**WriteFile**

Функция **WriteFile** пишет данные в файл с места, обозначенного указателем позиции в файле. Эта функция предназначена и для синхронной, и для асинхронной операции.

Функция **[WriteFileEx](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_writefileex.htm)** предназначена исключительно для асинхронной операции.

**Синтаксис**

|  |
| --- |
| **BOOL WriteFile(**  **HANDLE** *hFile***,** // дескриптор файла  **LPCVOID** *lpBuffer***,** // буфер данных  **DWORD** *nNumberOfBytesToWrite***,** // число байтов для записи  **LPDWORD** *lpNumberOfBytesWritten***,** // число записанных байтов  **LPOVERLAPPED** *lpOverlapped* // асинхронный буфер  **);** |

**Параметры**

***hFile***

[in] Дескриптор файла. Дескриптор файла, должен быть создан с правом доступа **GENERIC\_WRITE**. Для получения дополнительной информации, см. статью [**Защита файла и права доступа**](http://www.vsokovikov.narod.ru/New_MSDN_API/File_system/file_access_right.htm).

**Windows NT/2000/XP:**Для асинхронных операций записи, параметр ***hFile*** может быть любым дескриптором, открытым с флажком**FILE\_FLAG\_OVERLAPPED** функцией **[CreateFile](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_createfile.htm)**, или дескриптором сокета, возвращенным функцией **socket** или **accept**.

**Windows 95/98/Me:**Для асинхронных операций записи, параметр ***hFile*** может быть коммуникационным ресурсом, открытым с флажком **FILE\_FLAG\_OVERLAPPED** функцией **CreateFile**, или дескриптором сокета, возвращенным функцией **socket** или **accept**. Вы не можете исполнить асинхронные операции записи в почтовом ящике ядра **Windows**, в именованных каналах или дисковых файлах.

***lpBuffer***

[in] Указатель на буфер, содержащий данные, которые будут записаны в файл.

***nNumberOfBytesToWrite***

[in] Число байтов, которые будут записаны в файл.

Значение нуля определяет пустую операцию записи. Поведение пустой операции записи зависит от лежащей в основе файловой системы. Чтобы сократить или продлить файл, используйте функцию **[SetEndOfFile](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_setendoffile.htm)**.

Операции записи в именованном канале по всей сети ограничены 65 535 байтами.

***lpNumberOfBytesWritten***

[out] Указатель на переменную, которая получает число записанных байтов. Функция **WriteFile** устанавливает это значение в  нуль перед выполнением какой-либо работы или выявлением ошибок.

**Windows NT/2000/XP:** Если параметр ***lpOverlapped*** равен ПУСТО (NULL), то параметр ***lpNumberOfBytesWritten*** не может быть ПУСТО (NULL). Если ***lpOverlapped*** - не ПУСТО (NULL), ***lpNumberOfBytesWritten*** может быть ПУСТО (NULL). Если это асинхронная операция записи, Вы можете получить число записанных байтов при помощи вызова функции **GetOverlappedResult**. Если параметр***hFile*** связан с портом завершения ввода-вывода (**I/O**), Вы можете получить число записанных байтов при помощи вызова функции[**GetQueuedCompletionStatus**](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_getqueuedcompletionstatus.htm).

Если порты завершения ввода-вывода (**I/O**) используются, а Вы используете процедуру обратного вызова, чтобы освободить занимаемую память структурой [**OVERLAPPED**](http://www.vsokovikov.narod.ru/New_MSDN_API/Synchronization/str_overlapped.htm), на которую указывает параметр ***lpOverlapped***, установите  ПУСТО (NULL) как значение этого параметра, чтобы избежать проблемы порчи данных в памяти в ходе освобождения ресурса. Эта проблема порчи данных в памяти становится причиной неправильного числа байтов, которые возвращаются в этом параметре.

**Windows 95/98/Me:** Этот параметр не может быть ПУСТО (NULL).

***lpOverlapped***

[in] Указатель на структуру **OVERLAPPED**. Эта структура требуется тогда, если параметр ***hFile*** создавался с флажком**FILE\_FLAG\_OVERLAPPED**.

Если ***hFile*** был открыт с флажком **FILE\_FLAG\_OVERLAPPED**, у параметра ***lpOverlapped***  не должно быть значения ПУСТО (NULL). Он должен указывать на правильную структуру **OVERLAPPED**. Если ***hFile*** был открыт с флажком **FILE\_FLAG\_OVERLAPPED**, а***lpOverlapped*** имеет значение ПУСТО (NULL), функция может неправильно сообщить о завершении операции записи.

Если ***hFile*** был открыт с флажком **FILE\_FLAG\_OVERLAPPED**, а ***lpOverlapped*** имеет значение не ПУСТО (NULL), операция записи начинается при смещении, заданном в структуре **OVERLAPPED**, а **WriteFile** может возвратить значение прежде, чем операция записи будет закончена. В этом случае, **WriteFile** возвращает значение ЛОЖЬ (FALSE), а функция **[GetLastError](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm)** возвращает значение **ERROR\_IO\_PENDING**. Это дает возможность вызывающему процессу продолжать работу до тех пор, пока операция записи не закончится. После завершения операции записи, событие, определяемое в структуре **OVERLAPPED** устанавливается в сигнальное состояние. Вызывающая программа должна корректировать позицию указателя позиции в файле после завершения операции.

