M.M.E.S. WOMEN'S ARTS AND SCIENCE COLLEGE

(Affiliated to Thiruvalluvar University)

HAKEEM NAGAR - MELVISHARAM- 632 509

Walaja Taluk, Arcot Block Phone:04172 266167, 266463

DEPARTMENT OF COMPUTER SCIENCE

E-NOTES

OPERATING SYSTEM

PREPARED BY

Mrs. M.SILAMBARSI, M.Sc., M.Phil.,

HEAD, DEPARTMENT OF COMPUTER SCIENCE & INFORMATION SYSTEM MANAGEMENT

Mrs. R. SANGEETHA, M.C.A., M.Phil.,

ASSISTANT PROFESSOR, DEPARTMENT OF COMPUTER SCIENCE

OPERATING SYSTEM

Objective: Enable the student to get sufficient knowledge on various system resources.

CONTENTS

File Management

Text Books:

1. Abraham Silberschatz Peter B. Galvin, G. Gagne, "Operating System Concepts", Sixth Edition, Addison Wesley Publishing Co., 2003.
2. Operating Systems: Design and Implementation - Albert S. Woodhull and Andrew S. Tanenbaum

File Management

1.1 Introduction

- A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.
- A file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.
- Many different types of information may be stored in a file – Source Programs, Object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.

1.2 Basic Concepts of File

- A file has a certain defined structure according to its type.
- A **text file** is a sequence of characters organized into lines (and possibly pages).
- A **Source file** is a sequence of sub routines and functions, each of which is further organized as declarations followed by executable statements.
- An **Object file** sequence of bytes organized into blocks understandable by the systems linker.
- An **executable file** is series of code sections that the loader can bring into memory and execute.

1.2.1 File Attributes

A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters such as ***example.c.*** A file has certain attributes, which vary from one operating system to another consists of,

1. **Name:** The symbolic file name is the only information kept in human readable form.

2. **Identification:** This unique tag usually, a number, identifies the file within the file system; it is the non-human-readable name for the file.

3. **Type:** This information is needed for those systems that support different types.

4. **Location:** This information is a pointer to a device and to a device and to the location of the file on that device.

5. **Size:** The current size of the file (in bytes, words, or blocks), and possibly the maximum allowed size are included in this attribute.

6. **Protection:** Access – control information determines who can do reading, writing, executing and so on.

7. **Time, date, and user identification:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

The information about all files is kept in the directory structure that also resides on secondary storage. The directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes. It may take more than a kilobyte to record this information for each file.

1.2.2 File Operations

A file is an abstract data type. The operating system can provide system calls to create, write, read, repositioning, delete and truncate files.

1. **Creating a file:** To create a file two steps are necessary,
    - Space in the file system must be found for the file.
    - An entry for the new file must be made in the directory, the directory entry records the name of the file and the location in the file system, and possibly other information
2. **Writing a file:** To write a file

- Make a system call specifying both the name of the file and the information to be written to the file.
- While giving the name of the file, the system searches the directory to find the location of the file.
- The system must keep *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

3. **Reading a file:** To read from a file
   - Use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.
   - Again, the directory is searched for the associated directory entry, and the system needs to keep a ***read*** pointer to the location in the file where the next read is to takes place.
   - Once the read has taken place, the read pointer is updated. A given process is usually only reading or writing a given file, and the current operation location is kept as a per – process **current – file – position pointer.** Both the read and write operations use this same pointer, saving space and reducing the system complexity.
4. **Repositioning within a file**
   - The directory is searched for the appropriate entry, and the current –file- position is set to a given value.
   - Repositioning within a file does not need to involve any actual I/O. This file operation is also known as a ***file seek***.
5. **Deleting a file:** To delete a file,
   - We Search the directory for the named file, when found the associated directory entry, then release all file space, so that it can be reused by other files, and erase the directory entry.
6. **Truncating a file**
   - The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and the recreate it, this function allows all attributes to

remain unchanged except for the file length, but lets the file set to length zero and its file space released.

