**абота с каталогами**

Созданием папки в WinAPI занимается функция **CreateDirectory**

BOOL CreateDirectory(

LPCTSTR lpPathName,// указатель на строку пути

LPSECURITY\_ATTRIBUTES lpSecurityAttributes // указатель на SECURITY\_ATTRIBUTES

);

Структура SECURITY\_ATTRIBUTES имеет вид:

typedef struct \_SECURITY\_ATTRIBUTES {

DWORD nLength;

LPVOID lpSecurityDescriptor;

BOOL bInheritHandle;

} SECURITY\_ATTRIBUTES;

Кому охота в этом разбираться - пускай закуривают MSDN. Мне это тоже интересно, но требует тщательного разбора (что бы вы не думали про Windows, но к проблемам разделения доступа в MicroSoft подошли ответственно, и парой слов дескрипторы безопасности не описать), поэтому создавать папки мы будем с правами доступа по умолчанию, подставляя вместо указателя на SECURITY\_ATTRIBUTES просто NULL

У CreateDirectory есть "старшая сестра" - **CreateDirectoryEx**, отличающаяся тем, что принимает три параметра: указатель на путь к папке-шаблону, указатель на путь к создаваемой папке и указатель на SECURITY\_ATTRIBUTES. Зачем? А чтобы можно было, не заморачиваясь, наделить новую папку теми же атрибутами, что и у шаблона (в т.ч. и атрибутами безопасности)

Чтобы удалить папку, нужно вызвать функцию **RemoveDirectory**, принимающую единственный параметр - указатель на строку пути

BOOL RemoveDirectory(LPCTSTR lpPathName);

Единственное замечание: чтобы папку можно было удалить, она должна быть пустой, и удаление этой папки не должно быть запрещено атрибутами безопасности

Кстати, [в той статье](http://rfteam.110mb.com/asm/7.php) мы изучили, как задать текущую папку, чтобы проводить в ней файловые операции. Но как узнать текущую папку, чтобы понять, откуда запустили наше приложение? Это делает API-функция **GetCurrentDirectory**

DWORD GetCurrentDirectory(

DWORD nBufferLength,// размер буфера в символах (оставьте место и под завершающий ноль)

LPTSTR lpBuffer // указатель на буфер, куда положим путь к текущей директории

);

В случае неудачи функция возвращает 0, если всё прошло успешно - количество скопированных в буфер символов

Ещё мы запросто можем узнать пути к системным папкам: это делается функциями **GetSystemDirectory** и **GetWindowsDirectory**

UINT GetSystemDirectory(

LPTSTR lpBuffer,// указатель на буфер для пути к системной папке (на XP - system32)

UINT uSize // размер буфера в символах

);

UINT GetWindowsDirectory(

LPTSTR lpBuffer,// указатель на буфер для пути к папке с Виндой (обычно C:\Windows)

UINT uSize // размер буфера в символах

);

Обе функи при удачном стечении обстоятельств возвращают длину полученного пути, а при облажании - 0

Чтобы получить полный путь к файлу, юзают **GetFullPathName**

DWORD GetFullPathName(

LPCTSTR lpFileName,// указатель на строку с именем файла

DWORD nBufferLength,// размер буфера в символах

LPTSTR lpBuffer,// указатель на буфер для полного пути

LPTSTR \*lpFilePart // непонятно... обычно NULL

);

Если всё удачно - функция вернёт длину получившейся строки, если не получилось - 0

**Копирование и перемещение файлов**

Для копирования файлов в Win32 используется функция **CopyFile**

BOOL CopyFile(

LPCTSTR lpExistingFileName,// указатель на имя старого файла

LPCTSTR lpNewFileName,// указатель на имя нового файла

BOOL bFailIfExists // типа флаг

);

"Типа флаг" определяет поведение функции, если новое имя копируемого файла совпадает с именем уже существующего в этой папке файла. Если флаг равен 0, то происходит перезапись того файла, если 1 - копирование отменяется