Если ***hFile*** не открывался с флажком  **FILE\_FLAG\_OVERLAPPED**, а ***lpOverlapped*** - значение ПУСТО (NULL), операция записи начинается с текущей позиции в файле и **WriteFile** не возвращает значения до тех пор, пока операция не будет закончена. Система после завершения операции модернизирует указатель позиции в файле.

Функция **WriteFile** сбрасывает событие, заданное членом ***hEvent*** структуры **OVERLAPPED** в несигнальное состояние, когда она начинает операцию ввода-вывода (**I/O**). Поэтому, нет какой-либо необходимости вызывающей программе проделывать эту процедуру.

**Windows NT/2000/XP:** Если ***hFile*** не был открыт с флажком **FILE\_FLAG\_OVERLAPPED** и ***lpOverlapped*** - не ПУСТО (NULL), то операция записи начнется со смещения, заданного в структуре  **OVERLAPPED,**а **WriteFile** не возвращает значение до тех пор, пока операция записи не завершится.

**Windows 95/98/Me:** Для операций с файлами, дисками, каналами или почтовыми ящиками в ядре **Windows**, этим параметром должно быть значение ПУСТО (NULL); указатель на структуру **OVERLAPPED** порождает вызов, который завершиться ошибкой. Тем не менее, **Windows 95/98/Me** поддерживает асинхронные операции на последовательных и параллельном портах.

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получить дополнительные сведения об ошибке, вызовите **[GetLastError](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm)**.

**Замечания**

Функция **WriteFile** может завершиться ошибкой  **ERROR\_INVALID\_USER\_BUFFER** или **ERROR\_NOT\_ENOUGH\_MEMORY** всякий раз, когда имеется слишком много ожидающих обработки асинхронных запросов ввода - вывода.

Чтобы отменить  все незаконченные асинхронные операции ввода-вывода (**I/O**), используйте функцию **[CancelIo](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_cancelio.htm)**. Эта функция  отменяет только операции, порождаемые вызывающим потоком для заданного дескриптора файла. Операции ввода-вывода (**I/O**), которые отменяются, заканчиваются ошибкой **ERROR\_OPERATION\_ABORTED**.

При записи в файл, последнее время записи полностью не обновляется, до тех пор, пока все дескрипторы, используемые для записи не будут закрыты. Поэтому, чтобы гарантировать точное последнее время записи, закройте дескриптор файла немедленно после записи файла.

Если часть файла блокируется другим процессом, а операция записи накладывается на заблокированную часть, функция **WriteFile**завершается ошибкой.

Доступ к буферу вывода данных в то время, когда операция записи использует буфер, может вести к порче данных, записанных из этого буфера. Приложения не должны записывать, перераспределять, или освобождать буфер вывода данных, который использует операция записи до тех пор, пока она не завершит работу.

Приложение должно выполнить некоторые требования при работе с файлами, открытыми с флажком **FILE\_FLAG\_NO\_BUFFERING**:

* Доступ  к файлу должен начаться при смещениях байтов в пределах файла, которые должны быть целыми числами кратными размеру сектора тома. Чтобы выяснить размер сектора тома, вызовите функцию **GetDiskFreeSpace**.
* Доступ  к файлу должен быть числом байтов, которое является целым числом кратным размеру сектора тома. Например, если размер сектора - 512 байтов, приложение может запросить для чтения и записи  512, 1024 или 2048 байтов, но не 335, 981 или 7171 байт.
* Адреса буферов для операций чтения и записи должны выравниваться кратно сектору (выравниваются адреса в памяти, которые являются целыми числами, кратными размеру сектора тома). Одним из способов выравнивания буфера кратно сектору, является использование функции **VirtualAlloc**, которая назначает буфера. Эта функция выделяет память, выравнивая их по адресам, которые являются целыми числами, кратными размеру страницы памяти  системы. Поскольку размеры и страницы памяти и  сектора тома являются степенью 2, память, выровненная кратно размеру страницы памяти системы, также выравнивается кратно и размеру сектора тома.

|  |
| --- |
| **Обратите внимание!** на то, что то, что отметки времени последнего обновления файла  не могут быть  правильно модифицированы для отдаленного файла. Чтобы гарантировать непротиворечивые результаты, используйте не буферизированный ввод - вывод (**I/O**). |

Система интерпретирует нулевые байты для записи, как определение пустой операции записи и **WriteFile** не обрезает и не продлевает файл. Чтобы обрезать или продлить файл, используйте функцию **SetEndOfFile**.

При записи в неблокированный, байтового режима дескриптор канала с недостаточным буферным пространством, **WriteFile**возвращает значение ИСТИНА (TRUE) с параметрами ***\*lpNumberOfBytesWritten*** < ***nNumberOfBytesToWrite***.

