Inside the High Performance File System

Part 0: Preface

Written by Dan Bridges

Introduction

I am not a programmer's backside but I am an enthusiast interested in finding out more about HPFS. There is so little detailed information available on HPFS that I think you will find this modest series instructive. The REXX programs to be presented are functional but they are not particularly pleasing in an aesthetic sense. However they do ferret out information and will help you to understand what is going on. I'm sure that a programming guru, once motivated, could come up with superior versions. Hopefully they will. This installment originally appeared at the OS2Zone web site (http://www.os2zone.aus.net).

I've been asked [by someone else. Ed.] to write a preface to this series. Normally I prefer to write on little-covered topics whereas much of what I'm going to discuss in this installment often appears in a cursory examination of the HPFS. The trouble with most of what has been written about HPFS in books on OS/2 is that the topic is never considered very deeply. After finishing working your way through this series (still being written on a monthly basis, but expected to occupy eight parts including this one) you will have a detailed knowledge of the structures of the HPFS. Having said that, there is a place for some initial information for readers who currently know very little about the subject.

File Systems

A File System (FS) is a combination of hardware and software that enables the storage and retrieval of information on removable (floppy disk, tape, CD) and non-removable (HD) media. The File Allocation Table FS (FAT) is used by DOS. It is also built into OS/2. Now FAT appeared back in the days of DOS v1 in 1981 and was designed with a backward glance to CP/M. A hierarchical directory structure arrived with DOS v2 to support the XT's 10 MB HD. OS/2 v1.x used straight FAT. OS/2 v2.x and later provide "Super FAT". This uses the same layout of information on the storage medium (e.g. a floppy written under OS/2 v2 can easily be read by a DOS system) but adds performance improvements to the software used to transfer the data. Super FAT will be covered in Part 1.

FAT

Figure 1 shows the layout of a FAT volume. There are two copies of the FAT. These should be identical. This may seem like a safety feature but it only works in the case of physical corruption (if a bad sector develops in one of the sectors in a FAT, the other one is automatically used instead) not for logical

corruption. So if the FS gets confused and the two copies are not the same there is no easy way to determine which copy is still O K.

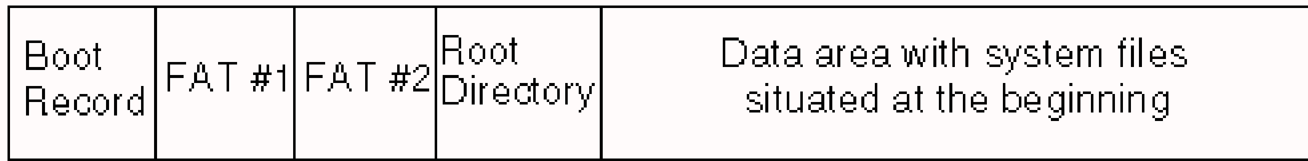

Figure 1: The layout of a volume formatted with the FAT file system. Note: this diagram is not to scale. The data area is quite large in practice.

The root directory is made a fixed known size because the system files are placed immediately after it. The known location for the initial system files enables DOS or OS/2 to commence loading itself. (The boot record, which loads first off, is small and only has enough space for code to find the initial system files at a known location.) However this design decision also limits the number of files that can be listed in the root directory of a FAT volume.

Entries in the root directory and in subdirectories are not ordered so searching for a particular file can take some time, particularly if there are many files in a directory.

The FAT and the root directory are positioned at the beginning of the volume (on a disk this is typically on the outside). These entries are read often, particularly in a multitasking environment, requiring a lot of relatively slow (in CPU terms) head movement.

How Files are Stored on a FAT Volume

Files are stored on a FAT volume using the FS' minimum allocation unit, the cluster (1-64 sectors). A 32-byte directory entry only provides sufficient space for a 8.3 filename, file attributes, last alteration date/time, filesize and the starting cluster. See Figure 2.

Data	Size
Filename	8
Extension	3
File Attributes	1
Reserved Space	10
File Timestamp	2
File Datestamp	2
First File Cluster Number	2
Filesize	4

Figure 2: The layout of the 32 bytes in a directory entry in a FAT system.