CopyFile была хороша для Win9x/Me. Но вот пришло следующее поколение - WinNT (NewTechnology), принесшее с собой многие новшества, в т.ч. и файловую систему NTFS, поддерживающую альтернативные потоки данных внутри файла. При обычном копировании эти потоки терялись, поэтому MicroSoft включили в системные библиотеки новую функцию **CopyFileEx**, поддерживающую альтернативные потоки и ещё кучу всего (но не копирующую атрибуты безопасности)

BOOL CopyFileEx(

LPCWSTR lpExistingFileName,// указатель на имя старого файла

LPCWSTR lpNewFileName,// указатель на имя нового файла

LPPROGRESS\_ROUTINE lpProgressRoutine,// указатель на callback-функцию

LPVOID lpData,// аргумент callback-функции

LPBOOL pbCancel,// флаг отмены операции

DWORD dwCopyFlags// флаги копирования

);

Начнём объяснение параметров функции с конца. dwCopyFlags может быть комбинацией двух значений: COPY\_FILE\_FAIL\_IF\_EXISTS (прекратить копирование, если файл с таким именем уже существует) и COPY\_FILE\_RESTARTABLE (если копирование файла не удалось, повторить через некоторое время). Флаг pbCancel проверяется во время копирования, и, если он установлен в 1, копирование отменяется. И, наконец, самое интересное - callback-функция

Callback-функция выполняется каждый раз, как скопировалась очередная порция данных. Её называют **CopyProgressRoutine**

DWORD WINAPI CopyProgressRoutine(

LARGE\_INTEGER TotalFileSize,// общий размер файла в байтах

LARGE\_INTEGER TotalBytesTransferred,// общее количество скопированных байт

LARGE\_INTEGER StreamSize,// общий размер данного потока

LARGE\_INTEGER StreamBytesTransferred,// общее количество скопированных из этого потока байт

DWORD dwStreamNumber,// номер текущего потока

DWORD dwCallbackReason,// причина callback'а

HANDLE hSourceFile,// хэндл исходного файла

HANDLE hDestinationFile,// хэндл конечного файла

LPVOID lpData// аргумент, переданный CopyFileEx

);

Причинами callback'а могут быть очередная скопированная порция данных (CALLBACK\_CHUNK\_FINISHED) или требование очередного потока его скопировать (CALLBACK\_STREAM\_SWITCH - именно по этой причине CopyProgressRoutine вызывается в первый раз с номером потока 1). Возвращает функция одно из следующих значений: PROGRESS\_CONTINUE (продолжить копирование), PROGRESS\_CANCEL (отменить его нахрен и удалить конечный файл), PROGRESS\_STOP (приостановить копирование и продолжить его попозже), PROGRESS\_QUIET (продолжить копирование, но больше не вызывать callback-функцию)

В общем, задумка хорошая, но можно обойтись и без этого, попросту передав вместо указателя на callback-функцию и её аргумента NULL'ы

Для перемещения файлов существует функция **MoveFile**

BOOL MoveFile(

LPCTSTR lpExistingFileName,// указатель имени файла

LPCTSTR lpNewFileName // указатель на новое имя файла

);

Эта функция способна переносить (а заодно и переименовывать) файлы и папки, причём файлы - куда угодно, а папки - только в пределах одного диска

Попутно майкрософтовцы создали ещё одну функцию - **MoveFileEx**, снабдив её расширенными возможностями

BOOL MoveFileEx(

LPCTSTR lpExistingFileName,// указатель на имя файла

LPCTSTR lpNewFileName,// указатель на новое имя файла

DWORD dwFlags // флаги

);

