Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных средств

ИСПОЛЬЗОВАНИЕ СИСТЕМНЫХ ФУНКЦИЙ WINDOWS ДЛЯ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ

Методическое пособие
по курсу
«Системное программирование»
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

Минск БГУИР 2011

УДК 004.45(075.8)
ББК 32.973.26-018.2я73
И88

Р е ц е н з е н т:
доцент кафедры электронно-вычислительных машин учреждения образования
«Белорусский государственный университет
информатики и радиоэлектроники», кандидат технических наук
А. А. Петровский

Авторы:
Д. С. Лихачёв, М. З. Лившиц, А. Е. Новиков, И. С. Азаров, М. М. Родионов

И88 **Использование** системных функций Windows для работы с файловой системой : метод. пособие по курсу «Системное программирование» для студ. спец. 1-40 02 02 «Электронные вычислительные средства» дневн. формы обуч. / Д. С. Лихачёв [и др.]. – Минск : БГУИР, 2011. – 31 с. : ил.

ISBN 978-985-488-707-4.

Пособие содержит описание набора системных функций Windows для работы с файловой системой. Описаны примеры использования этих функций для обработки файловых данных, для работы с атрибутами файлов и получения дополнительной информации о файловой системе. Дано описание лабораторной работы.

Коллектив авторов выражает отдельную благодарность студенту группы 810901 Мазуркевичу Артёму Сергеевичу за помощь в оформлении методического пособия.

УДК 004.45(075.8)
ББК 32.973.26-018.2я73

ISBN 978-985-488-707-4

1. Понятия файла и файловой системы

Программист имеет дело с логической организацией файла, представляя файл в виде совокупности определенным образом организованных логических записей. **Логическая запись** – это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система позволяет программисту работать с отдельной логической записью.

Файл – это простой набор неструктурированных данных, которые имеют символьное имя. Правила расположения файла на устройстве внешней памяти, в частности на диске, описываются физической организацией файла. Файл состоит из физических записей – блоков. Блок – наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.

Файловая система – это часть операционной системы, обеспечивающая пользователю удобный интерфейс при работе с данными, хранящимися на диске, и совместное использование файлов несколькими пользователями и процессами.

Файловая система включает:

– совокупность всех файлов на диске;

– наборы структур данных, используемых для управления файлами, такие как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;

– комплекс системных программных средств, реализующих управление файлами: создание, удаление, чтение, запись, именование, поиск и другие операции.

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени.

Разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется составным именем, представляющим собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу может быть дано несколько разных имен. В этом случае операционная система присваивает файлу дополнительно уникальное имя так, чтобы можно было установить однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы.

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и тому подобное. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не

используют ASCII-коды, они, как правило, имеют сложную внутреннюю структуру, например объектный код программы или архивный файл.

Специальные файлы – это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять соответствующие операции, используя обычные команды записи в файл или чтения из файла. Такие команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления вводом-выводом. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Каталог – это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений, а с другой стороны – это файл, содержащий системную информацию о группе его составляющих файлов. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками – атрибутами.

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

– информация о разрешенном доступе;
– пароль для доступа к файлу;
– владелец файла;
– создатель файла;
– признак «только для чтения»;
– признак «скрытый файл»;
– признак «системный файл»;
– признак «архивный файл»;
– признак «двоичный/символьный»;
– признак «временный», что означает «удалить после завершения процесса»;
– признак блокировки;
– длина записи;
– указатель на ключевое поле в записи;
– длина ключа;
– время создания, последнего доступа и последнего изменения;
– текущий размер файла;
– максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в UNIX. Каталоги могут образовывать иерархическую структуру, когда каталог более низкого уровня может входить в каталог более высокого уровня.

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть – если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру, а в UNIX – сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным име-

нем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

Определить права доступа к файлу – значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

– создание файла;
– удаление файла;
– открытие файла;
– закрытие файла;
– чтение файла;
– запись в файл;
– дополнение файла;
– поиск в файле;
– получение атрибутов файла;
– установление новых значений атрибутов;
– переименование;
– выполнение файла;
– чтение каталога и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки – всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции. В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

2. Функции для работы с директориями

2.1. Определение текущей директории. Функция GetCurrentDirectory

Каждый процесс запускается из так называемой текущей директории. Все операции с файлами также выполняются в текущей директории. Для определения того, какая директория является в настоящее время текущей, предназначена функция *GetCurrentDirectory()*. Она имеет следующее описание:

```
DWORD GetCurrentDirectory(DWORD nBufferLength,
                          LPSTR LpBuffer);
```

Параметры функции:

nBufferLength – размер буфера для принимаемой строки;

LpBuffer – указатель на буфер, в который будет записано наименование текущей директории.

В случае удачного выполнения функции возвращается число байтов, скопированных в буфер, в случае неудачи – 0.

Для корректного задания размера буфера имеется два возможных способа:

1) вызвать функцию *GetCurrentDirectory* с нулевым размером буфера, получить требуемое количество байтов, динамически выделить буфер требуемого размера, после чего повторно вызвать *GetCurrentDirectory;*

2) перед вызовом функции *GetCurrentDirectory* вызвать функцию *GetVolumeInformation,* с её помощью определить значение максимальной длины компонента, после чего динамически выделить буфер такого размера, чтобы он заведомо вместил бы наименование текущей директории.

2.2. Установка текущей директории. Функция SetCurrentDirectory

Для смены текущей директории используется следующая функция:

```
BOOL SetCurrentDirectory (LPCSTR lpPathName);
```

Данная функция устанавливает необходимый текущий каталог. Для установки текущего каталога необходимо, чтобы сам каталог существовал, и, если он отсутствует, тогда предварительно его создать (например функцией *CreateDirectory*).

Параметр функции *LpPathName* – это строка, содержащая путь к каталогу.

В случае успешного завершения возвращается значение TRUE, в случае неудачи – FALSE. Получить дополнительную информацию об ошибке исполнения можно с помощью функции *GetLastError*.

2.3. Создание директории. Функция CreateDirectory

Для создания директории предназначена следующая функция:

```
BOOL CreateDirectory (LPCSTR LpPathName,
     LPSECURITY_ATTRIBUTES LpSecurityAttributes);
```

Первый аргумент этой функции *LpPathName* – это указатель на буфер, в котором записано наименование директории, которая создается. При создании каталога процесс может инициализировать структуру SECURITY_ATTRIBUTES и присвоить каталогу особые атрибуты, чтобы, например, другой пользователь не мог «войти» в созданный каталог или удалить его. Разграничение доступа возможно только при создании каталога в разделе NTFS. При успешном выполнении функция возвращает TRUE, в ином случае – FALSE.

