Chapter 6

FILE SYSTEM MANAGEMENT

In this Chapter

÷ Introduction
÷ Directories and their Names
÷ File System Objects
÷ File System Functions
÷ Information Types
÷ File System Architecture

`

Introduction

Data Storage is a very necessary requirement of a computer system. A computer stores data permanently in form of files. *A file is a collection of data on a permanent storage medium.* This medium stores files for longer period of times.

A file system is an abstract system provided by the OS to handle files. It contains files and other objects which make the use of files convenient. I/O devices are also considered to be files. When an input device sends input, it is like reading a file. When an output device gives output, it is like saving data in a file.

Directories and their Names

A directory is also called a folder. It is a way to group and organize different file system objects.

A folder is a structure which contains names of other file system objects. But to the user, a directory behaves in a way as it contains other objects.

In some file systems, an object can have more than one name or is nameless. On some systems an object has a single name. When there is one name for every object then whenever we use this name, the object is referenced. But with multiple names supported, the name and object must be maintained differently. Each name must be mapped to the correct object.

The sequence of directories travelled to access a file or other object determine the full path or full name. To make the full path, we start from the root directory and name of every subsequent directory is joined by a suitable separator.

UNIX uses '/' and windows and DOS uses '\' as separator for file name.

Current Directory

The directory, where the actions of a process will be applied.

Relative Pathname

It is generated by joining current directory name before the relative name.

The full pathname has ‘/’ or ‘\’ in its beginning in DOS, windows and UNIX.

So, if the file is *readme.txt*, and directory is *snake*, then ‘readme.txt’ is just like *‘/snake/readme.txt’*.

If the OS supports multiple names for an object, then ‘/snake/readme.txt’ and ‘/snake/user_stuff/readme.txt’ could be the same object.

Partitions

To store files, disk is used. But in some cases RAM is also used to store files. This area on RAM is called RAM disk and it stores temporary and small files which need fast access. But the main device to store a file is secondary storage. Hard disk is the most popular of them.

Partition is a virtual device to store files. In windows and DOS, for the pathname of a file, the partition name is also included in it. For example, if in DOS or Windows, we write ‘D:\Games\snake\readme.txt’ then it means that readme.txt file is in games\snake directory, which is in D: partition.

In UNIX and some other OS, the partitions are mounted. Mounting means they are attached. Here we mean that these OS define a single **unified file system name space**. And all partitions are attached to it. They are also referenced using the unified file system name space.

Namespace is a domain/container in which an identifier is unique in representing a single object.[1]

In other words, namespace is a container from which we choose names for files. We use it to clarify the names even if they are same but refer to different objects under different directories. *A namespace is a collection of identifiers and names*.

While mounting partitions, the OS will make one partition the root partition. Root means the start point of the unified file system namespace. Now other file systems within any directory can be divided within that unified namespace. The partition’s namespace now becomes a child of unified file system namespace, under the mount point.

[1] https://spaces.internet2.edu/display/macepaccman/MACE-paccman-glossary

While a partition is being mounted, the OS takes care of the objects in that directory. Most OS do not mount a directory if it contains objects. A directory must be empty before mounting.

Some OS allow a non-empty directory to be mounted. But the contents are hidden.

It is also possible to mount a single partition multiple times. As a result, an object in such a partition will get more than one name.

Root Directories for Processes

OS gives a process a root directory of its own. This root directory can be a child of some other directory, but the process takes this as the head of file system structure. No other super folder is accessible by the process. It adds security to the file structure as the process can only interact with the objects within that directory.

The same policy is also applied to the users of a computer system. Now the user can use only its root directory. All processes executed by the user run under this directory.

When we host a website on a webserver, after buying some space on it, we actually get a directory in it, and that directory contains every file we upload for that website. So it remains in its own directory. Webservers, FTP servers and email servers all use such schemes.

Structuring Directories

Many types of structures are present to place objects in a directory. These are:

1. Single level tree
2. Two level tree
3. Multilevel tree
4. Acyclic graph
5. General graph

Single Level Tree

It is the structure which contains only one folder/directory. This directory cannot contain name of any other directory. Names of files and other non-directory file system objects can be placed in it.

It is a very simple structure. But it is very limited in its benefits. It is good for only those situations when the file objects are too few and we do not need to organize them in multiple folders.

Two Level Tree

In two level trees, one directory becomes the root. Only root can contain names of other directories. The children cannot contain name of another directory but only names of files and other non-directory objects.