These six basic operations comprise the minimal set of required file operations. Other common operations include ***appending*** new information to the end of an existing file and ***renaming*** an existing file. These primitive operations can then be combined to perform other file operations.

Other common operations include ***appending*** new information to the end of an existing file and ***renaming*** an existing file. Most of the file operations mentioned involve searching the directory fort the entry associated with the named file. To avoid this constant searching, many systems require that an open() system call be made before a file is first used actively. The operating system keeps a small table, called the **open-file table**, containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is ***closed*** by the process, and the operating system removes its entry from the open-file table. Create and delete are system calls that work with closed rather than open files. Several pieces of information are associated with an open file. They are,

1. **File pointer**

   On systems that do not include a file offset as part of the read () and write () system calls, the system must track the last read – write location as a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.

2. **File-open count**

   As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open counter tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.

3. **Disk location of the file**

Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

4. Access right

Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests. Some operating systems provide facilities for locking an open file (or sections of a file). File locks allow one process to lock a file and prevent other processes from gaining access to it. File locks are useful for files that are shared by several processes—for example, a system log file that can be accessed and modified by a number of processes in the system.

1.3 File Types

A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts

- Name
- Extension, separated by a period character.

To perform an operation on the file, the O/S must recognize its types. Normally file type is implemented by including it as a part of file name. The type of file represents the contents of that file. Using this way user and O/S can easily identify type of the file.

In Dos the name can be of 8 characters long followed by a period and extension up to 3 characters. The system uses the extension to identify the file type and allow operations that can be perform on that file. For example only file with **.com, .exe** or **.bat** extension are used for executing a file. **.com** and **.exe**are two forms of binary executable file whereas **.bat** file is a batch files. DOS supports only a few extensions but application program also uses extension to indicate file types.

File type	Extension	Function

executable	exe, com, bin or none	ready-to-run machine language program
Object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
Library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
Multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Table 1.1 Common File Types

1.4 Directory Structure

A **directory** is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner. The organization is done in two parts,

1. Disks are split into one or more partitions, also known as minidisks or volumes
2. Each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents.

The device directory records information such as name, location, size, and type. There are several logical structures of a directory, which are given below.

1.4.1 Single – level directory

Single level directory is simplest directory structure. In this files are contained in same directory which make it easy to support and understand. A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name. If two users call their dataset test, then the unique name rule violated.

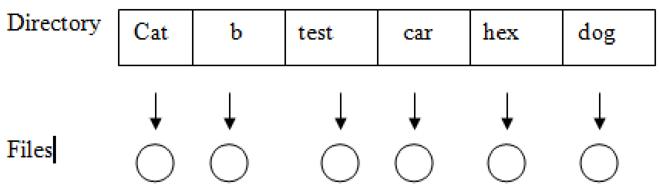

Fig 1.1 Single Level Directory

Advantages:

- Since it is a single directory, so its implementation is very easy.
- If files are smaller in size, searching will be faster.
- The operations like file creation, searching, deletion, updating is very easy in such a directory structure.

Disadvantages:

- There may change of name collision because two files can not have the same name.
- Searching will become time taking if directory will large.
- In this cannot group the same type of files together.

1.4.2 Two - level directory

Two-level directory systems are used to avoid the problem caused by the single-level directory system, we have learned in previous tutorial. In two-level directory systems, give each user a private directory. Therefore in this two-level directory system, names chosen by one user don't interfere with names chosen by a different user and there is no any problem that is caused by the same name occurring in two or more than two directories.

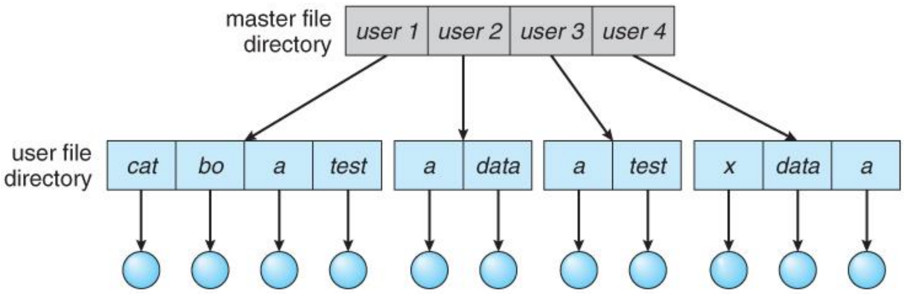

Fig 1.2 Two Level Directory

Advantages:

- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages:

- A user is not allowed to share files with other users.
- Two files of the same type cannot be grouped together in the same user.