2.4. Удаление директории. Функция Remove Directory

Для удаления директории используется следующая функция:

```
BOOL RemoveDirectory (LPCSTR LpPathName);
```

При успешном выполнении функция возвращает TRUE, в ином случае – FALSE. Значение FALSE может соответствовать случаю, когда пользователь не имеет права удаления данного каталога. Поэтому рекомендуется всегда использовать функцию *GetLastError* для идентификации возможных ошибок.

3. Работа с файлами

3.1. Создание и открытие файла. Функция CreateFile

```
HANDLE CreateFile (
     LPCSTR LpFileName,
     DWORD dwDesiredAccess,
     DWORD dwShareMode,
     LPSECURITY_ATTRIBUTES lpSecurityAttributes,
     DWORD dwCreationDisposition,
     DWORD dwFlagsAndAttributes,
     HANDLE hTemplateFile);
```

Эта функция не только создает и открывает дисковые файлы, но и открывает множество других устройств (COM-порт, LPT-порт и т. п.).

Параметр *LpFileName* представляет собой указатель на имя файла. К примеру, синтаксис описания различных устройств, распознаваемый функцией *CreateFile,* приведен в табл. 1.

Таблица 1

Синтаксис описания различных устройств

Устройство	Функция, применяемая для его открытия
Файл	CreateFile (LpFileName — полный или UNC-путь)
Каталог	CreateFile (LpFileName — имя или UNC-имя каталога при флаге FILE_FLAG_BACKUP_SEMANTICS)
Логический диск	CreateFile(LpFileName имеет вид «\\.\x:»)
Физический диск	CreateFile(LpFileName имеет вид «\\.\PHYSICALDRIVEx»)
Последовательный порт	CreateFile (LpFileName имеет вид «COM»)
Параллельный порт	CreateFile (LpFileName имеет вид «LPT»)

При открытии файла можно передать полный путь к нему, длина которого не должна превышать _MAX_PATH символов (260). Однако в Windows NT это ограничение можно преодолеть, обратившись к *CreateFileW* (Unicode-версии *CreateFile)* и указав перед путем такой префикс: «\\?\». Если использовать этот префикс, *CreateFileW* удалит его и разрешит передать путь, содержащий приблизительно до 32000 Unicode-символов. Но, указав упомянутый префикс, необходимо задать именно полный, а не относительный путь.

Параметр *dwDesiredAccess* определяет, какие действия можно произвести с содержимым файла, то есть определяет права на запись и на чтение данных. Он принимает одно из четырех значений, указанных в табл. 2.

Таблица 2

Значения параметра *dwDesiredAccess*

Флаг	Назначение
0	Содержание файла нельзя считывать в файл, нельзя производить запись. Флаг используется только тогда, когда необходимо всего лишь получить информацию о файле
GENERIC_READ	Разрешается чтение из файла
GENERIC_WRITE	Разрешается запись в файл
GENERIC_READ | GENERIC_WRITE	Разрешается чтение из файла и запись в файл

Параметр *dwShareMode* определяет права по совместному доступу разных процессов к файлу. В Windows типична ситуация, когда к какому-то устройству одновременно обращаются несколько компьютеров (в сетевой среде) или процессов (в многопоточной среде), а значит, необходимо решить, надо ли ограничивать доступ к устройству со стороны других компьютеров или процессов и, если надо, то – как. Допустимые значения параметра *dwShareMode* указаны в табл. 3.

Таблица 3

Значения параметра

Флаг	Назначение
0	Исключительный доступ к файлу, сторонние пользователи открыть файл не могут
FILE_SHARE_READ	Разрешено чтение содержимого файла сторонними пользователями. Если оно уже открыто для записи, вызов CreateFile даст ошибку. Кроме того, если устройство открыто для чтения, то последующий вызов CreateFile с требованием предоставить доступ для записи закончится неудачей
FILE_SHARE_WRITE	Разрешена запись в файл сторонними пользователями. Если устройство уже открыто на чтение, вызов CreateFile даст ошибку. Кроме того, если оно открыто для чтения, то последующий вызов CreateFile с требованием предоставить доступ для записи закончится неудачей
FILE_SHARE_READ|FILE_SHARE_WRITE	Разрешено чтение из файла и запись в файл. Если устройство уже открыто в монопольном режиме, вызов CreateFile даст ошибку. Повторный вызов CreateFile с требованием предоставить доступ для чтения или записи завершится успешно

Четвёртый параметр, *lpSecurityAttributes,* указывает на структуру SECURITY_ATTRIBUTES, которая позволяет задать атрибуты защиты для связанного с устройством объекта ядра и определить, будет ли возвращенный описатель наследуемым (т. е. получит ли дочерний процесс доступ к устройству). Если никакой особой защиты файлу не требуется, то в это поле можно занести NULL.

Параметр *dwCreationDisposition* определяет, какие действия необходимо произвести в тех случаях, когда файл уже существует либо когда файл ещё не существует. Это поле может принимать одно из значений, указанных в табл. 4.

Таблица 4

Принимаемые значения поля

Параметр	Назначение
CREATE_NEW	Создаётся новый файл. Если файл с указанным именем уже есть, функция завершает работу с ошибкой
CREATE_ALWAYS	Заставляет CreateFile создать файл независимо от того, имеется ли файл с таким именем или нет. Если файл с указанным именем уже существует, то он переписывается
OPEN_EXISTING	Заставляет CreateFile открыть существующий файл и, если он отсутствует, сообщить об ошибке
OPEN_ALWAYS	Заставляет CreateFile открыть существующий файл и, если он отсутствует, создать новый
TRUNCATE_EXISTING	Заставляет CreateFile открыть существующий файл, который сразу после открытия усекается до нулевой длины. Если такой файл отсутствует, функция выдаст ошибку. Этот флаг можно использовать только вместе с флагом GENERIC_WRITE

При открытии с помощью *CreateFile* устройство, отличное от файла, в параметре *dwCreationDistribution* необходимо передать OPEN_EXISTING.

Параметр *dwFlagsAndAttributes* служит двум целям: позволяет задавать флаги, тонко настраивающие взаимодействие с устройством и, если устройство является файлом, то устанавливать атрибуты.

