Designing a Write-Once File System

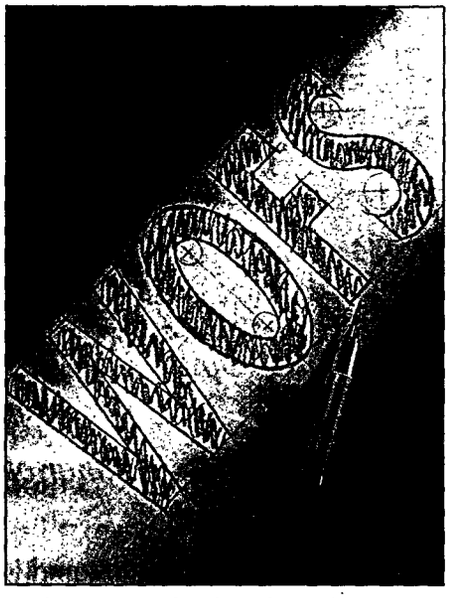
A general-purpose optical storage software technology

Simson Garfinkel

Write-once storage systems are one of three different optical storage technologies to emerge over the last decade. Write-once systems (called WORMs, for Write Once, Read Multiple) are the oldest of the optical systems and in many ways the most successful to date.

Once a block is written into WORM, it can't be changed. This unique characteristic of WORM is at once its virtue and its handicap: WORM offers data permanence, but traditional computer systems can't use WORM without special software. Operating systems like MS-DOS and Unix, for example, need to be able to update the blocks used to store directories when files are created or deleted.

In 1985 I started a research project at the MIT Media Lab to develop a file system designed for write-once devices. This article describes the design and evolution of this software component, known as the Write-Once File System (WOFS).



Why WORM?

Comparing WORM disks with read-only CD-ROMs and rewritable magneto-optical systems, we find that each optical storage technology has advantages and disadvantages.

CD-ROM, with more than 500 Mbytes on a 4.77-inch disk that costs less than a dollar to manufacture, is an ideal system for publishing databases and distributing large software systems. But CD-ROM can't be written, and thus isn't a replacement for conventional storage.

Rewritable magneto-optical disks are much slower than magnetic disks, and don't hold as much as CD-ROM, or even magnetic disks of the same size. Although rewritable cartridges are removable, they cannot easily serve as a publication format, because each disk must be individually recorded, rather than being mass-produced.

WORM cartridges can store two to four times more data than similarly-sized rewritable ones. Today, 5.25-inch drives are available that can store more than 500 Mbytes per side, and 14-inch write-once systems can store two to four gigabytes. As stated earlier, the unique virtue of write-once technology is the permanence it offers for valuable data. In many application areas, such as financial and medical applications, this feature is extremely critical.

WORM in Use

There are three approaches to using write-once technology in computer applications. In the first, a specialized application uses the optical disk for storing large datafiles (such as images). The application may track these files using its own dedicated routines, or use a database management system for this purpose.

The second approach is to use a special device driver that lets the write-once disk simulate a rewritable disk. When the operating system tries to rewrite a block, the driver writes a new block and remembers the translation.

This article focuses on the third approach: A file system designed specifically for use with write-once optical disks. The goal of our WOFS project was twofold: To invent a file system standard which defined the means of arranging information into files on the optical disk; and to create a file system implementation — a function library writ-

Simson is the principal scientist at N/Hance Systems, which sells a variety of products based upon WOFS. He can be reached at 52 1/2 Pleasant, Cambridge, MA 02139, or Internet mail as simsong@next.cambridge.ma.us.

(continued from page 78)

ten in the C programming language which would let us test the design.

From the beginning, we had high hopes for our project. We designed the file system so that it would be applicable to both read-only and rewritable optical disks, in addition to write-once. (Our system was designed one year before the High Sierra CD-ROM standard.) We chose not to take advantage of special features that were present in the drives made by certain manufacturers, so that the software would work with any drive. (See the accompanying text box entitled "The Problem with Post Fields.") Most importantly, we designed the file system to be operating system independent, so that files written to a disk with one operating system could be read back with another.

Over the past five years, WOFS has undergone three major redesigns, most recently in November 1989. Today it is a full-fledged file system that provides many of the function calls specified by the POSIX standard, including *open( )*, *close( )*, *read( )*, and *write( )*. WOFS has been ported to both MS-DOS and Unix; optical cartridges can be moved between the two systems, and files from one operating system can be shared with the other, even if the two operating systems are running on processors that use different byte orders. WOFS also allows the user to access previous versions of files, as well as take the entire disk "into the past."