Когда приложение использует функцию **WriteFile**, чтобы записать в канал, операция записи не сможет закончиться, если буфер канала заполонен. Операция записи завершается тогда, когда операция чтения (использующая функцию **[ReadFile](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_readfile.htm)**) сделает доступным больше буферного пространства.

Если дескриптор анонимного  канала чтения был закрыт, а **WriteFile** пытается записывать, используя соответствующий дескриптора анонимного канала записи, функция возвращает значение ЛОЖЬ (FALSE), а **GetLastError** возвращает ошибку**ERROR\_BROKEN\_PIPE**.

Буквы или знаки могут быть записаны в экранный буфер, используя функцию  **WriteFile** с дескриптором консольного вывода данных. Консольный режим работы обуславливает правильное поведение функции. Данные записываются в текущей позиции курсора. Позиция курсора обновляется после операции записи.

Если Вы пытаетесь записывать в накопителе на гибких дисках, который не имеет дискеты, система показывает на экране окно сообщения, побуждающее пользователя повторить  операцию. Чтобы воспрепятствовать системе показывать это окно сообщения, вызовите функцию **SetErrorMode** с флажком **SEM\_NOOPENFILEERRORBOX**.

**Код примера**

<http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/creat_temp_fl.htm>

**Создание и использование временных файлов**

Приложения могут получать уникальные имена для временных файлов, используя функцию **[GetTempFileName](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_gettempfilename.htm)**. Функция же[**GetTempPath**](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_gettemppath.htm) извлекает путь к каталогу, где будут создаваться временные файлы.

В примере ниже  первый файл копируется в другой. Конечный файл - версия первого файла  буквами верхнего регистра.

Приложение открывает файл **Original.txt**, используя функцию **[CreateFile](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_createfile.htm).**Затем программа при помощи функции **GetTempFileName**выбирает  имя временного файла  и использует **CreateFile**, чтобы создать его. Приложение читает 4-х килобайтные блоки в буфер, преобразует содержание  буфера в прописные буквы и пишет преобразованный буфер во временный файл. Когда весь **Original.txt**запишется во временном файле, программа закрывает оба файла и переименовывает временный файл в **Allcaps.txt**, используя функцию **[MoveFile](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_movefile.htm)**.

|  |
| --- |
| *#include* <windows.h>  *#include* <stdio.h>  *#define* **BUFSIZE** **4096**  **int** **main**()  **{**  **HANDLE** hFile;  **HANDLE** hTempFile;  **DWORD** dwBytesRead, dwBytesWritten, dwBufSize=BUFSIZE;  **char** szTempName[**MAX\_PATH**];  **char** **buffer**[**BUFSIZE**];  **char** lpPathBuffer[**BUFSIZE**];    // Откроем существующий файл.    hFile = **CreateFile**("original.txt", // имя файла  **GENERIC\_READ**, // открыт для чтения  0, // совместного использования нет  NULL, // защита по умолчанию  **OPEN\_EXISTING**, // только существующий файл  **FILE\_ATTRIBUTE\_NORMAL**, // атрибуты обычного файла  NULL); // шаблона атрибутов нет  **if** (hFile **==** **INVALID\_HANDLE\_VALUE**)  **{**  **printf**("Could not open file.");  **return** 0;  **}**    // Получим временный путь  **GetTempPath**(dwBufSize, // длина буфера  lpPathBuffer); // буфер для пути  // Создадим временный файл.    **GetTempFileName**(lpPathBuffer, // каталог для временных файлов  "NEW", // префикс имени временного файла  0, // создаем уникальное имя  szTempName); // буфер для имени  hTempFile = **CreateFile**((**LPTSTR**) szTempName, // имя файла  **GENERIC\_READ** **|** **GENERIC\_WRITE**, // открыт для чтения и записи  0, // совместного использования нет  NULL, // защита по умолчанию  **CREATE\_ALWAYS**, // переписывать существующий файл  **FILE\_ATTRIBUTE\_NORMAL**, // атрибуты обычного файла  NULL); // шаблона атрибутов нет  **if** (hTempFile **==** **INVALID\_HANDLE\_VALUE**)  **{**  **printf**("Could not create temporary file.");  ret**u**rn 0;  **}**    // Читаем в буфер 4-х килобайтными блоками.  // Изменяем все символы в буфере на прописные.  // Записываем буфер во временный файл.    **do**  **{**  **if** (**ReadFile**(hFile, buffer, **4096**,  **&**dwBytesRead, NULL))  **{**  **CharUpperBuff**(buffer, dwBytesRead);    **WriteFile**(hTempFile, buffer, dwBytesRead,  **&**dwBytesWritten, NULL);  **}**  } **while** (dwBytesRead **==** BUF**S**IZE);    // Закрываем оба файла.    **CloseHandle**(hFile);  **CloseHandle**(hTempFile);    // Перемещаем временный файл в новый текстовый файл.    **if** **(!MoveFileEx**(szTempName, "allcaps.txt", **MOVEFILE\_REPLACE\_EXISTING**))  **{**  **printf**("Could not move temp file.");  **return** 0;  **}**  **}** |