforces access control lists (ACL). For exam
ple, the NTFS file system preserves and enforces ACLs, and the FAT file system d
oes not.
The specified volume supports the Encrypted File System (EFS).
The specified volume supports extended attributes.
The specified volume supports hard links.
The specified volume supports object identifiers.
The specified volume supports open by FileID.
The specified volume supports reparse points.
The specified volume supports sparse files.
The specified volume supports transactions.
The specified volume supports update sequence number (USN) journals.
The specified volume supports Unicode in file names as they appear on disk.
The specified volume supports disk quotas.
Дисковое пространство (свободное/всего): 84707 / 238122 MiB
Press any key to continue . . .
```

```
Введите метку тома (например C): D
Тип диска: Дисковод
Имя диска:
Серийный номер: 0
Тип файловой системы:
Системные флаги:
Дисковое пространство (свободное/всего): 0 / 0 MiB
Press any key to continue . . .
```

```
Введите метку тома (например C): G
Тип диска: Съёмный носитель
Имя диска: DOS
Серийный номер: 738535716
Тип файловой системы: FAT32
Системные флаги:
The specified volume supports preserved case of file names when it places a name
 on disk.
The specified volume supports Unicode in file names as they appear on disk.
Дисковое пространство (свободное/всего): 5155 / 7435 MiB
Press any key to continue . . .
```

3 - Создать каталог:

```
Введите название папки (латинскими буквами, без пробелов): C:/testFolder
Папка создана
Press any key to continue . . .
```

4 - Удалить каталог:

```
Введите название папки (латинскими буквами, без пробелов): C:/testFolder
Папка удалена
Press any key to continue . . .
```

5 - Создать файл:

```
Введите имя файла (латинскими буквами, без пробелов): C:/testFile
Файл создан!
Press any key to continue . . .
```

6 - Копировать файл:

```
Введите путь к файлу (латинскими буквами, без пробелов): C:/testFile
Введите путь куда скопировать файл (латинскими буквами, без пробелов): C:/testFi
leCopy
Файл скопирован!
Press any key to continue . . .
```

7 - Переместить файл:

```
Введите путь к файлу (латинскими буквами, без пробелов): C:/testFile
Введите путь куда переместить файл (латинскими буквами, без пробелов): C:/test/t
estFile
Файл перемещён!
Press any key to continue . . .
```

8 - Информация о файле:

```
Введите имя файла (латинскими буквами, без пробелов): C:/testFile
Аттрибуты: 0x20

FILE_ATTRIBUTE_ARCHIVE:
A file or directory that is an archive file or directory. Applications typically
 use this attribute to mark files for backup or removal.
--
Время создания: 25.9.2016 10:25
Последнее обращение: 25.9.2016 10:25
Последнее изменение: 25.9.2016 10:25

Серийный номер тома: 1524111236
Количество ссылок: 1
Press any key to continue . . .
```

9 - Изменить атрибуты файла:

Изменим атрибуты и снова вызовем функцию информация о файле

```
Введите имя файла (латинскими буквами, без пробелов): C:/testFile
Сделать архивным? (y/n):n
Сделать невидимым? (y/n):y
Сделать обычным? (y/n):n
Индексировать содержание? (y/n):y
Доступен без сети? (y/n):n
Сделать доступным только для чтения? (y/n):y
Сделать системным? (y/n):y
Сделать временным? (y/n):y
Аттрибуты успешно установленны!
Press any key to continue . . .
```

```
Введите имя файла (латинскими буквами, без пробелов): C:/testFile
Аттрибуты: 0x2107

FILE_ATTRIBUTE_HIDDEN:
The file or directory is hidden. It is not included in an ordinary directory lis
ting.
--
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED:
The file or directory is not to be indexed by the content indexing service.
FILE_ATTRIBUTE_READONLY:
A 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.
--
FILE_ATTRIBUTE_SYSTEM:
A file or directory that the operating system uses a part of, or uses exclusivel
y.
--
FILE_ATTRIBUTE_TEMPORARY:
A 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 th
e data. Otherwise, the data is written after the handle is closed.
--
Время создания: 25.9.2016 10:25
Последнее обращение: 25.9.2016 10:25
Последнее изменение: 25.9.2016 10:25

Серийный номер тома: 1524111236
Количество ссылок: 1
Press any key to continue . . .
```

10 - Изменить время создания файла:

```
Введите имя файла: C:/testFile
Время успешно установленно
25.9.2016 10:32
Press any key to continue . . .
```

```
Введите имя файла (латинскими буквами, без пробелов): C:/testFile
Аттрибуты: 0x2107

