Использование файловой системы и функций символьного ввода/вывода Windows

Нередко самыми первыми средствами операционной системы (ОС), с которыми разработчик сталкивается в любой системе, являются файловая система и простой терминальный ввод/вывод. Ранние ОС для PC, такие как MS-DOS, не могли дать ничего больше, кроме возможностей работы с файлами и терминального (или *консольного)* ввода/вывода, но эти же ресурсы и сейчас занимают центральное место почти в любой ОС.

Файлы играют очень важную роль в организации долговременного хранения данных и программ и обеспечивают простейшую форму межпрограммного взаимодействия. Помимо этого, многие аспекты файловых систем оказываются применимыми также к взаимодействию между процессами и сетям.

На примере программ копирования файлов, которые рассматривались в главе 1, вы уже познакомились с четырьмя важными функциями, обеспечивающими последовательную обработку файлов:

CreateFile
ReadFile
WriteFile
CloseHandle

В данной главе не только подробно описываются эти и родственные им функции, но и обсуждаются функции, предназначенные для обработки символов и обеспечения консольного ввода/вывода. Сначала будут кратко охарактеризованы существующие типы файловых систем и их основные свойства. Далее будет показано, каким образом введение расширенной формы символов в кодировке Unicode помогает приложениям справиться с проблемой поддержки национальных языков. Главу завершает введение в управление файлами и каталогами в Windows.

Файловые системы Windows

Windows поддерживает на непосредственно подключенных устройствах файловые системы четырех типов, но только первый из них будет иметь для нас существенное значение на протяжении всей книги, поскольку именно полнофункциональная файловая система этого типа рекомендуется компанией Microsoft для использования в качестве основной:

1. Файловая система *NT* (NTFS) — современная файловая система, которая поддерживает длинные имена файлов, а также безопасность, устойчивость к сбоям, шифрование, сжатие, расширенные атрибуты, и позволяет работать с очень большими файлами и объемами данных. Заметьте, что на гибких дисках (флоппи-дисках, или дискетах) система NTFS использоваться не может; не поддерживается она и системами Windows 9x.

2. Файловые системы *FAT* и *FAT32* (от *File Allocation Table* — таблица размещения файлов) происходят от 16-разрядной файловой системы

(FAT16), первоначально использовавшейся в MS-DOS и Windows 3.1. FAT32 впервые была введена в Windows 98 для поддержки жестких дисков большого объема и других усовершенствованных возможностей; далее под термином FAT мы будем подразумевать любую из вышеуказанных версий. FAT является единственно доступной файловой системой для дисков (но не компакт-дисков), работающих под управлением Windows 9x, а также гибких дисков. Разновидностью FAT является TFAT — ориентированная на поддержку механизма транзакций версия, используемая в Windows CE. Постепенно FAT выходит из употребления и в большинстве случаев ее можно встретить лишь на устаревших системах, особенно тех, обновление которых после первоначальной установки на них Windows 9x выполнялось без преобразования типа существующей файловой системы.

3. Файловая система *компакт-дисков* (CDFS), как говорит само ее название, предназначена для доступа к информации, записанной на компакт-дисках. CDFS удовлетворяет требованиям стандарта ISO 9660.

4. *Универсальный дисковый формат* (Universal Disk Format, UDF) поддерживает диски DVD и, в конечном итоге, должен полностью вытеснить систему CDFS. Поддержка UDF в Windows XP поддерживает как чтение, так и запись файлов, тогда как в Windows 2000 для UDF обеспечивается только запись.

Windows поддерживает, причем как на стороне клиента, так и на стороне сервера, такие распределенные файловые системы, как Networked File System (Сетевая файловая система), или NFS, и Common Internet File System (Общая межсетевая файловая система), или CIFS; на серверах обычно используют NTFS. В Windows 2000 и Windows Server 2003 обеспечивается широкая поддержка сетевых хранилищ данных (Storage Area Networks, SAN) и таких развивающихся технологий хранения данных, как IP-хранилища. Кроме того, Windows дает возможность разрабатывать пользовательские файловые системы, которые поддерживают тот же API доступа к файлам, что и API, рассматриваемый в этой и следующей главах.

Доступ к файловым системам любого типа осуществляется одинаковым образом, иногда с некоторыми ограничениями. Например, поддержка защиты файлов обеспечивается только в NTFS. В необходимых случаях мы будем обращать ваше внимание на особенности, присущие только NTFS, но в этой книге, как правило, будет предполагаться использование именно этой системы.

Формат файловой системы (FAT, NTFS или пользовательской), будь то для диска в целом, или для его разделов, определяется во время разбивки диска на разделы.