Большая часть данных флагов сообщает системе как именно можно оптимизировать алгоритмы кеширования, обеспечивая более эффективное выполнение программы.

Флаг FILE_FLAG_NO_BUFFERING влияет на ввод/вывод в файловых системах. Чтобы повысить производительность, система кеширует данные, записываемые или считываемые с дисков. Обычно этот флаг не устанавливается, и диспетчер кеша сохраняет в памяти участки, к которым недавно обращались. Если необходимо считать несколько байтов из файла, а также считать новую порцию, то с большой долей вероятности эти данные уже находятся в памяти, и поэтому требуется лишь одно обращение к диску вместо двух.

Кроме буферизации, диспетчер кеша способен осуществлять опережающее чтение, благодаря чему данные, которые необходимо будет загрузить, уже будут присутствовать в памяти. Таким образом, скорость работы системы возрастает за счет того, что из файла считывается больше данных, чем нужно на этот момент. Однако если данные больше не будут считываться из этого файла, то получится, что память расходуется впустую.

Установив флаг FILE_FLAG_NO_BUFFERING, диспетчер кеша отказывается от буферизации каких-либо данных. В зависимости от поставленной цели, этот флаг может повысить скорость работы приложения и оптимизировать использование памяти. Драйвер устройства файловой системы записывает данные прямо в предоставляемый буфер, именно поэтому рекомендуется соблюдать следующие правила:

– всегда указывать при доступе к файлу смещения, кратные размеру сектора дискового тома;

– считывать и записывать данные блоками, кратными размеру сектора дискового тома;

– буфер в адресном пространстве процесса должен начинаться с адреса, значение которого делится на размер сектора без остатка.

Следующие два флага: FILE_FLAG_SEQUENTIAL_SCAN и FILE_FLAG_RANDOM_ACCESS имеют смысл, только если система буферизует данные. Когда установлен режим FILE_FLAG_NO_BUFFERING, оба эти флага игнорируются.

Если был задан флаг FILE_FLAG_SEQUENTIAL_SCAN, система считает, что идет обращение к файлу последовательно. Считывая порцию данных из файла, система на деле считывает информации больше, чем запрошено. Это сокращает число обращений к жесткому диску и повышает скорость работы программы. Если же данные считываются из файла в произвольном порядке, система затрачивает несколько больше времени и памяти на кеширование данных, к которым никто не обращался. Это нормально, но если часто выполняются подобные действия, лучше установить флаг FILE_FLAG_RANDOM_ACCESS. Он сообщает системе, что опережающее чтение данных не требуется.

Последний флаг FILE_FLAG_WRITE_THROUGH, связанный с управлением кешем, отключает промежуточное кэширование операций записи в файл, сокращая вероятность потери данных. При установке этого флага система сразу же записывает на диск все изменения, вносимые в файл. Однако система попрежнему поддерживает внутренний кеш для файловых данных, и в операциях чтения используются данные, помещенные в него. Если этот флаг указан при открытии файла на сетевом сервере, Win32-функции записи не возвращают управление вызвавшему их потоку, пока данные не будут записаны на диск сервера.

Чтобы система удалила файл после его закрытия, необходимо указать флаг FILE_FLAG_DELETE_ON_CLOSE. Обычно он применяется в сочетании с атрибутом FILE_ATTRIBUTE_TEMPORARY, тогда программа сможет создать временный файл, что-то записать в него, прочесть и, наконец, закрыть, а систе-

ма сама удалит временный файл, что является удобным решением. Если процесс закроет описатель файла, который пока еще открыт каким-то другим процессом, система удалит этот файл не сразу, а дождется закрытия всех его описателей.

Флаг FILE_FLAG_BACKUP_SEMANTICS используют в программах резервного копирования и восстановления. Прежде чем открыть или создать какой-либо файл, система обычно проверяет атрибуты его защиты и разрешает доступ к нему только процессу с соответствующими правами. Но резервное копирование трактуется особым образом, и в этом случае можно подавить некоторые виды проверки при доступе к файлу. Флаг FILE_FLAG_BACKUP_SEMANTICS заставляет систему контролировать права доступа у процесса и, если они достаточны, позволяет открыть файл исключительно для резервного копирования или восстановления. С помощью этого флага можно также открыть описатель каталога.

Флаг FILE_FLAG_POSIX_SEMANTICS сообщает, что при доступе к файлу следует применять правила POSIX. Файловые системы, используемые POSIX, чувствительны к регистру букв в именах файлов, и, например, LABRAB1.DOC, Labrab1.Doc и labRab1.doc считаются тремя разными файлами. В то же время MS-DOS, 16-разрядная Windows и Win32 к регистру букв в именах файлов не чувствительны. Поэтому необходимо быть крайне осторожным, используя FILE_FLAG_POSIX_SEMANTICS. Файл, при создании которого установлен этот флаг, может оказаться недоступным из приложений MS-DOS, 16-разрядной Windows и Win32.

Последний флаг, FILE_FLAG_OVERLAPPED, указывает на то, что необходимо получить асинхронный доступ к файлу. По умолчанию (в отсутствие флага FILE_FLAG_OVERLAPPED) устройства открываются для синхронного ввода/вывода. Когда читаются данные из файла, программа приостанавливается до завершения этой операции. Потом она вновь получит управление и продолжит работу.

Так как ввод/вывод на устройствах – операция медленная (по сравнению с большинством других), то может понадобиться асинхронная связь. В общих чертах это выглядит так: вызывается функция чтения или записи, и, не дожидаясь завершения операции ввода/вывода, она тут же возвращает управление, т. е. выполнение запрошенной операции система берет на себя, используя собственные потоки. Закончив, операционная система уведомляет об этом программу. Асинхронный ввод/вывод особенно полезен для серверных приложений.

Флаги файловых атрибутов полностью игнорируются во всех случаях, кроме одного, – когда создаётся новый файл и передается в функцию *CreateFile* вместо параметра *hTemplateFile* значение NULL. Значения атрибутов перечислены в табл. 5.

