Работа с файлами в WINDOWS

Создание файлов

Для создания нового или открытия существующего файла используется функция CreateFile. При использовании функции**CreateFile**необходимо указать, предполагается чтение или запись в файл или и то и другое. Также необходимо указать, необходимые действия в случае наличия файла на диске или его отсутствия (например, перезаписывать файл, если он существует и создавать новый, если – нет). Также функция**CreateFile**позволяет указать возможность разделения файла с другими приложениями (одновременного чтения/записи нескольких приложений в файл). Если некоторое приложение монополизировало доступ к файлу на чтение и/или запись, то никакое другое приложение не сможет читать и/или писать в файл, пока первое его не закроет.

Операционная система присваивает каждому открытому файлу уникальный идентификатор (дескриптор, filehandle), который может быть использован при обращениях к файлу для записи, чтения и получения информации о файле. Дескриптор действителен до тех пор, пока файл не закрыт. Приложение может наследовать дескрипторы файлов от процесса, который его запустил (если дескрипторы наследуемые).

Функция CreateFileсреди прочих объектов позволяет обращаться к файлам и каналам (pipes). При обращении к каналам функцияCreateFileпозволяет создавать клиентское подключение к именованным каналам, находящимся в режиме ожидания подключения. Серверная часть канала создаётся функцией CreateNamedPipe. Одно приложение может создавать множество клиентских подключений к каналу, но к одному экземпляру канала может подключаться только одно приложение (Стоит отметить, что возможно существование нескольких экземпляров именованных каналов с одинаковыми именами).

Функция CreateFileимеет следующий прототип:

**HANDLE CreateFile(**

**LPCTSTR***lpFileName***,**// file name

**DWORD***dwDesiredAccess***,**// access mode

**DWORD***dwShareMode***,**// share mode

**LPSECURITY\_ATTRIBUTES***lpSecurityAttributes***,**// SD

**DWORD***dwCreationDisposition***,**// how to create

**DWORD***dwFlagsAndAttributes***,**// file attributes

**HANDLE***hTemplateFile*// handle to template file

**);**

Параметры

*lpFileName*

[in] Указатель на строку, содержащую имя объекта для создания или открытия (имя файла, путь к файлу, имя канала и пр.). Если имя задается литеральной константой, то следует поместить её в макросTEXT() или поставитьLперед открывающей кавычкой. Например, TEXT("C:\\tmp.txt") или L"C:\\tmp.txt"

*dwDesiredAccess*

[in] Описание желаемого режима доступа к файлу. Существует возможность передачи большого количества различных флагов. Нас интересуют флаги GENERIC\_READ, GENERIC\_WRITE и их объединение. При доступе к каналам следует учитывать режим создания канала сервером. Если сервер создал канал для записи, то клиент открывает его для чтения и наоборот. Если сервер создал канал для чтения и записи, то клиент может открыть его как для чтения, так и для записи.

*dwShareMode*

[in] Определяет режим разделения объекта

0 – приложение открывает файл для монопольного доступа. Последующие обращения на открытие данного файла будут безуспешными, пока данных дескриптор не будет закрыт. Для разделения доступа к файлу могут использоваться следующие ключи (один или вместе):

FILE\_SHARE\_READ – разрешены лишь последующие операции открытия только на чтение.

FILE\_SHARE\_WRITE – разрешены лишь последующие операции открытия только на запись.

*lpSecurityAttributes*

[in] Указатель на сруктуру SECURITY\_ATTRIBUTES, которая определяет возможность наследования дескриптора дочерними процессами. Можно передаватьNULL– это значит, что дескриптор не может быть наследован (для наших приложений этого достаточно).

*dwCreationDisposition*

[in] Определяет то, какие действия необходимо предпринять в случаях, если файл существует и если файл не существует. Этот параметр должен иметь одно из следующих заначений:

|  |  |
| --- | --- |
| **Значение** | **Пояснение** |
| CREATE\_NEW | Создаёт файл. Вызов заканчивается неудачей, если файл существует. |
| CREATE\_ALWAYS | Создаёт новый файл. Если файл существует, то его содержимое и атрибуты будут стёрты. |
| OPEN\_EXISTING | Открытие файла. Если файл не существует, то вызов закончится неудачей. |
| OPEN\_ALWAYS | Открывает файл. Если файл не существует, то он будет создан. |
| TRUNCATE\_EXISTING | Открывает файл, размер которого обнуляется. Файл должен открываться как минимум с режимом доступа GENERIC\_WRITE. Если файл не существует, то вызов будет неудачен. |