Правила именования файлов

Windows поддерживает обычную иерархическую систему имен файлов, соглашения которой, однако, несколько отличаются от соглашений, привычных для пользователей UNIX, и основаны на следующих правилах:

• Полное имя файла на диске, содержащее путь доступа к нему, начинается с указания буквенного имени диска, например, A: или C:. Обычно буквы A: и B: относятся к флоппи-дисководам, а C:, D: и так далее — к жестким дискам и приводам компакт-дисков. Последующие буквы алфавита, например, H: или K:, обычно соответствуют сетевым дискам. *Примечание.* Буквенные обозначения дисков не поддерживаются в Windows CE.

• Существует и другой возможный вариант задания полного пути доступа — использование универсальной кодировки имен (Universal Naming Code, UNC), в соответствии с которой указание пути начинается с глобального корневого каталога, обозначаемого двумя символами обратной косой черты (\\), с последующим указанием имени сервера и *имени разделяемого ресурса* (share name) для определения местоположения ресурса на файловом сервере сети. Таким образом, первая часть полного пути доступа в данном случае будет иметь вид: \\servername\sharename.

• При указании полного пути доступа в качестве *разделителя* обычно используется символ обратной косой черты (\), но в параметрах API для этой цели можно воспользоваться также символом прямой косой черты (/), как это принято в C.

• В именах каталогов и файлов не должны встречаться символы ASCII, численные значения которых попадают в интервал 1-31, а также любой из перечисленных ниже символов:

< > : " | ? * \ /

В именах разрешается использовать пробелы. В то же время, если имена файлов, содержащие пробелы, указываются в командной строке, то каждое такое имя следует заключать в кавычки, чтобы его нельзя было интерпретировать как два разных имени, относящихся к двум отдельным файлам.

• Строчные и прописные буквы в именах каталогов и файлов не различаются, то есть имена *не чувствительны к регистру* (case-insensitive), но в то же время они *запоминают регистр* (case-retaining); другими словами, если файл был создан с именем MyFile, то это же имя будет использоваться и при его отображении, хотя, например, для доступа к файлу может быть использовано также имя myFILE.

• Длина имени каталога и файла не должна превышать 255 символов, а длина полного пути доступа ограничивается значением параметра MAX_PATH (текущим значением которого является 256).

• Для отделения имени файла от расширения используется символ точки (.), причем расширения имен (как правило, два или три символа, находящиеся справа от самой последней точки, входящей в имя файла) обозначают предположительные типы файлов в соответствии с определенными соглашениями. Так, можно ожидать, что файл atou.EXE — это исполняемый файл, а файл atou.C — файл с исходным текстом программы на языке C. Допускается использование в именах файлов нескольких символов точки.

• Одиночный символ точки (.) и два символа точки (..), используемые в качестве имен каталогов, обозначают, соответственно, текущий каталог и его родительский каталог.

После этого вступления мы можем продолжить изучение функций Windows, начатое в главе 1.

Операции открытия, чтения, записи и закрытияфайлов

Первой функцией Windows, которую мы подробно опишем, является функция CreateFile, используемая как для создания новых, так и для открытия существующих файлов. Для этой функции, как и для всех остальных, сначала приводится прототип, а затем обсуждаются соответствующие параметры и порядок работы с ней.

Создание и открытие файла

Поскольку данная функция является первой из функций Windows, к изучению которых мы приступаем, ее описание будет несколько более подробным по сравнению с остальными; для других функций часто будут приводиться лишь краткие описания. Вместе с тем, даже в случае функции CreateFile будут описаны далеко не все из возможных многочисленных значений ее параметров, однако необходимые дополнительные сведения вы всегда сможете найти в оперативной справочной системе.

Простейшее использование функции CreateFile иллюстрирует приведенный в главе 1 пример ознакомительной Windows-программы (программа 1.2), содержащей два вызова функций, в которых для параметров dwShareMode, lpSecurityAttributes и hTemplateFile были использованы значения по умолчанию. Параметр dwAccess может принимать значения GENERIC_READ и GENERIC_WRITE.

HANDLE CreateFile(LPCTSTR lpName, DWORD dwAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreate, DWORD dwAttrsAndFlags, HANDLE hTemplateFile)

Возвращаемое значение: в случае успешного выполнения — дескриптор открытого файла (типа HANDLE), иначе — INVALID_HANDLE_VALUE.