Таблица 5

Описание атрибутов

Идентификатор	Описание
FILE_ATTRIBUTE_ARCHIVE	Архивный файл (допускает удаление и резервное копирование). Когда *CreateFile* создает файл, этот атрибут устанавливается автоматически
FILE_ATTRIBUTE_HIDDEN	Скрытый файл (не включается в обычный список файлов каталога)
FILE_ATTRIBUTE_NORMAL	Файл, у которого не установлены другие атрибуты. Этот атрибут допустим, если не используются другие атрибуты
FILE_ATTRIBUTE_READONLY	Файл только для чтения. Приложения могут читать файл, но не имеют права записывать в него или удалять его
FILE_ATTRIBUTE_SYSTEM	Файл — часть операционной системы или используется исключительно операционной системой
FILE_ATTRIBUTE_COMPRESSED	Файл или каталог сжаты. Если это файл, все данные в нем сжаты. Для каталога данный атрибут означает, что новые файлы или подкаталоги по умолчанию создаются как сжатые
FILE_ATTRIBUTE_OFFLINE	Файл существует, но данные перемещены в хранилище, недоступное в оперативном режиме. Этот флаг применяется в системах с иерархической организацией физической памяти
FILE_ATTRIBUTE_TEMPORARY	Файл предполагается использовать непродолжительное время. Файловая система стремится поместить его в оперативную память, чтобы максимально сократить время доступа к нему

Флаг FILE_ATTRIBUTE_TEMPORARY применяется при создании временных файлов. Создавая файл с этим атрибутом, *CreateFile* пытается разместить его данные не на диске, а в оперативной памяти, что ускоряет доступ. Если запись в файл продолжится, то в какой-то момент система уже не сможет держать его целиком в памяти и начнет сбрасывать его данные на жесткий диск. Таким образом, можно увеличить производительность системы, скомбинировав флаги FILE_ATTRIBUTE_TEMPORARY и FILE_FLAG_DELETE_ON_CLOSE. Обычно при закрытии файла система сбрасывает на диск данные из кеша, но не делает этого, если файл надо удалить после его закрытия.

Параметр *hTemplateFile* содержит или описатель открытого файла, или NULL. Если в *hTemplateFile* указан описатель файла, функция *CreateFile* игнорирует флаги атрибутов в параметре *dwFlagsAndAttributes* и использует атрибуты того файла, на который указывает параметр *hTemplateFile.* Чтобы такая схема сработала, шаблонный файл надо предварительно открыть с флагом

GENERIC_READ. Если *CreateFile* не создает новый файл, а открывает существующий, параметр *hTemplateFile* игнорируется.

Если *CreateFile* успешно создаст или откроет устройство или файл, она вернет описатель этого устройства или файла, а при ошибке – код INVALID_HANDLE_VALUE.

Большинство Win32-функций при неудаче возвращают NULL, но *CreateFile* в таких случаях возвращает INVALID_HANDLE_VALUE (определенный как 0xFFFFFFFF).

Пример. Необходимо создать новый файл в рабочей директории с именем File_New.txt, с доступом по чтению и записи для разных процессов, с обычными атрибутами.

```
HANDLE myFile=CreateFile(
     "File_New.txt",        // имя файла

     GENERIC_READ | GENERIC_WRITE,
              // параметры доступа по чтению и записи

     FILE_SHARE_READ | FILE_SHARE_WRITE,
              //параметры доступа для сторонних процессов
     NULL,
     CREATE_NEW,            //создание нового файла
     FILE_ATTRIBUTE_NORMAL, //атрибуты по умолчанию
     NULL);
```

Проверка описателя файла выполняется следующим образом:

```
if(myFile == INVALID_HANDLE_VALUE){
// файл не создан
}
else{
// файл успешно создан
}
```

3.2. Понятие указателя файла. Функция для перемещения указателя файла SetFilePointer

С каждым файлом связана специальная внутренняя системная переменная, которая указывает на то место в файле, с которого будут начинаться операции ввода-вывода. Эта переменная называется указателем файла. При открытии файла этот указатель обычно устанавливается перед самым первым симво-

лом файла. Для установки указателя файла используется функция *SetFilePointer*:

```
DWORD SetFilePointer(
     HANDLE hFile,
     LONG lDistanceToMove,
     PLONG lpDistanceToMoveHigh,
     DWORD dwMoveMethod);
```

Первый аргумент этой функции, *hFile* – описатель рабочего файла, в котором перемещается указатель. Второй аргумент, *lDistanceToMove*, является числом символов, на которое необходимо передвинуть указатель файла. Для установки нового значения указателя файла это число складывается со старым значением указателя файла. Если аргумент *lDistanceToMove* больше нуля, то указатель файла перемещается вперёд, то есть к концу файла. Отрицательное значение означает, что указатель файла перемещается назад, то есть к началу файла. Тип этого аргумента – LONG, то есть, используя только этот аргумент, невозможно переместить указатель файла больше, чем на 2^{32} байта. Для решения этой проблемы используется третий аргумент, через который передается указатель на буфер, содержащий старшие тридцать два разряда требуемого значения. При определении нового значения указателя файла используется следующая формула:

```
НЗУ = СЗУ + lDistanceToMove + *lpDistanceToMoveHigh,
```

где НЗУ – новое значение указателя;
СЗУ – старое значение указателя.

Тип возвращаемого функцией значения – двойное слово. В этом двойном слове функция возвращает новое значение указателя файла. В том случае, когда тридцати двух разрядов для этого мало, старшие тридцать два разряда записываются по адресу *lpDistanceToMoveHigh*.

Последний аргумент, *dwMoveMethod*, определяет точку отсчёта для перемещения указателя файла. Возможные значения этого аргумента приведены в табл. 6.

Таблица 6

Возможные значения аргумента

Метод	Значение	Назначение
FILE_BEGIN	0	Старое значение указателя игнорируется и считается равным нулю, СЗУ в данном случае равно нулю
FILE_CURRENT	1	Для вычислений используется текущее значение указателя
FILE_END	2	Старое значение указателя принимается равным числу байтов в файле, то есть фактически указатель устанавливается в конец файла

В случае успешного выполнения функция возвращает новое значение указателя. Если же функция возвращает значение 0xFFFFFFFF, и при этом слово, на которое указывает третий аргумент, равно нулю, это будет означать, что при работе функции произошла ошибка. Или, если функция возвращает значение 0xFFFFFFFF и при этом слово, на которое указывает третий аргумент, не равно нулю, то для того чтобы определить, произошла ли ошибка, необходимо вызвать функцию *GetLastError*. Если эта функция вернёт значение, отличное от NO_ERROR, то это означает, что произошла ошибка.