Differences from Conventional Systems

Magnetic file systems update the data stored in files by rewriting the blocks that the data is stored in. Files are deleted from directories by rewriting the directory with the file's name missing, and returning the blocks associated with the file to the pool of unused blocks.

When working with write-once devices, however, things are more complicated. Because the physical blocks on a write-once disk cannot be changed, files and directories that are logically changed must be rewritten to new locations. The role of the write-once file system is to keep track of the most recent version of each directory and file on the disk and permit them to be found quickly. A good write-once file system also minimizes the number of blocks that have to be written for any given operation.

Making WOFS CPU-independent was tricky because different microprocessors store data in different ways

WOFS does not store information on the optical disk in MS-DOS, Unix, Macintosh, or any other "standard" format. WOFS uses its own format, instead; an operating system specific interface allows existing operating systems to read and write files on the optical.

Making WOFS CPU-independent was tricky, however, because different kinds of microprocessors store data in different ways. The Intel 80286 microprocessor, for example, stores 16-bit integer values on 2-byte boundaries; the Motorola 68020 processor stores 16-bit integers on 4-byte boundaries. Different microprocessors use different strategies for sign extension when converting from 16-bit values to 32-bit values.

To get around these problems, WOFS stores all directories and other file system information with a small set of predefined structures which were specifically designed to be portable and readable by many different kinds of microcomputers. The only two data types in the structures are 32-bit unsigned numbers and null-terminated character strings. Strings are always padded to a multiple of 4 bytes. WOFS solves word-alignment problems by storing all data on 4-byte boundaries. WOFS also has a mechanism for detecting and swapping byte order when necessary.

Basic Structures

WOFS records new data and updates to the optical file system as a series of sequential block-write operations, starting with the first block on the disk and continuing until the end.

The WOFS approach divides the disk into two discrete regions: One in which blocks have been recorded and one in which they are blank. The transition point between the two regions is called the "last written block."

By not having a specific part of the disk dedicated to directories or file pointers (such as Unix *inodes*), WOFS eliminates a problem that is common with other file systems: Part of the disk fills up, making the disk unable to hold more information, while other parts of the disk remain empty.

When a disk is mounted, WOFS finds the last written block with a binary search: If the block examined contains valid data, WOFS searches higher on the disk for the last written block; if the block does not contain data, WOFS searches lower on the disk.

Although a binary search across the disk requires much "seeking" (movement of the optical head), each seek is

The Problem With Post Fields

In the early days of write-once, some manufacturers tried to add special features to their drives to overcome the difficulty of using write-once with traditional operating systems. One popular system was called "post fields," small records at the end of each block of data that could be recorded independently from the block itself. Post fields were used as pointers to newer versions of blocks. If the operating system had to rewrite block 500, the new data might be written in block 567 and the post field of block 500 would be set to "567."

There are two primary difficulties with post fields: speed and reliability. The more often a block is modified with a post field-based file system, the longer it takes to find that block. This is a big problem with blocks that are used to store directories, for if the directory is modified 100 times, then 100 blocks and post fields have to be read in order to find the most recent version.

A second problem with post fields arises because of the inherent unreliability of write-once disks. WORM disks frequently contain blocks with bad bytes in them. WORM drives get around this problem by reading back every block after it is written and rewriting it in another location if required. The problem with a post field based-scheme is that occasionally it is the bytes in the post field itself that are bad, which makes it impossible to make the post field point at the rewritten data.

— S.G.

(continued from page 80)
precisely half the distance of the previous seek. If there are *N* blocks on the disk, the total number of seeks is guaranteed to be *LOG2(N)*. In practice, WOFS takes less than four seconds to find the last written block on a 5.25-inch optical disk drive.

Transactions and Filemaps

WOFS writes to the optical disk in groups of operations called *transactions*. A transaction might be creating a file, changing its data, or erasing a file or directory; it also might consist of many operations grouped together. At the end of each transaction, WOFS writes a special block called the *End of Transaction Block*, or EOT.

When a disk is mounted, WOFS locates the last written block on the drive and checks to see if it is an EOT. If it isn't, then the last transaction was interrupted by a hardware failure (for example, a loss of power or a pulled cable); WOFS then searches backwards on the disk, block-by-block, to find the last complete transaction.