Параметры

Имена параметров иллюстрируют некоторые соглашения Windows. Префикс dw используется в именах параметров типа DWORD (32-битовые целые без знака), в которых могут храниться флаги или числовые значения, например счетчики, тогда как префикс lpsz (длинный указатель на строку, завершающуюся нулем), или в упрощенной форме — lp, используется для строк, содержащих пути доступа, либо иных строковых значений, хотя документация Microsoft в этом отношении не всегда последовательна. В некоторых случаях для правильного определения типа данных вам придется обратиться к здравому смыслу или внимательно прочесть документацию.

lpName — указатель на строку с завершающим нулевым символом, содержащую имя файла, канала или любого другого именованного объекта, который необходимо открыть или создать. Допустимое количество символов при указании путей доступа обычно ограничивается значением MAX_PATH (260), однако в Windows NT это ограничение можно обойти, поместив перед именем префикс \\?\, что обеспечивает возможность использования очень длинных имен (с числом символов вплоть до 32 К). Сам префикс в имя не входит. О типе данных LPCTSTR говорится в одном из последующих разделов, а пока вам будет достаточно знать, что он относится к строковым данным.

dwAccess — определяет тип доступа к файлу — чтение или запись, что соответственно указывается флагами GENERIC_READ и GENERIC_WRITE. Ввиду отсутствия флаговых значений READ и WRITE использование префикса GENERIC_ может показаться излишним, однако он необходим для совместимости с именами макросов, определенных в заголовочном файле Windows WINNT.H. Вы еще неоднократно столкнетесь с именами, которые кажутся длиннее, чем необходимо.

Указанные значения можно объединять операцией поразрядного "или" (|), и тогда для получения доступа к файлу как по чтению, так и по записи, следует воспользоваться таким выражением:

GENERIC_READ | GENERIC_WRITE

dwShareMode — может объединять с помощью операции поразрядного "или" следующие значения:

• 0 — запрещает разделение (совместное использование) файла. Более того, открытие второго дескриптора для данного файла запрещено даже в рамках одного и того же вызывающего процесса.

• FILE_SHARE_READ — другим процессам, включая и тот, который осуществил данный вызов функции, разрешается открывать этот файл для параллельного доступа по чтению.

• FILE_SHARE_WRITE — разрешает параллельную запись в файл.

Используя блокирование файла или иные механизмы, программист должен самостоятельно позаботиться об обработке ситуаций, в которых осуществляются одновременно несколько попыток записи в одно и то же место в файле. Более подробно этот вопрос рассматривается в главе 3.

lpSecurityAttributes — указывает на структуру SECURITY_ATTRIBUTES. На первых порах при вызовах функции CreateFile и всех остальных функций вам будет достаточно использовать значение NULL; вопросы безопасности файловой системы рассматриваются в главе 15.

dwCreate — конкретизирует запрашиваемую операцию: создать новый файл, перезаписать существующий файл и тому подобное. Может принимать одно из приведенных ниже значений, которые могут объединяться при помощи операции поразрядного "или" языка С.

• CREATE_NEW — создать новый файл; если указанный файл уже существует, выполнение функции завершается неудачей.

• CREATE_ALWAYS — создать новый файл; если указанный файл уже существует, функция перезапишет его.

• OPEN_EXISTING — открыть файл; если указанный файл не существует, выполнение функции завершается неудачей.

• OPEN_ALWAYS — открыть файл; если указанный файл не существует, функция создаст его.

• TRUNCATE_EXISTING — открыть файл; размер файла будет установлен равным нулю. Уровень доступа к файлу, установленный параметром dwAccess, должен быть не ниже GENERIC_WRITE. Если указанный файл существует, его содержимое будет уничтожено. В отличие от случая CREATENEW выполнение функции будет успешным даже в тех случаях, когда указанный файл не существует.

dwAttrsAndFlags — позволяет указать атрибуты файла и флаги. Всего имеется 16 флагов и атрибутов. Атрибуты являются характеристиками файла, а не открытого дескриптора, и игнорируются, если открывается существующий файл. Некоторые из наиболее важных флаговых значений приводятся ниже.

• FILE_ATTRIBUTE_NORMAL — этот атрибут можно использовать лишь при условии, что одновременно с ним не устанавливаются никакие другие атрибуты (тогда как для всех остальных флагов одновременная установка допускается).

• FILE_ATTRIBUTE_READONLY — этот атрибут запрещает приложениям осуществлять запись в данный файл или удалять его.

• FILE_FLAG_DELETE_ON_CLOSE — этот флаг полезно применять в случае временных файлов. Файл будет удален сразу же после закрытия последнего из его открытых дескрипторов.

• FILE_FLAG_OVERLAPPED — этот флаг играет важную роль при выполнении операций асинхронного ввода/вывода, описанных в главе 14.

