

File system emulation

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2004-12/0283.html>

From: Beni Falk (*BeniFalk_at_discussions.microsoft.com*)

Date: 12/06/04

Date: Mon, 6 Dec 2004 09:39:08 -0800

We are designing a special-purpose embedded recorder. The storage media is removable and will be connected to a PC via an IDE or SCSI interface (note – several such media can be connected to the PC at the same time).

The recorder implements a proprietary file system, which is totally unlike the standard MS file systems (i.e. FAT32, NTFS, etc.). There are many good reasons for that and it is not going to change.

The "directory" structure of our proprietary file system is fixed (it contains 2 levels of directories and at most 100 files total). Individual files can be quite large – may exceed 4 GB.

We are capable to access our media (performing raw sector read/write accesses) – no problems there.

We need to provide to the user the ability to perform standard file operations on our media, using standard Windows utilities (e.g. Windows Explorer). This is important from a marketing perspective.

Ideally, we would like the removable media to behave something like an M-Systems Disk On Key (i.e. once the media is inserted, the user will see a new drive and should be able to invoke Explorer operations on it).

We understood that this may be difficult to accomplish. Instead, we were advised (in the course of a Microsoft seminar) to perform the following:

- a) Once sensed that the media is inserted, our software automatically creates a directory (on a normal hard disk). It reads the contents of our media and creates small dummy files in the created directory that "mirror" the files on our media. This directory is removed by our software upon sensing that our media is removed from the PC.
- b) We install a filter driver beneath the file system (between partmgr.sys and disk.sys). Then, when a user-invoked application (e.g. Windows Explorer) performs some action on one of the dummy files, our driver intercepts the IRP (MJ_READ), performs the operation vis-a-vis our media and completes the IRP vis-a-vis the I/O manager. As far as the user is concerned, it seems as if

the operation was performed vis-a-vis the actual file on the media rather than vis-a-vis the dummy file.

1. We tried doing that and found out that instead of logical file offsets, our driver receives physical disk offsets, which is not actually surprising, but it is not good for us – obviously we need to know the logical file offset which the user-invoked application tries to access.

2. Also, we found that it is necessary to make the dummy file as large as the actual file on the media in order to achieve usable results. We achieved this by using SetFilePointer on the dummy file. On a FAT32 disk that takes a significant amount of time for a large (multi-gigabyte) file. On NTFS that does not take time, however when subsequently trying to access the dummy file via Explorer, nothing happened (no requests were received by our driver).

Even if ignoring the question of the time it takes to perform SetFilePointer, this approach also requires user's PC to have a large amount of spare storage on his disk, which is problematic from marketing point of view.

Also, on FAT32, we were not able to create a file larger than 4 GB, which is not good enough for our application.

Can you suggest anything?

Thank you for your help.

Beni Falk