Multilevel Tree

This level allows any directory to contain name of any other object. This can be another directory or non-directory objects.

Graph

A single object can have multiple names. Then every name points to the same object. In this case, the structure is not a tree. We call it a graph.

This situation makes the structure complex. But if we restrict that only the non-directory objects will have multiple names then complexity is reduced. This modified graph makes the sharing of objects in different directories much easier.

If we allow directories to have multiple names, then it gives us a more usable and flexible structure. Suppose a situation where many users are working together on a single task. They are using same data which is organized in a directory. Any user can share it and add a name of its own to its working directory to represent that directory. When files are updated and modified, the changes are visible to all users in their respective working directories.

But all this adds complexity. If there was only one name for an object, then we never worry about which object was actually considered. But with multiple names, we must track all names which refer to the same object. We cannot do this by comparing the names. Additional metadata must be maintained to answer such questions.

Similarly, if a particular object is deleted; many additional actions must be performed.

Does the OS need to provide additional and separate commands to delete the names when an object is deleted? Or the deletion of an object will also delete them?

Is it necessary to delete all names related to an object and then delete object? Or we can delete the object directly leaving, the target-less names.

How to know that all the names of an object have been deleted?

On UNIX, when you delete a name, the object is not deleted. But on every deletion of name, the total number of remaining names is checked. If the name being deleted is the last name, then the object is also deleted.

Cycles

A cycle is a situation when an object has name of second object and the second object has name of the first. Thus, these two objects may produce an infinite number of names for each other. Every time an object references the other, a new name is produced. And depending upon the references, many names will be produced.

Acyclic Graphs

Acyclic Graphs provide a way to limit the names in cycles. But the available methods to find if a link will produce a graph are very slow. So in practical systems, the graphs of every type are avoided.

UNIX controls all this complexity and still provides multiple names. In UNIX, non-directory object may get multiple names. But only OS can give multiple names for directories.

When a directory is created, two names are automatically created for it, ● (dot)

and ●● (double dot).

●represents the directory and ●● represents its parent. They produce cycles but they are efficiently managed.

If we write any one of the following, then all of them represent the same object:

1. dir_x/../dir_x/file_a

2. dir_x/./file_a
3. dir_x/file_a
4. ./dir_x/./file_a

Directory Entries

To properly handle files, many types of data are maintained. These data items are maintained as directory entries.

These entries store the following information:

1. Information about ownership
2. Location
3. Size
4. Access rights
5. Creation date and time
6. Last modification date and time

A file is first opened then an operation is performed on it. Opening means to establish a pointer to the file's actual location. Using this pointer all operations can be applied to the file. So we won't need the file name. The pointer gives a convenient way to performs operations on the file.

On some systems, a separate file is used to sore information about the original file. UNIX uses this scheme. *Such a file containing information about the other file is called inode.* The real directory entry of the file contains the file name and number of its inode.

File System Objects

A file system contains files and other objects. The other objects are used to ease the management of files or other purposes. Following is the list of file system objects:

1. Shortcut
2. Device
3. Pipe
4. Shared memory

5. Semaphore

Shortcut

Shortcut is a file which contains address of the other file folder. We often call it a softlink. When a shortcut is opened, the address in it is accessed and the file or folder associated with it is opened.

Sometimes it happens that the associated file is deleted and softlink remains. Such a softlink is termed as dangling pointer.

Device

An I/O device is also considered a file. When an input device receives input, it is like reading a file. When an output device gives output, it is like data being saved in the file.

Pipe

We have discussed them in chapter 3. They are basically created between parents and their children for communication. They may or may not have a name. So they actually transfer the data between processes and their children.

A process attached to a pipe writes data in the pipe. The pipe keeps data until it is received by the receiving process. The capacity of the pipe is limited. If the capacity is full then the sending process is blocked.

Shared Memory

It is actually a portion in RAM for communication between the processes. It is maintained by the File system of the OS.

Semaphores

It is an object. It is used by the OS to synchronize processes.

File System Functions

The OS must provide some way to interact and manipulate different File System objects. All these functions are provided at the abstract level. In a single directory structure, following functions provide a powerful set of actions:

1. Create

2. Delete
3. Read
4. Write

create *Name*

It is used to create an object. The *name* is the name of the new object.

delete *Name*

It is used to erase an object whose name is *name*.

read Name, location

From the *location* in the object *name* it reads one byte of data.

