Файловые системы

Работа с файлами в Windows API

Соответствие Win32 API и UNIX

Win32 API	UNIX	Описание
CreateFile	open	создать новый файл или открыть существующий
DeleteFile	unlink	удалить существующий файл
CloseHandle	close	закрыть файл
ReadFile	read	прочитать данные из файла
WriteFile	write	записать данные в файл
SetFilePointer	lseek	установить указатель работы с файлом
GetFileAttributes	stat	вернуть аттритбуты файла
LockFile	fcntl	заблокировать регион файла для решения взаимного исключения
UnlockFile	fcntl	разблокировать регион файла

Соответствие Win32 API и UNIX

Win32 API	UNIX	Описание
CreateDirectory	mkdir	создать новый каталог
RemoveDirectory	rmdir	удалить пустой каталог
FindFirstFile	opendir	выполнить инициализацию для начала чтения записей каталога
FindNextFile	readdir	прочитать следующую запись каталога
MoveFile	rename	перенести файл из одного каталога в другой
SetCurrentDirectory	chdir	изменить текущий каталог

Работа с каталогами и файлами

Функция	Выполняемое действие
GetCurrentDirectory	Получение текущего каталога
SetCurrentDirectory	Смена текущего каталога
GetSystemDirectory	Получение системного каталога
GetWindowsDirectory	Получение основного каталога системы
CreateDirectory	Создание каталога
RemoveDirectory	Удаление каталога
CopyFile	Копирование файла
MoveFile MoveFileEx	Перемещение или переименование файла
DeleteFile	Удаление файла

Работа с томами

- Для выяснения того, какие логические диски существуют в системе, используется функция

  DWORD GetLogicalDrives(void)

  Каждый установленный бит возвращаемого значения соответствует существующему в системе логическому устройству. Например, если в системе существуют диски A:, C: и D:, то возвращаемое функцией значение равно 13(десятичное).
- Функция

  DWORD GetLogicalDrivesStrings(DWORD cchBuffer, LPTSTR lpszBuffer)

  заполняет lpszBuffer информацией о корневом каталоге каждого логического диска в системе. В приведенном выше примере буфер будет заполнен символами

  A:\<null>C:\<null>D:\<null><null>

  параметр cchBuffer определяет длину буфера. Функция возвращает реальную длину буфера, необходимую для размещения всей информации.

Работа с томами

- Для определения типа диска предназначена функция
  UINT GetDriveType(LPTSTR lpszRootPathName)
  В качестве параметра ей передается символическое имя корневого каталога (напр. **A:**), а возвращаемое значение может быть одно из следующих:

Идентификатор	Описание
0	Тип устройства определить нельзя
1	Корневой каталог не существует
DRIVE_REMOVABLE	Гибкий диск
DRIVE_FIXED	Жесткий диск
DRIVE_REMOTE	Сетевой диск
DRIVE_CDROM	Компакт диск
DRIVE_RAMDISK	RAM диск

Работа с томами

- Для получения подробной информации о носителе используется функция **GetVolumeInformation**. Она заполняет параметры информацией об имени тома, названии файловой структуры, максимальной длине имени файла, дополнительных атрибутах тома, специфических для файловой структуры.
- Функция **GetDiskFreeSpace** сообщает информацию о размерах сектора и кластера и о наличии свободных кластеров.

Создание и открытие файла

```
HANDLE CreateFile (
    LPCTSTR lpFileName, // pointer to name of the file
    DWORD dwDesiredAccess,       // access (read-write) mode
    DWORD dwShareMode,      // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                    // pointer to security descriptor
    DWORD dwCreationDistribution,// how to create
    DWORD dwFlagsAndAttributes, // file attributes
    HANDLE hTemplateFile       // handle to file with attributes to copy
);
```

В случае удачи функция **CreateFile** возвращает описатель открытого файла как объекта ядра. Существенно, что в противном случае она возвращает не NULL, а INVALID_HANDLE_VALUE.

Параметры *CreateFile ()*

- Параметр **dwDesiredAccess** задает тип доступа к файлу. Можно определить флаги GENERIC_READ и GENERIC_WRITE а так же их комбинацию для разрешения чтения или записи в файл.
- Параметр **dwShareMode** определяет режим совместного использования файла различными процессами. Если этот параметр равен нулю, то никакой другой поток не сможет открыть этот же файл. Флаги FILE_SHARE_READ и FILE_SHARE_WRITE а так же их комбинация разрешают другим потокам осуществлять доступ к файлу для чтения или записи.
- Параметр **dwCreationDistribution** определяет действия функции в зависимости от того, существует ли уже файл с указанным именем.
  - CREATE_NEW - Создает файл, если файл существует, то ошибка.
  - CREATE_ALWAYS - Создает файл , если файл существует, то старый файл удаляется и новый создается.
  - OPEN_EXISTING - Открывает существующий файл.
  - OPEN_ALWAYS - Создает файл, если файл не существует, то создается новый файл.
  - TRUNCATE_EXISTING - Открывает файл и урезает его до нулевой длины

Параметры *CreateFile ()*

- Параметр **dwFlagsAndAttributes** определяет атрибуты файла, если он создается и задает режим работы с файлом.
  - FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY
  - Атрибуты файла могут комбинироваться за исключением FILE_ATTRIBUTE_NORMAL, который всегда используется в одиночестве.
  - Вместе с атрибутами могут комбинироваться и флаги, задающие режим работы с файлом.
  - FILE_FLAG_NO_BUFFERING - Не осуществлять кэширование и опережающее чтение
  - FILE_FLAG_RANDOM_ACCESS - Кэшировать как файл произвольного доступа
  - FILE_FLAG_SEQUENTIAL_SCAN - Кэшировать как файл последовательного доступа
  - FILE_FLAG_WRITE_TROUGH - Не буферизовать операцию записи. Производить запись на диск немедленно.
  - FILE_FLAG_DELETE_ON_CLOSE - Уничтожить файл при закрытии. Полезно комбинировать с атрибутом FILE_ATTRIBUTE_TEMPORARY.
  - FILE_FLAG_OVERLAPPED - Работа с файлом будет осуществляться асинхронно.