1.4.3 Tree-structured directory

A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path. Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows the user to create their own subdirectories and to organize on their files accordingly.

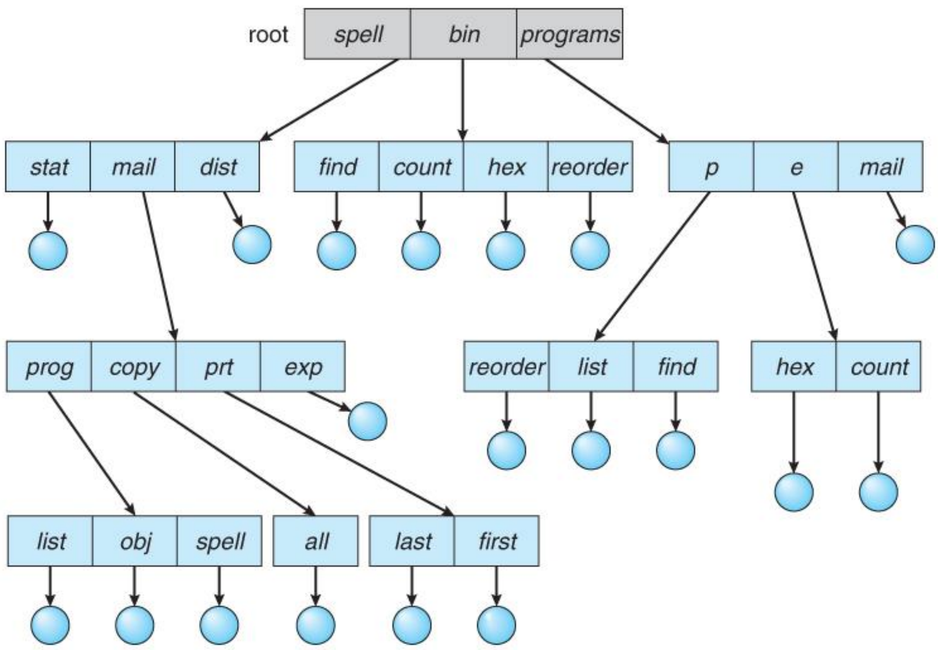

Fig 1.3 Tree Structured directory

Advantages:

- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy; we can use both absolute path as well as relative.

Disadvantages:

- Every file does not fit into the hierarchical model; files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

1.4.4 Acyclic graph directory

An acyclic graph is a graph with no cycle and allows sharing subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree - structured directory. It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separated them from other projects and files of other programmers since they are working on a joint project so they want to the subdirectories into their own directories. The common subdirectories should be shared. So here we use acyclic directories.

It is the point to note that shared file is not the same as copy file if any programmer makes some changes in the subdirectory it will reflect in both subdirectories.

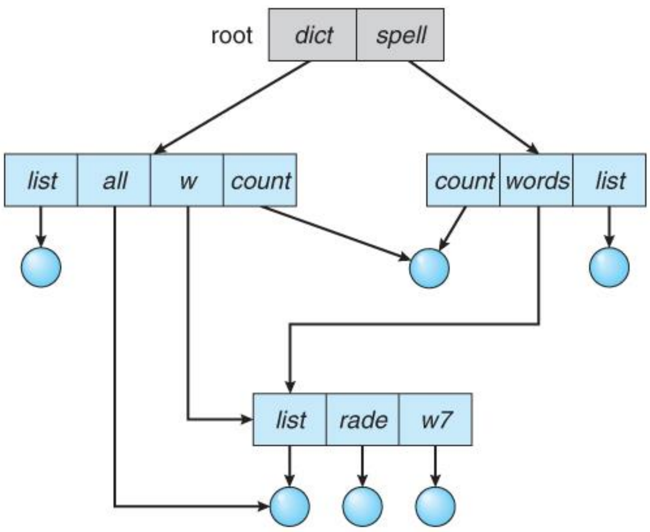

Fig 1.4 Acyclic graph Directory

Advantages:

- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case of deleting it may create the problem,
- If the link is soft link then after deleting the file we left with a dangling pointer.
- In case of hard link, to delete a file we have to delete all the reference associated with it.