3.3. Копирование файла. Функция CopyFile

Для копирования файла используется следующая функция:

```
BOOL CopyFile (
        LPCTSTR LpExistingFileName,
        LPCSTR LpNewFileName,
        BOOL bFailIfExists);
```

Функция *CopyFile* копирует файл, указанный в параметре *LpExistingFileName,* в новый файл, полное имя которого задано в параметре *LpNewFileName.* Параметр *bFailIfExists* указывает, должна ли функция сообщить об ошибке, если на диске уже существует файл с именем, указанным в *LpNewFileName.* Если файл с таким именем есть и параметр *bFaillfExists* равен TRUE, функция отказывается от копирования. В ином случае она удаляет существующий файл и создает новый. При благополучном завершении *CopyFile* возвращает TRUE. Копировать можно либо неоткрытые файлы, либо открытые только для чтения. Функция сообщает об ошибке, если существующий файл открыт каким-либо процессом для записи.

3.4. Удаление файла. Функция DeleteFile

Удаление файла производится с помощью следующей фнкции:

```
BOOL DeleteFile (LPCSTR LpFileName);
```

Удалив файл, заданный в параметре *LpFileName,* функция возвращает TRUE при благополучном завершении или сообщает об ошибке, если файл не существует или открыт каким-то процессом.

3.5. Запись информации в файл. Функция WriteFile

Запись информации в файл производится с помощью следующей функции:

```
BOOL WriteFile(
     HANDLE hFile,
     LPCVOID LpBuffer,
     DWORD nNumberOfBytesToWrite,
```

```
    LPDWORD LpNumberOfBytesWritten,
    LPOVERLAPPED LpOverlapped);
```

Параметры функции:

hFile – описатель файла;

LpBuffer – указатель на буфер, данные из которого нужно записать;

nNumberOfBytesToWrite – число байтов, которые записываются в файл;

LpNumberOfBytesWritten – указатель на буфер, в который после выполнения функции будет записано число байтов, записанных в файл;

LpOverlapped – указатель на структуру типа OVERLAPPED, которая используется в случае асинхронного ввода-вывода.

В случае успешного выполнения функция возвращает TRUE, в случае неудачи – FALSE.

Пример. Необходимо в файл, созданный ранее, записать фразу «Системное программирование».

```
char Buffer[]="Системное программирование"; // строка для
                                            // записи
DWORD dwBytes; // в эту переменную функция вернёт
               // кол-во байтов, записанных в файл
WriteFile(
    myFile, // описатель файла, в который производится запись
    Buffer,
    sizeof(Buffer), // размер записанной строки
    &dwBytes,
    NULL);
```

3.6. Чтение информации из файла. Функция ReadFile

Чтение информации из файла производится с помощью следующей функции:

```
BOOL ReadFile(
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToRead,
    LPDWORD lpNumberOfBytesRead,
    LPOVERLAPPED lpOverlapped );
```

Для того чтобы эта функция работала корректно, файл, из которого производится чтение, должен быть открыт с флагом GENERIC_READ.

Первый аргумент этой функции, *hFile*, является описатель того файла, из которого производится чтение данных.

Второй аргумент, *lpBuffer*, указывает на буфер, в который будет производиться чтение данных.

Третий аргумент, *nNumberOfBytesToRead*, определяет число байтов, которые необходимо прочесть из файла. В буфер, на который указывает *lpNumberOfBytesRead*, будет записано число реально прочитанных байтов.

Последний аргумент, *lpOverlapped*, является указателем на структуру типа OVERLAPPED. При синхронном вводе необходимо передать функции через этот аргумент значение NULL.

Если функция завершается нормально, она возвращает значение TRUE.

3.7. Установка метки конца файла. Функция SetEndOfFile

Обычно при закрытии файла система сама устанавливает конец файла. Но иногда возникает необходимость увеличить или уменьшить размер файла. Для этих целей в Windows существует функция *SetEndOfFile*:

```
BOOL SetEndOfFile(HANDLE hFile);
```

Единственным аргументом этого файла является описатель рабочего файла. Конец файла устанавливается в том месте, на котором находится указатель файла. После того, как будет установлен конец файла, необходимо закрыть файл.

3.8. Закрытие файла. Функция CloseHandle

Специальной функции для закрытия открытого ранее файла не существует, для этих целей используется функция для уничтожения описателя объекта ядра операционной системы *CloseHandle*:

```
BOOL CloseHandle (HANDLE hObject)
```

В данной функции параметр *hObject* – описатель файла.

3.9. Поиск файлов по шаблону

Для поиска файлов по заданному шаблону используются нижеперечисленные функции.

```
HANDLE FindFirstFile(
    LPCTSTR LpFileName,
    LPWIN32_FIND_DATA LpFindFileData);
```

Функция **FindFirstFile** сообщает системе, что необходимо искать именно файл. Параметр *LpFileName* указывает на строку (с нулевым символом в конце), содержащую имя файла. В имя можно включать символы подстановки (* и

?) и начальный путь. Параметр *LpFindFileData* – адрес структуры LPWIN32_FIND_DATA:

```
    typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD dwReserved0;
    DWORD dwReserved1;
    CHAR cFileName[MAX_PATH];
    CHAR cAlternateFileName[14];
} WIN32_FIND_DATA, * PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

Обнаружив файл, соответствующий заданной спецификации и расположенный в заданном каталоге, функция заполняет элементы структуры и возвращает описатель, а при отсутствии такого файла – возвращает код INVALID_HANDLE_VALUE и не изменяет содержимое структуры.

Функция *FindFirstFile* в случае неудачи возвращает INVALID_HANDLE_VALUE, а не NULL.

Структура WIN32_FIND_DATA содержит информацию о найденном файле: атрибуты, метки времени и размер.

Элемент *cFileName* содержит истинное имя файла. Его обычно и используют. В элемент *cAlternateFileName* помещается так называемый псевдоним файла. В этот элемент записывается преобразованное, или альтернативное имя (псевдоним) в соответствии со стандартом «8.3». Для файлов с краткими именами содержимое элементов *cFileName* и *cAlternateFileName,* конечно, совпадает. Для длинных – в элемент *cFileName* записывается настоящее имя, а в элемент *cAlternateFileName* — псевдоним. Например, система может преобразовать длинное имя файла «LabRabota» в псевдоним «LABRAB~1».

Если *FindFirstFile* нашла подходящий файл для поиска следующего файла, удовлетворяющего переданной при вызове *FindFirstFile* спецификации, необходимо вызвать функцию *FindNextFile:*