MoveFileEx позволяет подстановку во второй параметр NULL'а, что приводит к удалению файла. Опять же: файлы перемещайте, куда угодно, а папки - в пределах диска. dwFlags может быть комбинацией следующих флагов: MOVEFILE\_COPY\_ALLOWED (устанавливается, если нужно перемещать на другой диск, тогда вызов MoveFileEx распадается на CopyFile и DeleteFile), MOVEFILE\_DELAY\_UNTIL\_REBOOT (этот флаг поддерживают только NT: файл будет перемещён после перезагрузки - после работы autochk, но до создания файла подкачки), MOVEFILE\_REPLACE\_EXISTING (если файл с таким именем уже существует, затереть его нахрен:)), MOVEFILE\_WRITE\_THROUGH (поддерживают только NT: функция вернёт управление только тогда, когда удостоверится, что файл в натуре перемещён)

Специальной команды переименования в WinAPI нет, ведь перемещение с другим именем в пределах одной папки - и есть переименование

**Работа с атрибутами файла**

Каждый файл в Windows имеет свои атрибуты, определяющие его доступность для чтения/записи и не только. Для работы с ними существуют функции **SetFileAttributes** и **GetFileAttributes**. Первая из них устанавливает атрибуты файла, а вторая позволяет их просмотреть

BOOL SetFileAttributes(

LPCTSTR lpFileName,// указатель на имя файла

DWORD dwFileAttributes // указатель на атрибуты

);

А атрибуты мы можем установить вот такие: FILE\_ATTRIBUTE\_ARCHIVE (архивный: готов для архивирования или удаления), FILE\_ATTRIBUTE\_HIDDEN (скрытый), FILE\_ATTRIBUTE\_NORMAL (просто файл - ничего особенного; этот атрибут не комбинируется с другими), FILE\_ATTRIBUTE\_READONLY (только для чтения), FILE\_ATTRIBUTE\_SYSTEM (системный), FILE\_ATTRIBUTE\_TEMPORARY (временный), FILE\_ATTRIBUTE\_OFFLINE (непосредственный доступ к данным невозможен)

DWORD GetFileAttributes(

LPCTSTR lpFileName // указатель на имя файла или папки

);

Эта функция в случае провала вернёт в eax -1, а в случае успешного выполнения - атрибуты файла или папки. Атрибуты те же самые, что и в случае SetFileAttributes, но добавляются ещё два: FILE\_ATTRIBUTE\_DIRECTORY (а файл-то - папка!) и FILE\_ATTRIBUTE\_COMPRESSED (сжатый)

Кроме атрибутов файлы обладают размером, который можно определить функцией **GetFileSize**

DWORD GetFileSize(

HANDLE hFile,// хэндл файла

LPDWORD lpFileSizeHigh // указатель на переменную

);

В случае неудачи функция возвращает в eax -1, при удачном стечении обстоятельств - младшие 32 бита результата в eax и старшие 32 бита - в указанной переменной

А ещё файл характеризуется временем создания, временем последнего доступа и временем последней записи. Чтобы их получить, нужно заюзать функцию **GetFileTime**

BOOL GetFileTime(

HANDLE hFile,// хэндл файла

LPFILETIME lpCreationTime,// указатель на время создания

LPFILETIME lpLastAccessTime,// указатель на время последнего доступа

LPFILETIME lpLastWriteTime // указатель на время последней записи

);

Каждый указатель адресует 64-битное расположение в памяти, где первые 32 бита представляют собой младший dword времени, а следующие 32 - старший dword

Чтобы установить свои значения времени для файла, нужно заюзать API-шку **SetFileTime**, которая удивительно похожа на GetFileTime: также принимает хэндл файла и три указателя на время (с единственным отличием - здесь время уже задано). Кстати, и в той, и в другой функции, если какое-то из времён вас не интересует, можно задать вместо указателя на него NULL

**Создание, открытие и закрытие файла**

Создание и открытие файла в Win32 производится одной функцией **CreateFile**

HANDLE CreateFile(

LPCTSTR lpFileName,// указатель на имя файла

DWORD dwDesiredAccess,// режим доступа

DWORD dwShareMode,// режим разделения

LPSECURITY\_ATTRIBUTES lpSecurityAttributes,// указатель на атрибуты безопасности

DWORD dwCreationDistribution,// параметры открытия

DWORD dwFlagsAndAttributes,// атрибуты файла и флаги

HANDLE hTemplateFile // хэндл шаблонного файла

);