FILE_ATTRIBUTE_HIDDEN:
The file or directory is hidden. It is not included in an ordinary directory lis
ting.
--
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED:
The file or directory is not to be indexed by the content indexing service.
FILE_ATTRIBUTE_READONLY:
A 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.
--
FILE_ATTRIBUTE_SYSTEM:
A file or directory that the operating system uses a part of, or uses exclusivel
y.
--
FILE_ATTRIBUTE_TEMPORARY:
A 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 th
e data. Otherwise, the data is written after the handle is closed.
--
Время создания: 25.9.2016 10:32
Последнее обращение: 25.9.2016 10:25
Последнее изменение: 25.9.2016 10:25

Серийный номер тома: 1524111236
Количество ссылок: 1
Press any key to continue . . .
```

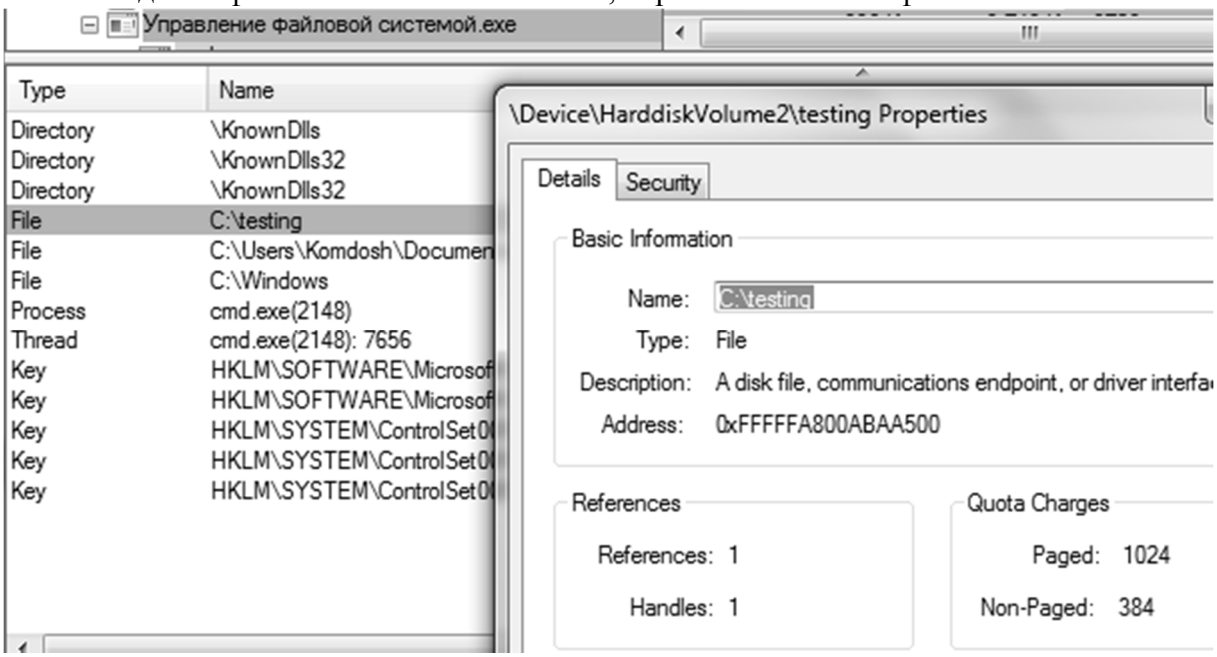
Определение процесса:

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

```
0: kd> !process 1A34 0
Searching for Process with Cid == 1a34
PROCESS fffffa800a556060
    SessionId: 1  Cid: 1a34    Peb: 7efdf000  ParentCid: 18dc
    DirBase: 0b7f4000  ObjectTable: fffff8a0085b9dd0  HandleCount:  13.
    Image: Oi?aaeaiea oaeeiaie nenoaiie.exe
```

При запуске программы, у неё имеется 13 объектов.

После создания файла появляется 14 объект, обработчик нашего файла



Информация о созданном файле:

```
0: kd> dt _object_header 0xFFFFFA800ABAA500
nt!_OBJECT_HEADER
   +0x000 PointerCount     : 0n14155781
   +0x008 HandleCount      : 0n-6047203996464
   +0x008 NextToFree       : 0xfffffa80`068dccd0 Void
   +0x010 Lock             : _EX_PUSH_LOCK
   +0x018 TypeIndex        : 0x40 '@'
   +0x019 TraceFlags       : 0x71 'q'
   +0x01a InfoMask         : 0x3b ';'
   +0x01b Flags            : 0x1b ''
   +0x020 ObjectCreateInfo : 0xfffff8a0`1b3b7330 _OBJECT_CREATE_INFORMATION
   +0x020 QuotaBlockCharged : 0xfffff8a0`1b3b7330 Void
   +0x028 SecurityDescriptor : 0xfffffa80`09634988 Void
   +0x030 Body             : _QUAD
0: kd> !object 0xFFFFFA800ABAA500
Object: fffffa800abaa500  Type: (fffffa8006107080) File
    ObjectHeader: fffffa800abaa4d0 (new version)
    HandleCount: 1  PointerCount: 1
    Directory Object: 00000000  Name: \testing {HarddiskVolume2}
```

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