```
BOOL FindNextFile (
    HANDLE hFindFile,
    LPWIN32_FIND_DATA LpFindFileData);
```

Первый параметр этой функции – описатель, полученный предыдущим вызовом *FindFirstFile.* Второй – вновь указывает на структуру WIN32_FIND_DATA. Это может быть и другой экземпляр структуры, а не тот, который был указан при вызове *FindFirstFile.*

При успешном выполнении *FindNextFile* возвращает TRUE и заполняет структуру WIN32_FIND_DATA, в ином случае она возвращает FALSE.

Закончив поиск, описатель, возвращенный *FindFirstFile* или *FindFirstFileEx,* закрывается следующим образом:

```
BOOL FindClose (HANDLE hFindFile);
```

Это один из тех (весьма редких в Win32) случаев, когда для закрытия описателя *CloseHandle* не применяется. Вместо этой функции вызывается *FindClose* для удаления служебной информации, которую использовала система.

FindFirstFile и *FindNextFile* позволяют просматривать только те файлы (и подкаталоги), что находятся в одном, указанном каталоге. Чтобы «пройти» по всей иерархии каталогов, необходимо вызывать эти функции рекурсивно.

3.10. Работа с атрибутами и свойствами файла

Для того чтобы узнать атрибуты файла, существует функция *GetFileAttributes*:

```
DWORD GetFileAttributes(LPCWSTR lpFileName);
```

Единственным аргументом этой функции является имя файла, атрибуты которого необходимо получить. Возвращает функция двойное слово, биты которого соответствуют атрибутам файла.

При необходимости изменить какие-либо атрибуты файла существует другая функция, *SetFileAttributes()*:

```
BOOL SetFileAttributes(LPCWSTR lpFileName,
                       DWORD dwFileAttributes);
```

Первый аргумент этой функции *lpFileName* – имя файла, атрибуты которого необходимо установить. Второй аргумент *dwFileAttributes* – это двойное слово, показывающее, какие атрибуты файла должны быть установлены. В случае успешного завершения функция возвращает значение TRUE.

Если необходимо узнать размер файла, то используется функция *GetFileSize:*

```
DWORD GetFileSize(
    HANDLE hFile,
    LPDWORD lpFileSizeHigh);
```

Первый аргумент этой функции, *hFile*, содержит описатель файла, размер которого необходимо получить. При помощи этой функции возможно получить размер только открытого файла.

Возвращаемое функцией значение типа DWORD содержит младшие тридцать два разряда размера файла. Если нужно получить старшие тридцать два разряда размера файла, то функция поместит их в двойное слово, указатель на которое передан функции в качестве второго аргумента *lpFileSizeHigh*. Если старшие тридцать два разряда размера файла не нужны, то в качестве *lpFileSizeHigh* можно указать NULL.

3.11. Метки времени файла

С файлом связаны три метки времени. Первая метка – это время создания файла, вторая – время последнего обращения к файлу и третья – это время последней записи в файл. Для того чтобы получить значения этих меток, необходимо использовать функцию *GetFileTime:*

```
BOOL GetFileTime(
     HANDLE hFile,
     LPFILETIME lpCreationTime,
     LPFILETIME lpLastAccessTime,
     LPFILETIME lpLastWriteTime);
```

В качестве первого аргумента функций *GetFileTime*, *hFile* должен быть передан описатель открытого файла. Второй, третий и четвёртый аргументы представляют собой указатели на структуры типа FILETIME. По этим адресам будут записаны соответственно время создания файла, время последнего обращения к файлу и время последней записи в файл. Если нет необходимости в использовании какой-то из этих меток, то в качестве указателя на соответствующую структуру нужно передать NULL.

Структура FILETIME описана следующим образом:

```
typedef struct_FILETIME {
     DWORD dwLowDateTime;
     DWORD dwHighDateTime;
     } FILETIME, *PFILETIME *LPFILETIME;
```

В первое поле этой структуры, *dwLowDateTime*, функция записывает младшие тридцать два разряда метки времени, а во второе, *dwHighDateTime*, – старшие тридцать два разряда.

Для получения времени в виде системного времени используется функция *FileTimeToSystemTime:*

```
BOOL FileTimeToSystemTime(
     CONST FILETIME *lpFileTime
     LPSYSTEMTIME lpSystemTime);
```

Аргумент *lpFileTime* – это указатель на структуру типа FILETIME. Второй аргумент, *lpSystemTime*, – это указатель на структуру типа SYSTEMTIME. Формат этой структуры описан в файле winbase.h следующим образом:

```
typedef struct _SYSTEMTIME {
     WORD wYear;
     WORD wMonth;
     WORD wDayOfWeek;
     WORD wDay;
     WORD wHour;
     WORD wMinute;
     WORD wSecond;
     WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME *LPSYSTEMTIME;
```

При успешном выполнении функция возвращает значение TRUE.

Обратная функция по отношению к FileTimeToSystemTime – SystemTimeToFileTime:

```
BOOL SystemTimeToFileTime(
     CONST SYSTEMTIME *lpSystemTime,
     LPFILETIME lpFileTime);
```

При возникновении необходимости изменить метки времени, связанные с файлом, используется функция *SetFileTime:*

```
BOOL SetFileTime(
     HANDLE hFile,
     CONST FILETIME *lpCreationTime,
     CONST FILETIME *lpLastAccessTime,
     CONST FILETIME *lpLastWriteTime);
```

Устанавливать метки времени можно только у открытого файла. Остальные три аргумента – это указатели на структуры типа FILETIME, в которых записаны времена создания файла, последнего доступа к нему и последней записи в файл. Если не нужно менять какую-то из меток времени, то нужно вместо соответствующего указателя передать значение NULL.

При успешном выполнении функция возвращает значение TRUE.

3.12. Взятие информации о файле. Функция GetFileInformationByHandle

Для взятия информации о файле по его описателю служит функция *GetFileInformationByHandle*:

```
BOOL GetFileInformationByHandle(
```

```
    HANDLE hFile,
    LPBY_HANDLE_FILE_INFORMATION lpFileInformation);
```

Первый аргумент – описатель файла, информацию о котором нужно получить.

Второй аргумент показывает, какого рода информация может быть получена. Он является указателем на заполняемую этой функцией структуру типа BY_HANDLE_FILE_INFORMATION:

```
typedef struct _BY_HANDLE_FILE_INFORMATION {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD dwVolumeSerialNumber;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD nNumberOfLinks;
    DWORD nFileIndexHigh:
    DWORD nFileIndexLow;
} BY_HANDLE_FILE_INFORMATION,
*PBY_HANDLE_FILE_ INFORMATION,
*LPBY_HANDLE_FILE_INFORMATION;
```

Краткое описание полей в том порядке, в каком они следуют в структуре:
– атрибуты файла;
– время создания файла;
– время последнего доступа к файлу;
– время последней записи в файл;
– серийный номер тома, на котором находится файл;
– старшие тридцать два разряда размера файла;
– младшие тридцать два разряда размера файла;
– число ссылок на файл;
– старшие тридцать два разряда идентификатора файла;
– младшие тридцать два разряда идентификатора файла.

При открытии файла система присваивает файлу уникальный идентификатор. Значение именно этого идентификатора и записывается в последние два поля структуры типа BY_HANDLE_FILE_INFORMATION.

4. Файлы, отображаемые в память

Выше был описан стандартный способ работы с файлами. Он подразумевает открытие файла, чтение части файла в буфер, создаваемый в оперативной памяти, и запись содержимого буфера обратно в файл. При использовании отображаемых в память файлов возможно, практически не занимая оперативную

память, обращаться к файлу таким образом, словно файл полностью загружен в память. Таким образом, можно обойтись без операций файлового ввода-вывода и буферизации.

Для того чтобы отобразить файл в память, нужно выполнить следующие операции:

– создать или открыть объект ядра «файл», идентифицирующий дисковый файл, который необходимо использовать как проецируемый в память;

– создать объект ядра «проекция файла», чтобы сообщить системе размер файла и способ доступа к нему;

– указать системе, как спроецировать в адресное пространство процесса объект «проекция файла» – целиком или частично.

Закончив работу с проецируемым в память файлом, необходимо выполнить следующие действия:

– отменить проецирование файла в память, то есть закрыть объект «проекция файла»;

– закрыть проецируемый файл.

Этап 1: создание объекта «файл»

На этом этапе создаётся или открывается файл с помощью функции *CreateFile*. Таким образом, операционной системе указывается, где находится физическая память для проекции файла: на жестком диске, в сети, на CD-ROM или в другом месте.

Этап 2: создание объекта «проецируемый файл»

На этом этапе необходимо сообщить системе, какой объем физической памяти нужен проекции файла. Для этого вызывается функция *CreateFileMapping:*

```
HANDLE CreateFileMapping (
     HANDLE hFile,
     LPSECURITY_ATTRIBUTES LpFileMappingAttributes,
     DWORD flProtect,
     DWORD dwMaximumSizeHigh,
     DWORD dwMaximumSizeLow,
     LPCSTR LpName);
```

Первый параметр, *hFile,* идентифицирует описатель файла, проецируемого на адресное пространство процесса. Этот описатель получается после вызова *CreateFile.*

Параметр *LpFileMappingAttributes* – указатель на структуру SECURITY_ATTRIBUTES, которая относится к объекту ядра «проекция файла», для установки защиты по умолчанию ему присваивается NULL.

В *flProtect* надо указать желательные атрибуты защиты. Обычно используется один из перечисленных в табл. 7 атрибутов.

Таблица 7

Описание атрибутов защиты

Атрибут	Назначение
PAGE_READONLY	Отобразив объект «проекция файла» на адресное пространство, можно считывать данные из файла. При этом необходимо передать в CreateFile флаг GENERIC_READ
PAGE_READWRITE	Отобразив объект «проекция файла» на адресное пространство, можно считывать данные из файла и записывать их. При этом необходимо передать в CreateFile комбинацию флагов GENERIC_READ | GENERIC_WRITE
PAGE_WRITECOPY	Отобразив объект «проекция файла» на адресное пространство, можно считывать данные из файла и записывать их. Запись приведет к созданию закрытой копии страницы. При этом необходимо передать в CreateFile либо GENERIC_READ, либо GENERIC_READ|GENERIC_ WRITE

Основное назначение *CreateFileMapping* – гарантировать, что объекту «проекция файла» доступен нужный объем физической памяти. Через параметры *dwMaximumSizeHigh* и *dwMaximumSizeLow* системе сообщается максимальный размер файла в байтах. Так как Windows позволяет работать с файлами, размеры которых выражаются 64-разрядными числами, в параметре *dwMaximumSizeHigh* указываются старшие 32 бита, а в *dwMaximumSizeLow* — младшие 32 бита этого значения.

Последний параметр функции *LpName* – строка с нулевым байтом в конце; в ней указывается имя объекта «проекция файла», которое используется для доступа к данному объекту из другого процесса.

При удачном завершении этой функции система возвращает описатель объекта «проекция файла». Если объект создать не удалось, возвращается нулевой описатель NULL.

Пример. Создание проекции файла с атрибутом защиты на чтение и запись:

```
DWORD fSize, fhSize;   // младшие и старшие двойные слова
                       // размера файла
HANDLE hFile, hMapFile;//описатель файла и проекции файла
…
hFile = CreateFile(…); //создаем файл
…
fSize = GetFileSize(hFile, &fhSize); //высчитываем размер
                                     //файла
hMapFile = CreateFileMapping(
              hFile,
              NULL,
              PAGE_READWRITE,    // из проекции файла можно
                                 // читать данные и записывать
                                 // новые
```

```
        fhSize,
        fSize,
        NULL);
```

Этап 3: проецирование файловых данных на адресное пространство процесса

Когда объект «проекция файла» создан, нужно, чтобы система, зарезервировав регион адресного пространства под данные файла, передала их как физическую память, отображенную на регион. Это делает функция *MapViewOfFile*:

```
LPVOID MapViewOfFile(
     HANDLE hFileMappingObject,
     DWORD dwDesiredAccess,
     DWORD dwFileOffsetHigh,
     DWORD dwFileOffsetLow,
     DWORD dwNumberOfBytesToMap);
```

Параметр *hFileMappingObject* идентифицирует описатель объекта «проекция файла», возвращаемый предшествующим вызовом либо *CreateFileMapping*.

Параметр *dwDesiredAccess* идентифицирует вид доступа к данным. Можно задать одно из четырех значений, приведенных в табл. 8.

Таблица 8

Описание значений параметра *dwDesiredAccess*

Значение	Назначение
FILE_MAP_WRITE	Файловые данные можно считывать и записывать. Объект «проецируемый файл» должен быть создан с атрибутом PAGE_READWRITE
FILE_MAP_READ	Файловые данные доступны только для чтения. Объект «проецируемый файл» должен быть создан с атрибутом PAGE_READONLY, PAGE_READWRITE или PAGE _WRITECOPY
FILE_MAP_ALL_ACCESS	То же, что и FILE_MAP_WRITE
FILE_MAP_COPY	Файловые данные можно считывать и записывать, но запись приводит к созданию закрытой копии страницы. Объект «проецируемый файл» должен быть создан с атрибутом PAGE_READONLY, PAGE_READWRITE или PAGE _WRITECOPY

Остальные три параметра относятся к резервированию региона адресного пространства и к отображению на него физической памяти. При этом не обязательно проецировать на адресное пространство весь файл сразу. Можно спроецировать лишь малую его часть. Проецируя на адресное пространство процесса представление файла, нужно сделать следующее. Во-первых, сообщить системе, какой байт файла данных считать в представлении первым. Для этого пред-

назначены параметры *dwFileOffsetHigh* и *dwFileOffsetLow.* Размер проецируемого блока нужно указать в параметре *dwNumberOfBytesToMap.*

Если функция завершилась с ошибкой, то возвращается NULL.

Пример. Резервируется регион адресного пространства под первый байт файла и передаётся как физическая память с доступом по чтению и записи.

```
HANDLE hMapFile;
LPVOID LpMapFile;
LpMapFile = MapViewOfFile(
    hMapFile,                          //описатель проекции файла
    FILE_MAP_READ|FILE_MAP_WRITE,  //файловые данные
                                  //можно читать и записывать
    0,                            //смещение в файле равно 0
    0,
    1);                           //резервируется один байт
```

Этап 4: отмена отображения файла на адресное пространство процесса

Когда необходимость в данных файла (спроецированного на регион адресного пространства процесса) отпадёт, необходимо освободить регион вызовом:

```
BOOL UnmapViewOfFile (LPCVOID LpBaseAddress);
```

Единственный параметр этой функции, *LpBaseAddress,* указывает базовый адрес возвращаемого системе региона.

Для повышения производительности при работе с проекцией файла система буферизует страницы данных в файле и не обновляет образ файла. При необходимости можно заставить систему записать измененные данные (все или частично) в образ файла, вызвав функцию *FlushViewOfFile.*

В случае успеха функция возвращает TRUE, в случае неудачи – FALSE.

```
BOOL FlushViewOfFile(
    LPCVOID lpAddress,
    SIZE_T dwNumberOfBytesToFlush);
```

Ее первый параметр принимает адрес байта, который содержится в границах представления файла, проецируемого в память. Переданный адрес округляется до значения, кратного размеру страниц. Второй параметр определяет количество байтов, которые надо записать в дисковый образ файла. Если *FlushViewOfFile* вызывается в отсутствие измененных данных, она просто возвращает управление.

При успешном выполнении функция возвращает TRUE, в случае неудачи – FALSE.

Этап 5: закрытие объекта «проекция файла»

Закончив работу с любым открытым объектом ядра, нужно его закрыть, иначе в процессе начнется утечка ресурсов. Конечно, по завершении процесса система автоматически закроет объекты, оставленные открытыми. Но если процесс поработает еще какое-то время, может накопиться слишком много не-закрытых описателей.

Для закрытия объекта «проекция файла» используется функция *CloseHandle.*

5. Лабораторная работа

Использование системных функций Windows для работы с файловой системой

Цель работы

Получить практические навыки по использованию системных функций для работы с файловыми данными.

Порядок выполнения работы

1. Разработать приложение с графическим пользовательским интерфейсом в соответствии с нижеприведённым заданием.

С помощью функции **CreateDirectory** в рабочей директории создать директорию с именем **LABDIR**. Скопировать в неё все файлы из указываемой пользователем директории с расширением ***.txt** (использовать функцию **CopyFile** и функции поиска файлов – FindFirstFile, FindNextFile и др.). Файлы должны копироваться и из указанной директории, и из всех поддиректорий. В процессе копирования переименовать файлы по шаблону **var№_файл№.html**, где **var№** – номер варианта; **файл№** – порядковый номер файла.

Создать текстовый файл (функция **CreateFile**) в который записать имена файлов до переименования и соответствующий им список переименованных файлов (функции **WriteFile** и **SetFilePointer**).

С помощью функций для работы с проекцией файла (**CreateFileMapping**, **MapViewOfFile**, **UnmapViewOfFile** и **FlushViewOfFile** и др.) в начало каждого файла вставить теги **<html><body>**, в конец файла вставить теги **</html></body>** и обработать содержимое всех файлов в **LABDIR** в соответствии с одним из нижеприведённых индивидуальных вариантов:

1) удалить все символы-пробелы;

2) все слова, содержащие символы «d», с помощью html-тегов сделать подчёркнутыми;

3) все слова, в которых присутствует символ «a», с помощью html-тегов выделить жирным шрифтом;

4) в файле все слова, в которых присутствует символ «a», с помощью html-тегов выделить жирным шрифтом;

5) удалить все слова, в которых присутствует символ «e»;

6) все слова, которые начинаются с прописной буквы, с помощью html-тегов сделать подчѐркнутыми.

При обработке ошибок там, где это возможно, использовать функции GetLastError и FormatMessage.

Выполнить вышеописанные действия, но без проекции файла для обработки данных.

2. Продемонстрировать результат работы преподавателю.

3. Согласовать с преподавателем содержание отчёта и подготовить его.

4. Защитить работу.

Литература

1. Румянцев, П. В. Работа с файлами в Win32 API / Румянцев, П. В. – СПб. : Питер, 2005.

2. Рихтер, Дж. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Рихтер Дж. ; пер. с англ. – 4-е изд. СПб. : Питер; М. : Издательско-торговый дом «Русская Редакция», 2004.

3. Финогенов, К. Г. Win32. Основы программирования / К. Г. Финогенов. – 2-е изд., испр. и дополн. – М. : ДИАЛОГ МИФИ, 2006.

Содержание

Св. план 2011, поз. 13

Учебное издание

Лихачев Денис Сергеевич
Лившиц Михаил Зенадьевич
Новиков Алексей Евгеньевич и др.

Использование системных функций Windows для работы с файловой системой

Методическое пособие
по курсу
«Системное программирование»
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

Редактор Т. П. Андрейченко
Корректор А. В. Тюхай

Подписано в печать 13.07.2011.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л.
Уч.- изд. л. 2,0.	Тираж 130 экз.	Заказ 65.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 30.04.2009.
220013, Минск, П. Бровки, 6