1.4.5 General graph directory structure

In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory. The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.

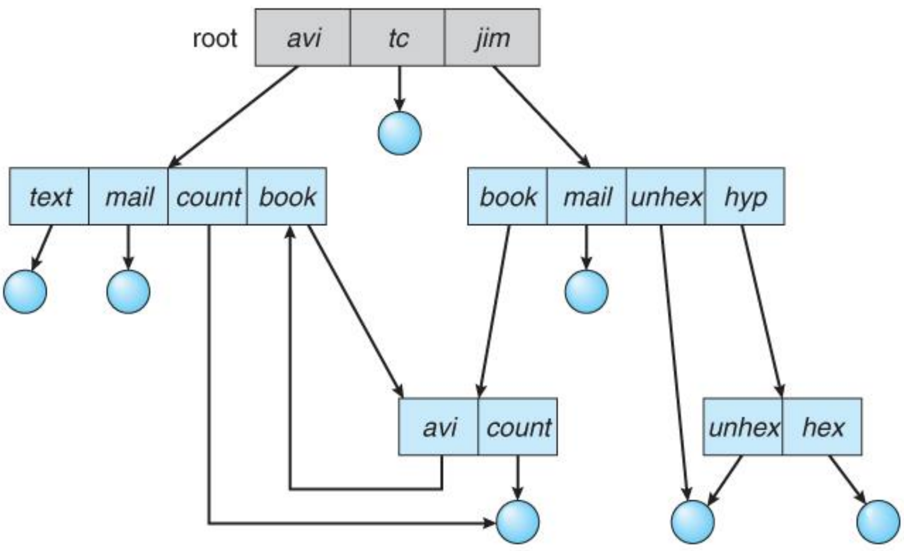

Fig 1.5 General graph directory

Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

1.5 File Protection

- File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally.

- Bugs in the file-system software can also cause file contents to be lost.
- Protection can be provided in many ways.
- For a small single-user system, to provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multiuser system, however, other mechanisms are needed.

1.5.1 Types of Access

The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

- **Read**: Read from the file.
- **Write**: Write or rewrite the file.
- **Execute**: Load the file into memory and execute it.
- **Append**: Write new information at the end of the file.
- **Delete**: Delete the file and free its space for possible reuse.
- **List**: List the name and attributes of the file.

1.5.2 Access Control

The most general scheme to implement identity dependent access is to associate with each file and directory an **access-control list (ACL)** specifying user names and the types of access allowed for each user.

When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:

- **Owner**. The user who created the file is the owner.
- **Group**. A set of users who are sharing the file and need similar access is a group, or work group.
- **Universe**. All other users in the system constitute the universe.

1.5.3 Other Protection Approaches

Another approach to the protection problem is to associate a password with each file. Just as access to the computer system is often controlled by a password, access to each file can be controlled in the same way. If the passwords are chosen randomly and changed often, this scheme may be effective in limiting access to a file. The use of passwords has some disadvantages,

1. The number of passwords that a user needs to remember may become large, making the scheme impractical.
2. If only one password is used for all the files, then once it is discovered, all files are accessible; protection is on an all-or-none basis.

In a multilevel directory structure, the protection is not only for individual files but also collections of files in subdirectories; that is, to provide a mechanism for directory protection. The directory operations that must be protected are somewhat different from the file operations. Sometimes, knowledge of the existence and name of a file is significant in itself. Thus, listing the contents of a directory must be a protected operation. Similarly, if a path name refers to a file in a directory, the user must be allowed access to both the directory and the file.