Кроме того, существует несколько дополнительных флагов, позволяющих уточнить способ обработки файла и облегчить реализации Windows оптимизацию производительности и обеспечение целостности файлов.

• FILE_FLAG_WRITE_THROUGH — устанавливает режим сквозной записи промежуточных данных непосредственно в файл на диске, минуя кэш.

• FILE_FLAG_NO_BUFFERING — устанавливает режим отсутствия промежуточі ной буферизации или кэширования, при котором обмен данными происходит непосредственно с буферами данных программы, указанными при вызове функций ReadFile или WriteFile (описаны далее). Соответственно требуется, чтобы начала программных буферов совпадали с границами секторов, а их размеры были кратными размеру сектора тома. Чтобы определить размер сектора при указании этого флага, вы можете воспользоваться функцией GetDiskFreeSpace.

• FILE_FLAG_RANDOM_ACCESS — предполагается открытие файла для произвольного доступа; Windows будет пытаться оптимизировать кэширование файла применительно к этому виду доступа.

• FILE_FLAG_SEQUENTIAL_SCAN — предполагается открытие файла для последовательного доступа; Windows будет пытаться оптимизировать кэширование файла применительно к этому виду доступа. Оба последних режима реализуются системой лишь по мере возможностей.

hTemplateFile — дескриптор с правами доступа GENERIC_READ к шаблону файла, предоставляющему расширенные атрибуты, которые будут применены к создаваемому файлу вместо атрибутов, указанных в параметре dwAttrsAndFlags. Обычно значение этого параметра устанавливается равным NULL. При открытии существующего файла параметр hTemplateFile игнорируется. Этот параметр используется в тех случаях, когда требуется, чтобы атрибуты вновь создаваемого файла совпадали с атрибутами уже существующего файла.

Оба вызова функции CreateFile в программе 1.2 максимально упрощены за счет использования для параметров значений по умолчанию, и, тем не менее, они вполне справляются со своими задачами. В обоих случаях было бы целесообразно использовать флаг FILE_FLAG_SEQUENTIAL_SCAN. (Эта возможность исследуется в упражнении 2.3, а соответствующие результаты тестирования производительности приведены в приложении B.)

Заметьте, что для данного файла могут быть одновременно открыты несколько дескрипторов, если только это разрешается атрибутами совместного доступа и защиты файла. Открытые дескрипторы могут принадлежать одному и тому же или различным процессам. (Управление процессами описано в главе 6).

В Windows Server 2003 предоставляется функция ReOpenFile, которая возвращает новый дескриптор с иными флагами, правами доступа и прочим, нежели те, которые были указаны при первоначальном открытии файла, если только это не приводит к возникновению конфликта между новыми и прежними правами доступа.

Закрытие файла

Для закрытия объектов любого типа, объявления недействительными их дескрипторов и освобождения системных ресурсов почти во всех случаях используется одна и та же универсальная функция. Исключения из этого правила будут оговариваться отдельно. Закрытие дескриптора сопровождается уменьшением на единицу счетчика ссылок на объект, что делает возможным удаление таких не хранимых постоянно (nonpersistent) объектов, как временные файлы или события. При выходе из программы система автоматически закрывает все открытые дескрипторы, однако лучше все же, чтобы программа самостоятельно закрывала свои дескрипторы перед тем, как завершить работу.

Попытки закрытия недействительных дескрипторов или повторного закрытия одного и того же дескриптора приводят к исключениям.

BOOL CloseHandle(HANDLE hObject)

Возвращаемое значение: в случае успешного выполнения функции — TRUE, иначе — FALSE.

Функции UNIX, сопоставимые с рассмотренными выше, отличаются от них в нескольких отношениях. Функция (системный вызов) UNIX open возвращает целочисленный дескриптор (descriptor) файла, а не дескриптор типа HANDLE, причем для указания всех параметров доступа, разделения и создания файлов, а также атрибутов и флагов используется единственный целочисленный параметр oflag. Возможные варианты выбора, доступные в обеих системах, перекрываются, однако набор опций, предлагаемый Windows, отличается большим разнообразием.

В UNIX отсутствует параметр, эквивалентный параметру dwShareMode. Файлы UNIX всегда являются разделяемыми.

В обеих системах при создании файла используется информация, касающаяся его защиты. В UNIX для задания хорошо известных разрешений на доступ к файлу для владельца, членов группы и прочих пользователей используется аргумент mode.

Функция close, хотя ее и можно сопоставить с функцией CloseHandle, отличается от последней меньшей универсальностью.