dwFlagsAndAttributes

[in] Позволяет задавать файловые атрибуты (только для чтения, скрытый, системный и пр.). Также позволяет сообщать операционной системе желаемое поведение при работе с файлами. Например, запись в файл без буферизации (FILE\_FLAG\_NO\_BUFFERING и FILE\_FLAG\_WRITE\_THROUGH); оптимизация для неупорядоченного доступа (FILE\_FLAG\_RANDOM\_ACCESS); открытие для асинхронного ввода/вывода (FILE\_FLAG\_OVERLAPPED).

Возвращаемое значение

Если вызов успешен, возвращается дескриптор открытого файла. Если вызов неудачен, возвращается константа INVALID\_HANDLE\_VALUE. Код ошибки можно получить вызовом функции GetLastError. Подробную информацию об ошибке (по её коду) можно получить вызовом функции FormatMessage.

Для копирования файлов используется функция CopyFile

**BOOL CopyFile(**

**LPCTSTR***lpExistingFileName***,** // имя существующего файла

**LPCTSTR***lpNewFileName***,** // имя нового файла

**BOOL***bFailIfExists*// действие, если файл существуетTRUE– ошибка

// FALSE - перезаписывать

**);**

В случае успеха возвращается ненулевое значение

Для переименования файлов и директорий используется функция MoveFile.

**BOOL MoveFile(**

**LPCTSTR***lpExistingFileName***,** // имя файла

**LPCTSTR***lpNewFileName* // новое имя файла

**);**

В случае успеха возвращается ненулевое значение

Для удаления существующих файлов используется функция **DeleteFile.**

**BOOL DeleteFile(**

**LPCTSTR***lpFileName* // имя удаляемого файла

**);**

В случае успеха возвращается ненулевое значение

Чтение/запись в файл

Каждый открытый файл имеет файловый указатель (filepointer), который указывает позицию следующего файла, который будет записан/прочтен. При открытии файла его файловый указатель перемещается на начало файла. После прочтения/записи очередного файла система перемещает файловый указатель. Файловый указатель можно перемещать, используя функцию SetFilePointer.

Для чтения/записи в файл используются функции ReadFile и WriteFile, при этом необходимо, чтобы файл был открыт на чтение и на запись соответственно.

Функция ReadFileчитает из файла указанное количество символов, начиная с позиции, обозначенной файловым указателем. При синхронном (в противоположность асинхронному) чтении файловый указатель сдвигается на фактически прочитанное количество байт.

**BOOL ReadFile(**

**HANDLE***hFile***,** // дескриптор файла

**LPVOID***lpBuffer***,** // буфер данных

**DWORD***nNumberOfBytesToRead***,** // количество байт для прочтения

**LPDWORD***lpNumberOfBytesRead***,** // количество фактически прочитанных байт

**LPOVERLAPPED***lpOverlapped*// указатель на структуру, используемую при

//асинхронном чтении

**);**

Параметры

*hFile*

[in] Дескриптор читаемого файла. Должен быть открыт с доступомGENERIC\_READ.

*lpBuffer*

[out] Указатель на буфер, принимающий данные из файла.

*nNumberOfBytesToRead*

[in] Задаёт количество байт, которые необходимо прочитать из файла.

*lpNumberOfBytesRead*

[out] Указатель на переменную, которая принимает количество реально прочитанных байт.

*lpOverlapped*

[in] Укзатель на структуру**OVERLAPPED**. При не асинхронном доступе следует передаватьNULL.

Возвращаемое значение

Если выполнение функции произошло успешно, то возвращается ненулевое значение. Если возвращено ненулевое значение, но прочитано 0 байт, значит файловый указатель стоял на конце файла перед операцией чтения.

Функция WriteFile записывает в файл данные, начиная с позиции, обозначенной файловым указателем. При синхронной (в противоположность асинхронному) записи файловый указатель сдвигается на фактически записанное количество байт.