Write Name, Location, Byte

It saves *byte* to the *location* in the *name* object. As a result, the size of the file will change.

Information Types

An OS stores data about information types in a file. It means that OS knows which file contains which type of data. Files can have many types of data. For example:

1. Source codes
2. Binary data
3. ASCII data
4. Image data
5. Executable files
6. Database contents

The OS uses this information to process a file according to its contents. For example, if an image file is sent to a printer to print then it is ok. But if an image file is loaded into the compiler for execution then it should be denied. The OS sees the requested operation and denies it if it is not appropriate.

There are three methods available to recognize the data in a file:

1. Extensions in file name
2. Magic number
3. OS supported files

Extensions in File Names

This is the most popular and straight forward method. Every type of data is given a short name. This short name is called extension. For example, an ASCII file which contains plain text has an extension *.txt*. When a program creates such a file, a *.txt* is automatically appended after its name. Next time when the file is accessed, the OS will check the extension and load the file in the correct program. Other extensions include:

1.	.ppt	for	Power Point
2.	.mdb	for	Acess
3.	.doc	for	Word
4.	.exe/.com	for	executable
5.	.wp	for	word perfect

DOS and many other OS allow any file to have any extension. But once the extension is given, only the actions associated with that extension can be performed.

For example, even an executable can have *.wp* extension. But the DOS will not execute it. This restriction means that DOS will only attempt to execute those files which have *.exe* or *.com* extensions.

Obviously, if someone gives *.exe* extension to a text file, then DOS will attempt to execute it, an inappropriate thing will occur.

Magic Number

This method looks for a magic byte or collection of bytes in a file at a set location. The specific value in this byte[s] determines the type of contents in the file. Mostly these bytes are placed in the start of the file. For example, GIF files use this. The value 'GIF8' is placed in the beginning of the file. So it is identified.

UNIX recognizes executable files by this technique. If an executable does not contain a magic number, it is not executed by UNIX.

Mostly this technique is used by the application programs. OS does not use them much.

OS Supported Files

In this method, the OS keeps information about a file type along with other information. When a file is created, then its type is also recognized and saved.

Mac OS from Apple use this technique heavily. They store file type and associated program. When a file is clicked, all other information and file type is used and the associated program is discovered and used to open this file.

File System Architecture

File System is mostly managed in user and OS levels.

User level

File: It is the collection of records. For example, all records about students are stored in one file.

Records: records contain logically related information. For example, record of Student 10. A record is a unit of an I/O operation. Records contain fields. OS can define fixed sized or variable sized or both types of records.

Field: It is the base element of data. For example name of a student.

OS level

For the OS, the file is a related collection of **logical blocks**. The logical blocks make a file. Logical blocks are stored in physical blocks.

Physical blocks are found on the storage medium. A medium, e.g. disk is a collection of physical blocks. A physical block contains a logical block and administrative information. They are actually storage units. The disk I/O and file system buffers use blocks as there I/O units. Block size is a multiple of I/O unit of data transfer set by the disk driver.

Access Methods

There are two access methods:

1. Sequential access method
2. Direct access method

Sequential Access Method

In sequential access method, the records in a file are accessed in ascending number of their location in the file. The OS maintains a pointer for the records in the file. After accessing a record, the pointer is incremented and moved to the next record. This method is quite slow. Using this method, we do not need to specify the location while requesting a read or write operation, because the pointer automatically moves to the next location in the file.

Direct Access Method

With direct access method, the speed of access improves. The OS is able to access any record from within the file directly. There are two methods to perform direct access:

1. The location of record is passed to the OS as a parameter while requesting an operation.
2. To specify the location with seek before actual read or write is performed.

On database systems and many OS, these two access methods are used to develop more sophisticated access methods. Such a method is called indexed access method.

Indexed Access Method

When this method is used, the records contain two fields. First filed contains the actual data in the field, while second contain a number called index. To specify the record, this index is passed as a parameter. Then the OS finds the record with matching index.

Access methods can offer synchronous or asynchronous access for both read and write operations.

Synchronous Read/Write Operations:

In this method, the process requesting the operation is blocked until the operation is completed.

Mostly synchronous read/write is used. Its use is simple for the programmers.

Asynchronous Read/Write Operations

In this method, after starting the operation, the control is shifted back to the process. The process can continue its execution without worrying about the I/O operation. The process is notified when the I/O is completed. To notify the process following options are available:

1. Signal the process
2. Change the value of a variable in the process
3. Provide an SVC to the process to check the status of its started operation.