Теперь о параметрах подробнее: режим доступа может быть GENERIC\_READ (доступен для чтения), GENERIC\_WRITE (доступен для записи) или GENERIC\_READ+GENERIC\_WRITE

Режим разделения: NULL - доступ к файлу полностью монополизирован открывшим его процессом, FILE\_SHARE\_READ - другие процессы могут читать файл, но запись в него монополизирована, FILE\_SHARE\_WRITE - другие процессы могут записывать в файл, но чтение из него монополизировано, FILE\_SHARE\_READ+FILE\_SHARE\_WRITE - тут понятно без слов. Под WinNT есть ещё режим FILE\_SHARE\_DELETE, когда и чтение, и запись монополизированы открывшим файл процессом, но другие процессы могут его удалить

dwCreationDistribution определяет, что делать в "спорных ситуациях". CREATE\_NEW - создать новый файл (если файл с таким именем уже существует, то функция выдаёт ошибку), CREATE\_ALWAYS - создать новый файл (если уже существует - затереть его), OPEN\_EXISTING - открыть существующий файл (если не существует - ошибка), OPEN\_ALWAYS - открыть существующий файл (если не существует - создать), TRUNCATE\_EXISTING - открыть файл с усечением его до нулевой длины (если не существует - ошибка)

Ну, про атрибуты файла сказано было уже много. Здесь они те же: FILE\_ATTRIBUTE\_ARCHIVE, FILE\_ATTRIBUTE\_COMPRESSED, FILE\_ATTRIBUTE\_HIDDEN, FILE\_ATTRIBUTE\_NORMAL, FILE\_ATTRIBUTE\_OFFLINE, FILE\_ATTRIBUTE\_READONLY, FILE\_ATTRIBUTE\_SYSTEM, FILE\_ATTRIBUTE\_TEMPORARY и их комбинации. Интереснее флаги: FILE\_FLAG\_OVERLAPPED - асинхронные чтение и запись в файл разрешены с определённого смещения, FILE\_FLAG\_WRITE\_THROUGH - не использовать промежуточное кэширование при записи на диск, FILE\_FLAG\_NO\_BUFFERING - не использовать средства буферизации ОС, FILE\_FLAG\_RANDOM\_ACCESS и FILE\_FLAG\_SEQUENTIAL\_SCAN - помогают системе оптимизировать кэширование, FILE\_FLAG\_DELETE\_ON\_CLOSE - удалить файл после закрытия, FILE\_FLAG\_DELETE\_ON\_CLOSE - файл открыт для резервного копирования или восстановления (только на NT), FILE\_FLAG\_DELETE\_ON\_CLOSE - доступ к файлу возможен при использовании правил POSIX

Если CreateFile открывает клиентскую часть именованного канала, то среди флагов могут содержаться данные о безопасности от службы QoS. Если вызывающее приложение установило флаг SECURITY\_SQOS\_PRESENT, то в dwFlagsAndAttributes могут быть SECURITY\_ANONYMOUS, SECURITY\_IDENTIFICATION, SECURITY\_IMPERSONATION, SECURITY\_DELEGATION, SECURITY\_CONTEXT\_TRACKING, SECURITY\_EFFECTIVE\_ONLY - я хреново понимаю, что это значит, кому интересно - курите MSDN

Хэндл шаблонного файла используется для создания нового файла с теми же атрибутами, что и у шаблона

Функция CreateFile возвращает хэндл файла или INVALID\_HANDLE\_VALUE в случае неудачи

Закрыть файл можно булевой функцией **CloseHandle**, принимающей только один параметр - закрываемый хэндл

Чтение и запись осуществляются уже знакомыми вам ReadFile и WriteFile, а чтобы установить позицию указателя в файле, используется функция **SetFilePointer**

DWORD SetFilePointer(

HANDLE hFile,// хэндл файла

LONG lDistanceToMove,// на сколько байт продвинуть указатель?