Every EOT contains a pointer to a block (or blocks, if necessary) that stores a list of current directories. The first directory in the list is the root directory.

Additional fields in the EOT are used to maintain the disk volume name and accounting information, such as the time of the last transaction. The EOT also has a pointer to the previous EOT, which is used for stepping the disk into the past.

WOFS uses the same basic structure to remember where both files and directories are stored. We call it a fragment table, but it can be thought of as a list of regions on the disk where data is located and pointers that tell the file system how to assemble the data into a continuous file.

When a file that has been written is closed (or when a modified directory entry is written to the disk), WOFS writes two additional blocks to the disk for that file: a "filemap," and a "file header."

The filemap is the database that tells WOFS where on the disk the data in the logical file is actually written. Each record in the database has three fields:

- Ordinal (the byte position in the logical file where this data fragment starts)
- Length (the number of bytes in this data fragment)
- Starting block (the point on the disk where the data fragment is located)

Because WOFS writes most files contiguously on the optical disk, most files are represented by a single fragment. For example, a file consisting of 65,536 bytes starting at block 1000 might have a filemap that looked like that in Figure 1.

The filemap allows programs that update records within a file (such as database programs) to be used with the optical disk. If a program were to reopen file AAA, as in Figure 1, and rewrite the first 1024 bytes of the file with new information, the new filemap might look like that shown in Figure 2 when the file is closed (assuming a disk-block size of 512 bytes).

If this file is then opened for reading, WOFS will return to the calling program the 1024 bytes starting at block 1140, followed by the 64,512 bytes that start at block 1024.

Of course, the filemap for version 1 of file AAA is still on the write-once disk; the user can still access the original version of the file by reading the file with the first filemap instead of the second. A special function allows the calling program to find out which version of the file it is reading. Another special function is provided which can step individual files — or an entire optical cartridge — backwards in time.

At the heart of WOFS is a set of state machines that manipulate filemaps and transfer information to and from the optical disk. These state machines provide many standard file-system functions, as well as some that are only possible because of the WOFS filemaps. These functions include the following:

- *filestr_seek*
- *filestr_read*
- *filestr_write*
- *filestr_insert*
- *filestr_delete*
- *filestr_check*

The *filestr_seek* function repositions the file pointer in much the same way as *lseek( )*. Likewise, *filestr_read* works like *read( )*, and *filestr_write* is like *write( )*. The function *filestr_insert* has no equivalent in other file systems. This function opens up the file and inserts bytes at the current position within the file, making the file longer. A converse function, *filestr_delete*, also has no standard equivalent, and deletes bytes at the current position, making the file smaller.

WOFS is a file system specifically designed for write-once optical disks, yet it can be used with rewritable systems

Finally, *filestr_check* provides a consistency check, by checking the variables used by the state machines for self-consistency. Frequently, use of this function made it relatively easy to find and isolate programming errors during the file system's development.

The Insert and Delete functions are used primarily to maintain directories; by keeping the directory entries in sorted order, WOFS reduces the average time needed to create a file by a factor of two.

The state machines are optimized to transfer data directly between the optical disk interface and user memory when more than a complete block of data is transferred. Thus, WOFS often performs reads and writes of large files at the maximum possible transfer rate of the hardware.

Directory Entries

All of the operations that file systems perform can be broken down into four main categories:

- Looking up a name in a directory
- Making an entry in a directory (for a file or a subdirectory)
- Removing an entry in a directory
- Transferring data between an open file (or directory) and the operating systems

To resolve a filename, WOFS starts at the root directory and searches for the names of each successive subdirectory, one-by-one. Eventually, WOFS determines the location on the disk of the file's containing directory. These translations are cached, which substantially

File AAA Version 1 filemap

Ordinal	Start Block #	Length
0	1000	65536

Figure 1: *A sample file in its initial state*

File AAA Version 2 filemap

Ordinal	Start Block #	Length	
0	1140	1024	← new data
1024	1002	64512	← original data

Figure 2: *Sample file in subsequent state*

improves the file system's performance.

Like the data in files, each WOFS subdirectory is stored in one or more fragments. The actual directory entries are built out of one or more variable-length records. The contents of a directory entry are shown in Figure 3.

The first four bytes of the directory record are the directory entry flag, 0x00001000. If WOFS reads 0x00100000 instead of 0x00001000, all of the binary values stored in the directory entry are byte-swapped. This might happen if the directory entry was written by a VAX and read back by a SPARC microprocessor — the VAX and the SPARC store the same binary value in the reverse order. By detecting byte-swaps and swapping the data back, WOFS allows cartridges written with one byte order to be read back on a system that uses another.