The output operations may not have a notification system.

With the passage of time, the I/O devices have improved in speed and they can perform quite fast. Therefore, asynchronous read/write is seldom used.

Access Control

A human or an application can access a file. We call them file user. Access control means to control and restrict:

1. Which user can access the file
2. How can the file be used by an authorized user
3. What type of operations can be performed by an authorized user.

One method is to give all the users unlimited access to every object. Obviously, it is simple but in-efficient. DOS uses this type of access controlling method. A user who wants to restrict some files does it by physical means. E.g. keeping the machine locked.

Second method is to control the operations performed by a user on a file. The operations which an OS would control are:

Read: a user can open and read a file.

Write: a user can add new information to the file and overwrite the old information.

Append: a user can add new records at the end of file.

Delete: a user can remove a file. It will release disk memory for reuse.

List: a user can see the names contained in the directory.

Execute: a user can load a program file in RAM and run a process on its basis.

Change Access: to change the file access rights of a user.

One challenging thing here is how to make a decision about which user will have a right on which file. We normally use the identity of the user and associate rights to it. The file access rights of a user are stored in a list named *access list*. *Access Lists contains list of operations a user is allowed to perform on a file.*

When unlimited access is granted, permissions set for each user is maintained independently.

When OS handles the rights and controls them, the access list contains high volume of data.

Grouping Permissions

This technique reduces the overall data needed to maintain permissions. Access permissions are grouped together. The grouping occurs on the basis of similar users or files.

User Groups

Users with similar rights and permissions make a group. The files are associated with the group. The users in the group can perform similar actions on that file. Using this technique, we don't need to store access information for each user.

Similarly, all files in a directory can be allocated to a similar set of permissions.

Using Password

Before performing an operation on a file, the user is asked to provide a password. These passwords are provided for that operation. This password is not used to login. We can also group the passwords and reduce information to be saved and remembered. For example, all files in a folder may require same password to read data from them.

Capabilities

The rights of a process are called capabilities. While using capabilities, the rights are given to the processes not the files. So when a process accesses a file and attempts to perform an action, its capabilities are analyzed to know whether it can perform the requested operation on the file or not. This method is not common.

Locking Files

File locking means that when a process is using a file, no other process can access it. We may call it *mutual exclusive access.*

There are three different considerations while locking a file:

1. Partial or complete
2. Operation Type
3. Advisory or mandatory

Partial or Complete

The file can be completely locked or only the part can be locked which is in use by a process. Other parts will remain accessible by other processes. Complete locking is easy to implement. But partial locking is complex.

Operation Type

There are two types of operations on files: read and write. And for both operations locks are available.

Read lock: If a file is locked for reading by a process, then newly coming read requests from other processes can be granted. But the file cannot be written.

Write Lock: If a file is locked by a process for writing, then both read and write requests by other processes are denied.

Mandatory vs. Advisory Locking

Mandatory Locking: It does not grant access to a file when a process has locked it.

Advisory Locking: The locking mechanism provides the status of the lock. The access will be denied when an accessing process reads the lock status and accepts it.

Implicit Locking

It means that while performing an operation, the target file automatically enters the appropriate lock. For example: when on UNIX a file is being executed, access to it is denied.

Building Blocks of Files/Blocking

On a disk, the unit for I/O is a block. I/O subsystem either reads or writes a block at a time.

Size: The block size is multiple of sector size. Sector size is usually 512B so the block size can be 1024B, 1536B etc.

File: A file is simply a collection of storage units. These units are called blocks.

Blocking Method: This method decides how the file will be stored in blocks. It is of 3 types:

1. Fixed Blocking
2. Unspanned blocking
3. Spanned blocking

Fixed Blocking

This is suited for those files in which record size is fixed.

Every block contains a fixed an integral number of records. A record must not be larger than the block. If the block size is integral multiple of record size, then it will be fully utilized. Else there will be some wasted space at its end.

If the block size and record number are provided, then the OS can easily find the record from its block.

Unspanned blocking

This is suited when the record size is variable.

Many records may be contained in a block, but a record cannot go beyond one block. The size of record must not exceed the block size.

If the next record to store is greater than the space available in the block, than he space at the end of the block will be wasted.

To find the location of a record, the sizes of all records before it must be provided.

Spanned Blocking

It is also suited for files with variable sized blocks.

The record size can be any. It may exceed the size of block. A record can be stored on more than one block. No space is wasted within the block except possibly the last one.