The corresponding initial cluster entry in the FAT then points to the next FAT entry for the second cluster of the file (assuming that the file was big enough) which in turn points to the next cluster and so on. FAT entries can be 16-bit (max. FFFFh) or 12-bit (max. FFFh) in size, with volumes less than 16 MB using the 12-bit scheme. FAT entries can be of four types:

- Contain 0000h if the cluster is free (available);
- Contain the number of the next cluster in the chain;
- If this is the last cluster in the chain then the FAT entry will consist of a character which signifies the end of the chain (EOF);
- Another special character if the cluster of the disk is bad (unreliable).

The FAT FS is prone to fragmentation (i.e. a file's clusters are not in one, contiguous chain) in a single-tasking environment because the FAT is searched sequentially for the next free entry in the FAT when a file is written, regardless of how much needs to be written. The situation is even worse in a multitasking environment because you can have more than one writing operation in progress at the same time. See Figures 3 and 4 for an example of a fragmented file under FAT.

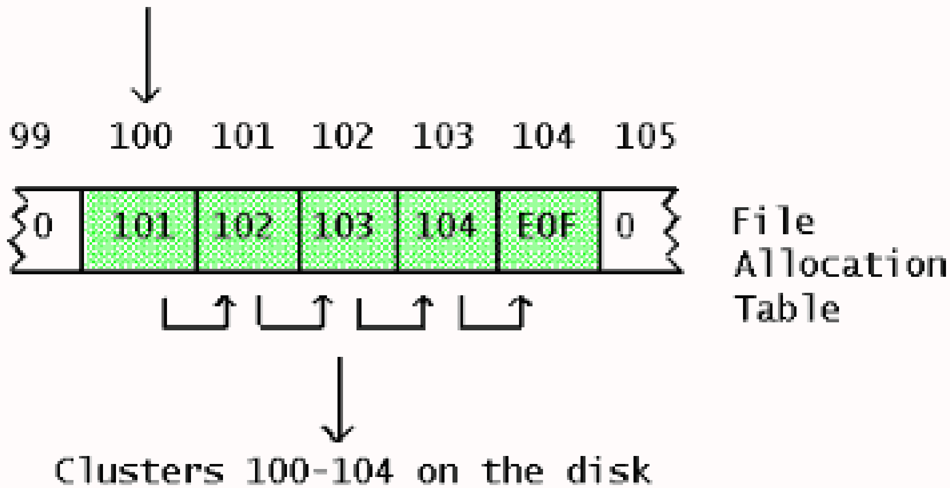

Figure 3: The layout of a contiguous file in the FAT.