WOFS only swaps on read. If WOFS running on a SPARC updates a directory written by WOFS running on a VAX, it does not swap the binary values back before writing. Chances are that data written by one microprocessor will be read back by that same microprocessor; if the directory entry is eventually read again by a VAX, the VAX will swap the data when it reads it.

Directory entries are created with the *filestr_insert( )* function; likewise, they are deleted with the *filestr_delete( )* function. A function called *wofsfile_readdir( )* reads directory entries and translates them into a form appropriate for Unix, MS-DOS, or the Macintosh operating system.

When a file is opened for reading or writing, that file's fragment table is loaded into memory and a state machine is set up to handle data transfer. When the file is closed, the fragment table is written back to the optical disk, followed by a file header.

Individual Files

Every WOFS file has a file header. File headers contain all of the information in the directory entry — allowing the directory entry to be reconstructed in the event of media damage — as well as additional information used by operating systems other than MS-DOS. File headers can be extended to include security-related information such as access control lists used by newer, security-conscious operating systems. Normally, however, the file header is used only by WOFS and remains invisible to the user.

The file header fits within a single WORM block. The information it contains is shown in Figure 4.

File System Blocks

EOTs, File Headers, and Fragment Tables are all stored on the optical disk within a special kind of data structure called a *File System Block* (FSB). The FSB has a special 12-byte header that makes it possible to identify the block in the event of a media failure, which allows for quick and reliable recovery of the data on the WORM disk.

The FSB header contains a 4-byte FSB flag, a 4-byte "type" field which indicates whether the FSB holds an EOT, a file header or a fragment table, and a 4-byte self-referential pointer that contains the disk block address where the FSB was actually written. In C, the FSB has the structure shown in Figure 5.

In the event of media failure, the user can scan the entire optical car-

- directory entry flag (0x00001000, used for byte-swap detection)
- size of directory entry
- file type
- modification time
- file version number
- x,y location (for Macintosh OS)
- file header location
- file length in bytes
- filename

Figure 3: *Contents of a directory entry*

tridge with a special recovery program that searches for FSBs that hold file headers. The entire directory hierarchy can then be automatically recovered. In practice, this procedure takes less than 20 minutes for a 400-Mbyte cartridge.

Identifying a file-system block by both self-referential pointer and 4-byte flag minimizes the chance that a random block of data will appear to be an FSB.

WOFS also uses the FSB to allow implementations running on computers with one byte order to use files written by implementations using another byte order. If the WOFS function that reads the EOT discovers that the byte order of the FSB flag is reversed, it reverses the byte order of every 4-byte integer value stored in the EOT. Like the byte-swapping for directory entries, this swapping happens only on read operations. When the EOT is written to the disk at the conclusion of the next transaction, it is written in the new byte order. The functions which read file headers and fragment tables swap similarly when necessary.

- File header version number
- File number
- File type
- File version number
- Location of previous version
- Directory number of containing directory
- Filename (without directory name prefix)
- Time of last write
- Time of file creation
- Number of bytes in file
- *x,y* (for Macintosh OS)
- Location on disk of fragment table
- Number of fragments in fragment table
- Name and version of operating system that created file
- Name of site that created file

Figure 4: Contents of the file header block

The Operating System Interface

Although it is possible to link an application program directly with the WOFS function library, most developers choose to access write-once optical disks through the operating system. Having the WORM disk behave like a regular magnetic drive lets developers develop and test and their programs with conventional hard disks, before they go to the expense of purchasing an optical subsystem.

Presently, WOFS runs transparently under MS-DOS and Unix. The MS-DOS implementation uses a terminate-and-stay-resident (TSR) program that intercepts all uses of software interrupt 0x21. Each software interrupt is examined to see if it should be handled by WOFS or by MS-DOS, and then the appropriate functions are called. This is the same technique used by a variety of network systems available for MS-DOS.