Синхронный и асинхронный ввод/вывод

- При синхронной работе с файлами прикладная программа, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).
- При асинхронной работе с файлами прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения а продолжает исполняться.

Функции файлового ввод-вывода

```
BOOL ReadFile(
        HANDLE hFile,     // handle of file to read
        LPVOID lpBuffer,  // address of buffer that receives data
        DWORD nNumberOfBytesToRead,// number of bytes to read
        LPDWORD lpNumberOfBytesRead,// address of number of bytes read
        LPOVERLAPPED lpOverlapped // address of structure needed for
                                  // overlapped I/O
);
BOOL WriteFile(
        HANDLE hFile,     // handle to file to write to
        LPCVOID lpBuffer,      // pointer to data to write to file
        DWORD nNumberOfBytesToWrite, // number of bytes to write
        LPDWORD lpNumberOfBytesRead,// pointer to number of bytes written
        LPOVERLAPPED lpOverlapped // address of structure needed for
                                  //overlapped I/O
);
```

Параметры функция файлового ввода-вывода

- Параметры функции *ReadFile ()* имеют следующее предназначение:
  - hFile – описатель объекта ядра “файл”, полученный в результате вызова функции CreateFile
  - LpBuffer – адрес буфера, в который будет производиться чтение
  - nNumberOfBytesToRead – количество байт, которые необходимо прочитать
  - lpNumberOfBytesRead – адрес переменной, в которой будет размещено количество реально прочитанных байт. Существенно, что сразу после выполнения функции ReadFile, этот параметр не может быть установлен, так как операция чтения только началась.
  - lpOverlapped – указатель на структуру OVERLAPPED, управляющую асинхронным вводом выводом.
- Параметры функции *WriteFile ()* аналогичны параметрам функции *ReadFile ()*.

Пример синхронного копирования файла

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL,
     OPEN_EXISTING, 0, NULL);
outhandle = CreateFile ("newf", GENERIC_WRITE, 0, NULL,
     CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
       s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
       if (s && count > 0) WriteFile(outhandle, buffer, count, Socnt, NULL);
} while (s>0 && count>0);

/* Close the files. */
CloseHandle (inhandle):
CloseHandle (outhandle);
```

Асинхронный ввод-вывод

- Для организации асинхронной работы с файлами необходимо при вызове функции *CreateFile ()* установить флаг FILE_FLAG_OVERLAPPED в параметре dwFlagsAndAttributes.
- После этого функции *ReadFile ()* и *WriteFile ()* будут работать асинхронно, т.е. только запускать операции ввода вывода и не ожидать их завершения.
- В отличие от синхронных операций, при организации асинхронного чтения (записи) необходимо явно указать позицию, начиная с которой производится операция. Это связано с тем, что текущей позиции не существует, так как несколько операций чтения и записи могут производиться одновременно с разных позиций в одном файле.

Асинхронный ввод-вывод

```
typedef struct _OVERLAPPED {
    DWORD Internal;     //Используется операционной системой.
                        //Хранит статус завершения операции.
    DWORD InternalHigh;    //Используется ОС. Хранит
                        //количество переданных байт.
    DWORD Offset;       //Позиция в файле, начиная с которой
                        //необходимо производить операцию
                        //чтения (записи).
    DWORD OffsetHigh;//Количество байт для передачи.
    HANDLE hEvent;      //Описатель события, которое произойдет
                        //при завершении операции чтения
          //(записи).
} OVERLAPPED;
```

Вариант 1 организации асинхронного ввода-вывода

- Перед запуском операции создается объект ядра “событие” и его описатель передается в функцию *ReadFile ()* или *WriteFile ()* в качестве элемента hEvent параметра lpOverlapped.
- Программа, выполнив необходимые действия одновременно с операцией передачи данных, вызывает одну из функций ожидания (например, WaitForSingleObject), передавая ей в качестве параметра описатель события.
- Выполнение программы при этом приостанавливается до завершения операции ввода-вывода.

Вариант 2 организации асинхронного ввода-вывода

- Событие не создается. В качестве ожидаемого объекта выступает сам файл. Его описатель передается в функцию *WaitForSingleObject ()*.
- Этот метод прост и корректен, но не позволяет производить параллельно несколько операций ввода-вывода с одним и тем же файлом.

Вариант 3 организации асинхронного ввода-вывода

- “Тревожный” асинхронный ввод-вывод. Схема построена на использовании функций *ReadFileEx ()* и *WriteFileEx ()*. В качестве дополнительного параметра в эти функции передается адрес функции завершения, которая будет вызываться всякий раз при завершении операции ввода-вывода.
- Существенно, что эти функции выполняются в том же самом потоке что и функции файлового ввода/вывода. Это значит, что поток, запустивший операции чтения записи должен обратиться к функции ожидания, чтобы разрешить системе вызвать функцию завершения.