**BOOL WriteFile(**

**HANDLE***hFile***,** // дескриптор файла

**LPCVOID***lpBuffer***,** // буфер данных

**DWORD***nNumberOfBytesToWrite***,** // количество байт для записи

**LPDWORD***lpNumberOfBytesWritten***,**// количество фактически записанных байт

**LPOVERLAPPED***lpOverlapped*// указатель на структуру, используемую при

// асинхронном чтении

**);**

Параметры

*hFile*

[in] Дескриптор файла, в который производится запись. Должен быть открыт с доступомGENERIC\_WRITE.

*lpBuffer*

[in] Указатель на буфер, содержащий данные, которые необходимо записать.

*nNumberOfBytesToWrite*

[in] Задаёт количество байт, которые необходимо записать в файл.

*lpNumberOfBytesWritten*

[out] Указатель на переменную, которая принимает количество реально записанных байт.

*lpOverlapped*

[in] Укзатель на структуру**OVERLAPPED**. При не асинхронном доступе следует передаватьNULL.

Возвращаемое значение

Если выполнение функции произошло успешно, то возвращается ненулевое значение.

Блокировка файлов

Т.к. система позволяет более чем одному приложению открывать файл и писать в него, приложения не должны одновременно писать в одну область файла. Эта проблема может быть решена путем временного блокирования части файла. Функция LockFile позволяет приложению получить в монопольное пользования определённое количество байт в файле. Отрезок заблокированных байт может выходить за текущий конец файла. Все попытки сторонних приложений обратиться к заблокированному участку файла потерпят неудачу.

Приложение может файл с помощью функции UnlockFile.

**BOOL LockFile(**

**HANDLE***hFile***,** // дескриптор файла

**DWORD***dwFileOffsetLow***,** // младшее слово смещения начала отрезка

**DWORD***dwFileOffsetHigh***,** // старшее слово смещения начала отрезка

**DWORD***nNumberOfBytesToLockLow***,** // младшее слово длины отрезка

**DWORD***nNumberOfBytesToLockHigh*// старшее слово длины отрезка

**);**

Параметры

*hFile*

[in] Дескриптор файла, в который производится запись. Должен быть открыт с доступомGENERIC\_READилиGENERIC\_WRITE(или обоими).

*dwFileOffsetLow*

[in] Определяет младшее слово смещения начала блокируемого отрезка относительно начала файла.

*dwFileOffsetHigh*

[in] Определяет старшее слово смещения начала блокируемого отрезка относительно начала файла. Не равно нулю, если смещение относительно начала более чем 232байт.

*nNumberOfBytesToLockLow*

[in] Определяет младшее слово длины блокируемого отрезка.

*nNumberOfBytesToLockHigh*

[in] Определяет старшее слово длины блокируемого отрезка. Не равно нулю, если длина отрезка более 232байт или 4 ГБ.

Возвращаемое значение

Если выполнение функции произошло успешно, то возвращается ненулевое значение.

Блокирование предоставляет процессу монопольный доступ к отрезку файла. Файловые блокировки не наследуются. Остальные процессы не могут ни читать, ни писать в заблокированную часть файла.

Функция UnlockFile позволяет разблокировать участок файла, ранее заблокированный функцией LockFile.

**BOOL UnlockFile(**

**HANDLE***hFile***,** // дескриптор файла

**DWORD***dwFileOffsetLow***,** // младшее слово смещения начала отрезка

**DWORD***dwFileOffsetHigh***,** // старшее слово смещения начала отрезка

**DWORD***nNumberOfBytesToUnlockLow***,** // младшее слово длины отрезка

**DWORD***nNumberOfBytesToUnlockHigh*// старшее слово длины отрезка

**);**

Параметры данной функции аналогичны параметрам функции LockFile.

Возвращаемое значение

Если выполнение функции произошло успешно, то возвращается ненулевое значение.

Отрезок файла, который разблокируется функцией UnlockFile должен в точности соответствовать отрезку, заблокированному функцией LockFile. Например, две соседних части файла не могут быть заблокированы по отдельности, а разблокированы как единое целое. Процесс не должен завершать выполнение, имея заблокированные части файлов. Файловый дескриптор, для которого есть заблокированные отрезки, не должен закрываться.

Для оптимального использования ресурсов операционной системы приложение должно закрывать ненужные файлы с помощью функции CloseHandle. Файлы, открытые на момент завершения приложения, закрываются автоматически.

**BOOLCloseHandle(**

**HANDLE***hObject*// Дескриптор объекта (файла, канала и пр.)

**);**

Возвращаемое значение

Если выполнение функции произошло успешно, то возвращается ненулевое значение.