Функции библиотеки C, описанные в заголовочном файле <stdio.h>, используют объекты FILE, которые можно поставить в соответствие дескрипторам (дисковые файлы, терминалы, ленточные устройства и тому подобные), связанным с потоками. Параметр mode функции fopen позволяет указать, должны ли содержащиеся в файле данные обрабатываться как двоичные или как текстовые. Имеются также опции открытия файла в режиме "только чтение", обновления файла, присоединения к другому файлу и так далее. Функция freopen обеспечивает возможность повторного использования объектов FILE без их предварительного закрытия. Средства для задания параметров защиты стандартной библиотекой C не предоставляются.

Для закрытия объектов типа FILE предназначена функция fclose. Имена большинства функций стандартной библиотеки C, предназначенных для работы с объектами FILE, снабжены префиксом "f".

Чтение файла

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

Возвращаемое значение: в случае успешного выполнения (которое считается таковым, даже если не был считан ни один байт из-за попытки чтения с выходом за пределы файла) — TRUE, иначе — FALSE.

Пока мы будем предполагать, что дескрипторы файлов создаются *без* указания флага перекрывающегося ввода/вывода FILE_FLAG_OVERLAPPED в параметре dwAttrsAndFlags. В этом случае функция ReadFile начинает чтение с текущей позиции указателя файла, и указатель файла сдвигается на число считанных байтов.

Если значения дескриптора файла или иных параметров, используемых при вызове функции, оказались недействительными, возникает ошибка, и функция возвращает значение FALSE. Попытка выполнения чтения в ситуациях, когда указатель файла позиционирован в конце файла, не приводит к ошибке; вместо этого количество считанных байтов (*lpNumberOfBytesRead) устанавливается равным 0.

Параметры

Описательные имена переменных и естественный порядок расположения параметров во многом говорят сами за себя. Тем не менее, ниже приводятся некоторые краткие пояснения.

hFile — дескриптор считываемого файла, который должен быть создан с правами доступа GENERIC_READ. lpBuffer является указателем на буфер в памяти, куда помещаются считываемые данные. nNumberOfBytesToRead — количество байт, которые должны быть считаны из файла.

lpNumberOfBytesRead — указатель на переменную, предназначенную для хранения числа байт, которые были фактически считаны в результате вызова функции ReadFile. Этот параметр может принимать нулевое значение, если перед выполнением чтения указатель файла был позиционирован в конце файла или если во время чтения возникли ошибки, а также после чтения из именованного канала, работающего в режиме обмена сообщениями (глава 11), если переданное сообщение имеет нулевую длину.

lpOverlapped — указатель на структуру OVERLAPPED. На данном этапе просто устанавливайте значение этого параметра равным NULL.

Запись в файл

BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)

Возвращаемое значение: в случае успешного выполнения — TRUE, иначе — FALSE.

Все параметры этой функции вам уже знакомы. Заметьте, что успешное выполнение записи еще не говорит о том, что данные действительно оказались записанными на диск, если только при создании файла с помощью функции CreateFile не был использован флаг FILE_FLAG_WRITE_THROUGH. Если во время вызова функции указатель

файла был позиционирован в конце файла, Windows увеличит длину существующего файла.

Функции ReadFileGather и WriteFileGather позволяют выполнять операции чтения и записи с использованием набора буферов различного размера.

Сопоставимыми функциями UNIX являются функции read и write, которым программист в качестве параметров должен предоставлять дескриптор файла, буфер и счетчик байтов. Возвращаемые значения этих функций указывают на количество фактически переданных байтов. Возврат функцией read значения 0 означает чтение конца файла, а значения –1 — возникновение ошибки. В противоположность этому в Windows для подсчета количества переданных байтов используется отдельный счетчик, а на успех или неудачу выполнения функции указывает возвращаемое ею булевское значение.

В обеих системах функции имеют сходное назначение и могут выполнять соответствующие операции с использованием файлов, терминалов, ленточных устройств, каналов и так далее.

Входящие в состав стандартной библиотеки C функции read и fwrite, выполняющие операции ввода/вывода в двоичном режиме, вместо счетчика одиночных байтов, как в UNIX и Windows, используют размер объекта и счетчик объектов. Преждевременное прекращение передачи данных может быть вызвано как достижением конца файла, так и возникновением ошибки; точная причина устанавливается с использованием функций feof или ferror. Библиотека предоставляет полный набор функций, ориентированных на работу с текстовыми файлами, таких как fgets или fputs, для которых в каждой из рассматриваемых ОС аналоги вне библиотеки C отсутствуют.