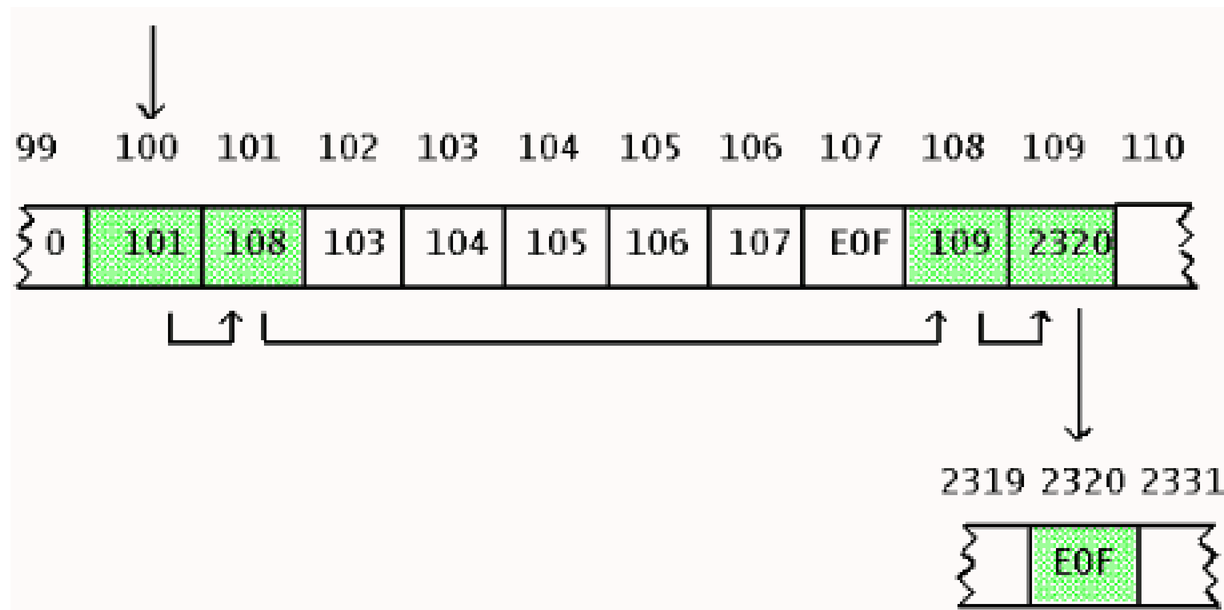

Figure 4: An example of a fragmented file under FAT in three pieces.

The FAT FS uses a singly-linked scheme i.e. the FAT entry points only to the next cluster. If, for some reason, the chain is accidentally broken (the next cluster value is corrupted) then there is no information in the isolated next cluster to indicate what it was previously connected to. So the FAT FS, while relatively simple, is also rather vulnerable.

FAT was designed in the days of small disk size and today it really shows its age. The maximum number of entries (clusters) in a 16-bit FAT is just under 64K (due to technical reasons, the actual maximum is 65,518). Since we can't increase the number of clusters past this limit, a large volume

requires the use of large cluster sizes. So, for example, a volume in the 1-2 GB range has 32 KB clusters. Now a cluster is the minimum allocation unit so a 1 byte file on such a volume would consume 32 KB of space, a 33 KB file would consume 64 KB and so on. A rough assumption you can make is that, on average, half a cluster of space is wasted per file. You can run CHKDSK on a FAT volume, note the total number of files and also the allocation unit size and then multiply these two figures together and divide the result by 2 to get some idea of the wastage. The situation is quite different with HPFS as you will see when you read Part 1.

Finally, FAT under OS/2 supports Extended Attributes (EAs - up to 64 KB of extra information associated with a file), but since there is very little extra space in a 32-byte directory entry it is only possible to store a pointer into an external file with all EAs on a volume being stored in this file ("EA DATA. SF"). In general it is fair to state that EAs are tacked on to FAT. With HPFS the integration is much better. If the EA is small enough HPFS stores it completely within the file's FNODE (every file and directory has an FNODE). Otherwise EAs is stored outside the file but closely associated with it and usually situated physically close to the file for performance reasons. Some users have occasionally reported crosslinking of EAs under FAT. This can be quite a serious matter requiring reinstallation of the operating system. I've not heard of this occurring under HPFS. Note that the WorkPlace Shell relies heavily on EAs.

HPFS

HPFS is example of a class of file systems known as Installable File Systems (IFS). Other types of IFS include CD support (CDFS), Network File System (NFS), Toronto Virtual File System (TVFS - combines FS elements of VM, namely CMS search path, with elements of UNIX, namely symbolic link), EXT2-OS (read Linux EXT2FS partitions under OS/2) and HPFS386 (with IBM LAN Server Advanced).

An IFS is installed at start-up time. The software to access the actual device is specified as a device driver (usually BASEDEV=xxxxx.DMD/.ADD) while a Dynamic Link Library (DLL) is load to control the format/layout of the data (with IFS=xxxxx.IFS). OS/2 can run more than one IFS at a time so you could, for example, copy from a CD to a HPFS volume in one session while reading a floppy disk (FAT) in another session.

HPFS has many advantages over FAT: Long Filename (254 characters including spaces); excellent performance when directories containing many files; designed to be fault tolerant; fragmentation resistant; space efficient with large partitions; works well in a multitasking environment. These topics will be explored in the series.

REXX

One of the many benefits of using OS/2 is that it comes with REXX (providing you install it - it requires

very little extra space). REXX is a surprisingly versatile and powerful scripting language and there are oodles of REXX programs and add-ons available, much of it for free. This series presents REXX programs that access HPFS structures and decode their contents.

Conclusion

In this installment you have seen that the FAT FS has a number of problems related to its ancient origins. HPFS comes from a fresh design with one eye on likely advances in storage that would occur in the foreseeable future and the other eye on obtaining good performance. In the next installment we look at the many techniques HPFS uses to achieve its better performance.