One significant problem we encountered in writing the TSR was that many

```
typedef struct {
        u_long  flag;
        u_long  location;
        u_long  type;
    } fs_hdr;

typedef struct {
        fs_hdr  hdr;
        char    data[ BLOCK_SIZE - sizeof(fs_hdr) ];
    } fs_block;
```

Figure 5: FSB structure declarations

DOS application programs use MS-DOS functions that are undocumented. Along the way, we learned that many application programs currently on the market — such as WordPerfect — use the File Control Block (FCB) disk I/O access mechanism left over from MS-DOS Version 1.0. This is despite the fact that ever since MS-DOS Version 2.0, Microsoft has declared these functions obsolete and asked developers to avoid using them. The problem is that it is very difficult to properly implement all of the subtleties of the FCB; it took several months of trial and testing before we stopped finding problems with our implementation.

One problem that remains with WOFS is memory consumption. The WOFS core requires approximately 50 Kbytes of code and another 40 Kbytes of data space. These memory constraints are modest under every operating system environment with the exception of MS-DOS. Unfortunately, MS-DOS is a very large part of today's write-once market. We are currently working an EMM-version of WOFS which will lower the MS-DOS low-memory requirements from 90 Kbytes to less than 50 Kbytes.

People who have the Unix operating system can use WOFS with a server we have developed for Sun Microsystems' Network File System. A workstation on a local area network simply mounts the WOFS disk the same way it would any other remote file system. The server program translates all network requests into the appropriate WOFS operations on the optical drive. The WOFS NFS server makes WOFS available to an extremely large base of users.

A WOFS interface for the Macintosh operating system is currently under development. One of the biggest expected users is the graphics arts community, which will be able to use WOFS as an alternative to local area networks for transporting large image files between computers running different operating systems.

The Evolution of the Design

The design of WOFS has undergone several iterations. The original system was developed in 1985 as a research file system for experimenting with basic concepts. This was used at the MIT Media Lab for work with read-only and write-once compact disks. Although we used that version to master a number of CD-ROMs, we never successfully integrated it into a running operating system. The limitation of the first WOFS was that files could only be stored contiguously, and directories had to be buffered in memory in their entirety.

The first major rewrite introduced the idea of fragmented files, which allowed files and directories to be updated a piece at a time. The second rewrite eliminated dynamic allocation of memory inside the WOFS core, which was necessary to allow the WOFS to interface with the kernels of real operating systems, such as MS-DOS and Unix. The third rewrite replaced all 2-byte integers stored in the file system structures with 4-byte integers, to facilitate adopting WOFS to computer architectures such as the 68000, which can store integers only on arbitrary 4-byte boundaries.

WOFS's feature set has remained basically unchanged for the past five years. From the beginning, the system had to provide all of the basic Unix functions for file and directory access with off-the-shelf write-once optical disks.

The biggest surprise in developing WOFS was that it was much harder to achieve our original design goals than we ever anticipated. It took four years of thinking about the byte-swapping problem before we realized that we could support heterogeneous byte orders on the same disk — to the point of switching byte orders on successive directory entries. Likewise, it was difficult to develop a set of file structures that would work reliably and efficiently under a variety of different operating criteria; every time we thought we had solved the problem, we discovered a way to improve the structures that would make them either more efficient, more portable, or interpretable with less lines of code.

We were not alone in encountering these difficulties. Indeed, one of the reasons that write-once optical drives have failed to catch on is the dismal state of most write-once software.

The Rewritable Future

Perhaps the industry's current excitement about rewritable optical disks comes from a belief that finally, here is a high-density mass storage system that can be used unmodified with existing operating systems.

Unfortunately, in jumping on the rewritable bandwagon, I believe that we will be losing the very characteristic that attracted me to write-once in the first place: data permanence. I have always been comforted by the idea of being able to undelete any file that I have previously deleted. You can do that with WOFS. You can't do that with Unix or MS-DOS file systems running on rewritable optical disks.

Nevertheless, not all data needs to be permanently archived. Furthermore, high performance and price of rewritable disks is now improving at a greater rate than write-once due to market pressure. For this reason I am now in the process of designing an improved version of WOFS, "WOFS 2," which will work interchangeably with write-once and rewritable optical disks.

Why use a WOFS 2 with a rewritable optical disk when you can just as well use rewritable optical disks with native Unix or MS-DOS file systems? There are two reasons: portability and speed. WOFS 2 will retain WOFS's ability to move disks between operating systems and CPU types. And because it will be specifically optimized for the performance characteristics of optical disks, it will be faster and more reliable when used with optical disks than file systems developed for the hard-disk technologies of the 1970s.

Write-once compact disks, the media that WOFS was designed to be used with, may be several years in the future. But WOFS exists today and is in productive use.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**