```
Введите путь к файлу (латинскими буквами, без пробелов): C:/testing
Введите путь куда скопировать файл (латинскими буквами, без пробелов): C:/testin
gCopy
Произошла ошибка, файл не был скопирован!
Press any key to continue . . .
```

После закрытия обработчика:

```
Введите путь к файлу (латинскими буквами, без пробелов): C:/testing
Введите путь куда скопировать файл (латинскими буквами, без пробелов): C:/testin
gCopy
Файл скопирован!
Press any key to continue . . .
```

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

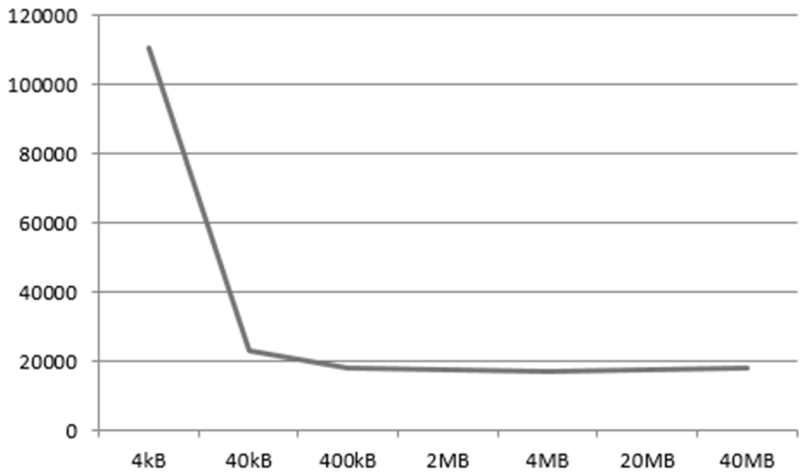
Вывод: Операционная система Windows позволяет разработчикам легко работать с файловой системой посредствам WinAPI.

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

11 - Асинхронное копирование файла

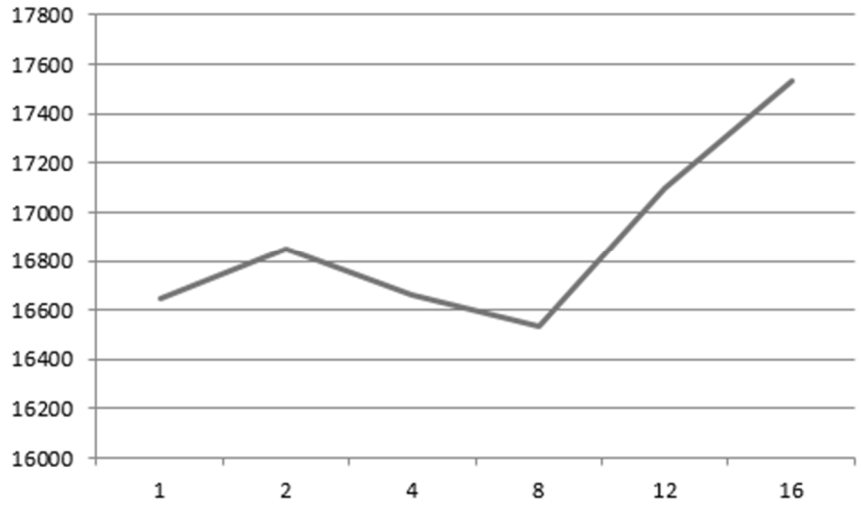
Эксперимент 1: SSD – размер файла 4765 Мб, размер кластера – 4 Кб, чтение и запись в пределах одного диска. Чем больше размер блока, тем меньше происходит операций переключения чтения\запись.

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.

```
0: kd> !process A8C 0
*** ERROR: Module load completed but symbols could not be loaded for LiveKdD.SYS