To find the location of a record, the size of all records before it must be provided.

Allocation

Allocation means how the disk space is allocated to a file when it created and updated. This scheme decides how a logical block of a file will be mapped to a physical block on disk. The size of physical block is a multiple of sector size. Sector size is usually 512B. so the physical block can be from 512B to 4096B. It means a physical block can have 1 to 8 sectors.

There are 3 methods to allocate space to a file:

1. Contiguous allocation
2. Linked Allocation
3. Indexed Allocation

Contiguous Allocation

It is the simplest one. All the logical blocks of a file are stored consecutively. So a partition or consecutive group of physical blocks is allocated to a file. In the file system, a directory entry stores the start address of the file on the disk and its size. The OS can find any byte if it's number in the file [offset] and start address of the file on disk is known. The offset is added to the start address and the address of the byte is generated.

In this scheme, the OS must allocate space to a file when it is created. From time to time new data will be added and the file will expand. The OS needs to do something for expansion of a file. If the size of file after expansion is greater than the space allocated by the OS and there are other files on the disk after this file, than those file must be moved to some other location on the disk and then the file can expand. The movement of files adds considerable overhead.

Also, allocation of space to files at the time of creation causes unused space to be created at the end of partitions. This effect is called checker boarding. The partitions must be selected for a file according to their sizes. For this we need some algorithms. These algorithms are called partition selection algorithms. (Discussed in Ch 4, Memory Management)

Unused space is called external fragmentation or holes and many holes develop across the disk. Compaction/garbage collection is performed to collect and combine holes to get a large free space. But it is very slow and complex because disks are slow devices. Therefore the compaction is seldom used. (Discussed in Ch 4, Memory Management)

Linked Allocation

In linked allocation, a file's parts can be separated and stored on disk noncontiguously. The size of physical block is kept larger than the logical block. After storing a logical block in a physical block, a pointer is also stored which contains address of the next physical block of the file. In the directory entry, we need to store the address of the first block. Then subsequent blocks can be found using these pointers. If the directory entry also contains the last address, then the append operation can be performed easily.

External fragmentation does not occur so there is no need for compaction.

Indexed Allocation

In indexed allocation, the sizes of physical and logical blocks are kept the same. One physical block contains one logical block. But the information about which logical block is in which physical block is maintained in an index table. All links are stored in the index table contiguously. In the directory entry, either the full index table or a pointer to it is stored.

Since index table needs to store addresses of every block, it produces large volume of data. To minimize the storage needs, multilevel indexing is applied. But it increases the entries for a file. In multilevel indexing, first level table contains links to lower level index tables. When the location of a logical block is to be calculated, first level table is consulted and a link to lower level table is found. Then using this entry either the physical block is found or the next level is searched.

Maintaining Information about Free Space on Storage Medium

The OS must know which physical blocks are not yet allocated to any file. When a request to save some contents of a file arrives, the OS can use this information and store the file on some free physical blocks*. A list which contains information about free blocks is called a free list.* There are many methods to maintain a free list.

1. Bitmap
2. Free list array

Bitmap

A physical block can have only two states, it is in use or it is not in use. So, we can represent these states as a single bit. Now, there will be a bit for every physical block. We call it **free list bitmap.**

This method uses less space as only a single bit is needed to store state of a physical block. All bits are organized in the form of words. The only disadvantage is the extra processing required to calculate the state of a block from the bits in the appropriate word.

Free List Array

This is quite an efficient way to manage used and unused blocks. An array is used to record status of the blocks. Every physical block is represented as an element in the array. If its element is in the array, then it is free. If it is not in the array, then it is in use.

As files are created, expanded, shrunk and deleted, the representing elements are added or removed from the array.

Linked List

In this method, the first free block contains a pointer to the second free block. The second contains a pointer to the third free block, and so on.

This scheme uses memory very efficiently as the information about the holes is stored in them. No extra space is wasted for this purpose.

Managing Groups of Free Holes

We can manage the full contiguous free physical blocks as a group. We will need to store information about groups, not the individual blocks. Such a formation is called *free holes list*.

Such a list will store the start address of each hole/group and number of blocks in it. This scheme uses more bits to maintain records about holes than bitmap. But the number of elements is reduced. This scheme is really useful when we need large number of free blocks in contiguous allocation.

Linked Holes List

In this arrangement, the first hole/group of free blocks contains start address of the second hole, second contains start address of the third and so on. Each hole also contains number of blocks in the next hole.