PLONG lpDistanceToMoveHigh,// указатель на старшее слово дистанции продвижения указателя

DWORD dwMoveMethod // точка отсчёта

);

Точка отсчёта может быть задана, как FILE\_BEGIN (отсчитывать дистанцию от начала файла), FILE\_CURRENT (отсчитывать от текущей позиции курсора), FILE\_END (отсчитывать от конца файла). Дистанцию можно задавать как положительным, так и отрицательным числом байт - указатель ведь должен двигаться в обе стороны. Если мы ставим NULL вместо адреса старшего слова дистанции, то ограничиваем дальнобойность указателя 2^32-2 байт. Если пропишем адрес, то диапазон расширяется до 2^64 - 2

**Кое-что можно упростить...**

Итак, мы перечислили основные API для работы с файлами. Согласитесь, всё довольно просто и понятно, но как-то громоздко: узнать атрибуты файла - одна функция, размер - другая, время создания - третья, полный путь - четвёртая... Долго и неэлегантно. Вот потому и существует функция **GetFileInformationByHandle**

BOOL GetFileInformationByHandle(

HANDLE hFile, // хэндл файла

LPBY\_HANDLE\_FILE\_INFORMATION lpFileInformation // указатель на BY\_HANDLE\_FILE\_INFORMATION

);

Ну а структура BY\_HANDLE\_FILE\_INFORMATION в свою очередь выглядит так:

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;

Эта структура содержит в себе тотальное досье на файл: атрибуты; время создания, доступа и записи; серийный номер дискового тома, на котором он расположен; размер; количество ссылок на файл (на FAT всегда 1, на NTFS можно и больше), идентификатор файла

**ShellAPI**

Ещё одно упрощение можно сделать для базовых операций с файлами (удаление, копирование, переименование, перемещение) - использовать всего одну функцию для всех этих действий

Дело в том, что все вышеперечисленные функции экспортируются kernel32.dll. Эта либа содержит базовые API-шки, без которых не сможет работать ни одно приложение (а даже если и сможет, системный загрузчик не запустит программу, в секции импорта которой не значится kernel32.dll), я даже больше скажу: импортируя функи из одной только kernel32.dll, можно получить доступ ко всему многообразию WinAPI-функций

Однако, кроме базовых, есть специальные API-шки для файлов и папок - ShellAPI, функции которого сосредоточены в библиотеке shell32.dll. Начиная с четвёртой версии, в этой библе появилась функция **SHFileOperation**, предоставляющая почти неограниченные возможности:

WINSHELLAPI int WINAPI SHFileOperation(

LPSHFILEOPSTRUCT lpFileOp //указатель на структуру \_SHFILEOPSTRUCT

);

Ну а структура \_SHFILEOPSTRUCT уже и содержит детальные указания: какой файл имеется в виду, что с ним делать и т.д. Эта структура состоит из следующих полей:

typedef struct \_SHFILEOPSTRUCT {

HWND hwnd; //хэндл окна, заказавшего файловые операции (туда мы прогресс-бар покажем)

UINT wFunc; //Что делать с файлом?

LPCSTR pFrom; //где находится файл?

LPCSTR pTo; //и куде его перемещать, переносить и т.д.

FILEOP\_FLAGS fFlags; //поле флагов

BOOL fAnyOperationsAborted; //это поле сообщит: не прервал ли юзер выполнение функции

LPVOID hNameMappings; //сам ещё не разобрался

LPCSTR lpszProgressTitle; //заголовок к прогресс-бару

} SHFILEOPSTRUCT, FAR \*LPSHFILEOPSTRUCT;

В фасмовских инклудах есть понятие об SHFileOperation, но об \_SHFILEOPSTRUCT там и не слыхивали (по крайней мере, в версии 1.67.12, которую юзаю я), поэтому инклуду придётся варганить самим. Вот что наколдовал я:

;описуха структуры,

;взята из масмовских инклудов

struct \_SHFILEOPSTRUCT

 hwnd dd ?

 wFunc dd ?