Searching for Process with Cid == a8c
PROCESS fffffa800bf79b10
    SessionId: 1  Cid: 0a8c    Peb: 7efdf000  ParentCid: 18dc
    DirBase: 51b80000  ObjectTable: fffff8a01b4c2710  HandleCount:  17.
    Image: Oi?aaeaiea oaeeiaie nenoaiie.exe
```

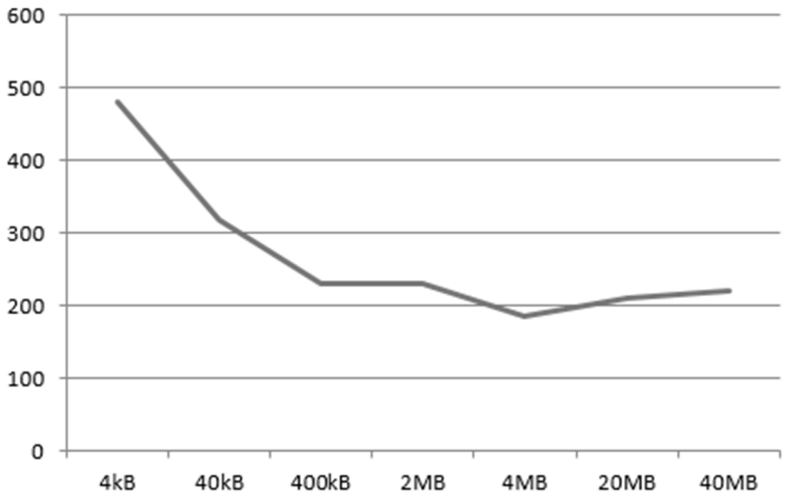
Как видим программа имеет 17 объектов после асинхронного копирования, хотя изначально она имеет только 13 объектов. Эти объекты, наши файлы (скопированный и оригинал)

```
0: kd> !handle 0034 3 a8c

Searching for Process with Cid == a8c
PROCESS fffffa800bf79b10
    SessionId: 1  Cid: 0a8c    Peb: 7efdf000  ParentCid: 18dc
    DirBase: 51b80000  ObjectTable: fffff8a01b4c2710  HandleCount:  17.
    Image: Oi?aaeaiea oaeeiaie nenoaiie.exe

Handle table at fffff8a01b4c2710 with 17 entries in use

0034: Object: fffffa800a6fc7c0  GrantedAccess: 00120089 Entry: fffff8a019f990d0
Object: fffffa800a6fc7c0  Type: (fffffa8006107080) File
    ObjectHeader: fffffa800a6fc790 (new version)
        HandleCount: 1  PointerCount: 1
        Directory Object: 00000000  Name: \Users\Komdosh\Documents\Visual Studio
 2015\Projects\╚яЁртыхэшх Ірщыютющ ёшёЄхьющ\╚яЁртыхэшх Ірщыютющ ёшёЄхьющ\fd {Har
ddiskVolume2}
```

```
0: kd> !handle 0038 3 a8c

Searching for Process with Cid == a8c
PROCESS fffffa800bf79b10
    SessionId: 1  Cid: 0a8c    Peb: 7efdf000  ParentCid: 18dc
    DirBase: 51b80000  ObjectTable: fffff8a01b4c2710  HandleCount:  17.
    Image: Oi?aaeaiea oaeeiaie nenoaiie.exe

Handle table at fffff8a01b4c2710 with 17 entries in use

0038: Object: fffffa800bf10070  GrantedAccess: 00120196 Entry: fffff8a019f990e0
Object: fffffa800bf10070  Type: (fffffa8006107080) File
    ObjectHeader: fffffa800bf10040 (new version)
        HandleCount: 1  PointerCount: 1
        Directory Object: 00000000  Name: \Users\Komdosh\Documents\Visual Studio
 2015\Projects\╚яЁртыхэшх Ігрщюютющ ёшёЄхьющ\╚яЁртыхэшх Ігрщюютющ ёшёЄхьющ\qd {Har
ddiskVolume2}
```

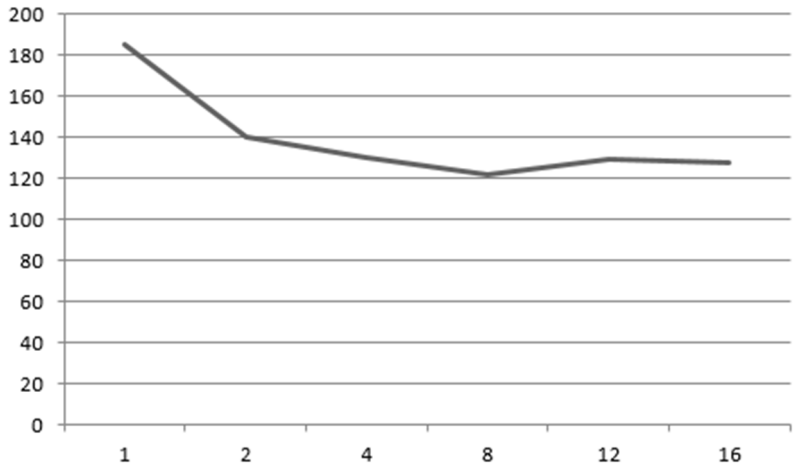
Эксперимент 2: HDD – размер файла 5139 Мб, размер кластера – 4 Кб, чтение и запись в пределах одного диска. Чем больше размер блока, тем меньше происходит операций переключения чтения\запись.

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);
	}
}
```