 pFrom dd ?

 pTo dd ?

 fFlags dd ?

 fAnyOperationsAborted dd ?

 hNameMappings dd ?

 lpszProgressTitle dd ?

ends

;числовые коды символьных имён

;функций и флагов

FO\_MOVE = 1h;переместить

FO\_COPY = 2h;копировать

FO\_DELETE = 3h;удалить

FO\_RENAME = 4h;переименовать

FOF\_MULTIDESTFILES = 1h;в поле pTo - несколько имён

FOF\_CONFIRMMOUSE = 2h

FOF\_SILENT = 4h;не выводить прогресс-бар

FOF\_RENAMEONCOLLISION = 8h;переименовать при совпадении

FOF\_NOCONFIRMATION = 10h;не требовать подтверждения

FOF\_WANTMAPPINGHANDLE = 20h

FOF\_ALLOWUNDO = 40h;сохранять информацию для отката операции

FOF\_FILESONLY = 80h;обрабатывать только файлы

FOF\_SIMPLEPROGRESS = 100h;показывать прогресс-бар, но не светить имена файлов

FOF\_NOCONFIRMMKDIR = 200h;создавать новую папку без подтверждения

Теперь закатайте всё это в текстовый файл, обзовите его, например, shellapiA.inc и положите в папку с фасмовскими инклудами. Этот файлик нам очень пригодится
Что ж, быстренько напишем консольную программку, реализующую удаление файлов с помощью функции SHFileOperation. Большая часть кода вам уже знакома, комментировать буду то, что для вас ново:

|  |
| --- |
| **format PE console****include 'win32axp.inc'**;подключаем инклуду win32axp.inc;и нашу самописную shellapiA.inc**include 'shellapiA.inc'****section '.data' data readable writeable****ns dd ?****hout dd ?****buffer db 261 dup (?)****help db 'Using: shdel.exe path\_to\_file',0****Retry db 'Program fails. Please, retry',0****shf \_SHFILEOPSTRUCT 0,FO\_DELETE,0,0,FOF\_NOCONFIRMATION,0,0**;объявляем переменную shf типа \_SHFILEOPSTRUCT;и заполняем поля hwnd,wFunc,fFlags**section '.code' code readable executable****fuck:****invoke GetStdHandle,STD\_OUTPUT\_HANDLE****mov [hout],eax****invoke GetCommandLine****mov esi,eax****cycle1:** **cmp byte [esi],20h** **je parameter** **cmp byte [esi],0Dh** **je najobka** **inc esi** **jmp cycle1****parameter:** **mov edi,buffer** **mov ecx,260** **cycle2:** **inc esi** **mov al,byte [esi]** **cmp al,0Dh** **je konets** **mov byte [edi],al** **inc edi** **loop cycle2****konets:** **mov byte [edi],0** **invoke lstrlen,buffer** **test eax,eax** **jz najobka**;пропарсив командную строку,;получили в buffer путь и имя удаляемого файла**mov eax,buffer****mov [shf.pFrom],eax**;заполняем в shf поле pFrom;и вызываем функцию**invoke SHFileOperation,shf****exit:** **invoke ExitProcess,0****najobka:** **invoke WriteConsole,[hout],help,29,ns,NULL** **jmp exit****.end fuck** |

Как видите, по сравнению с DeleteFile SHFileOperation позволяет сократить размер кода. Правда, её использование сопряжено с немалым умственным геморроем: у меня (на WinXP SP1 Pro) приведенный выше код нормально компилировался и запускался, но попытка удаления файла с именем короче 3-х символов приводила к ошибке чтения с диска. Удаление по маске тоже не всегда удаётся... Не в этом ли причина того, что при всём удобстве ShellAPI-функций программисты юзают их довольно редко? Поэтому, уважаемые читатели, мне было бы очень интересно узнать, каковы ваши впечатления от этой API-шки: работает ли она в вашей системе, не глючит ли?

Что ж, до новых встреч, уважаемые читатели! Надеюсь, вам с нами интересно:)