**File-System**

**File Concept**

**File Attributes**

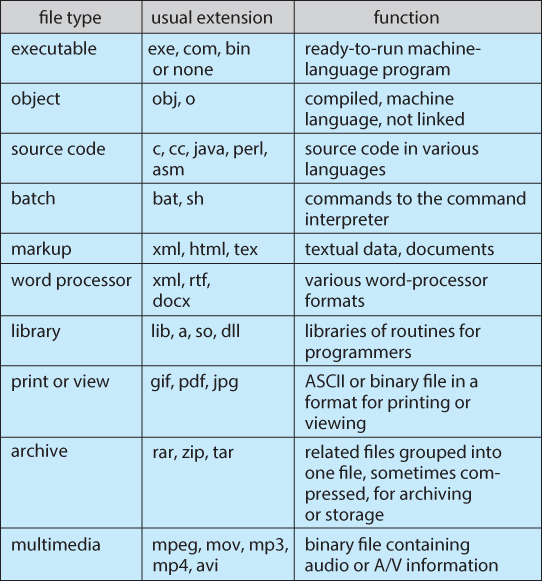
* Different OSes keep track of different file attributes, including:
  + **Name** - Some systems give special significance to names, and particularly extensions ( .exe, .txt, etc. ), and some do not. Some extensions may be of significance to the OS ( .exe ), and others only to certain applications ( .jpg )
  + **Identifier** ( e.g. inode number )
  + **Type** - Text, executable, other binary, etc.
  + **Location** - on the hard drive.
  + **Size**
  + **Protection**
  + **Time & Date**
  + **User ID**

**File Operations**

* The file ADT supports many common operations:
  + Creating a file
  + Writing a file
  + Reading a file
  + Repositioning within a file
  + Deleting a file
  + Truncating a file.
* Most OSes require that files be ***opened*** before access and ***closed*** after all access is complete. Normally the programmer must open and close files explicitly, but some rare systems open the file automatically at first access. Information about currently open files is stored in an ***open file table***, containing for example:
  + **File pointer** - records the current position in the file, for the next read or write access.
  + **File-open count** - How many times has the current file been opened ( simultaneously by different processes ) and not yet closed? When this counter reaches zero the file can be removed from the table.
  + **Disk location of the file.**
  + **Access rights**
* Some systems provide support for ***file locking.***
  + A ***shared lock*** is for reading only.
  + A ***exclusive lock*** is for writing as well as reading.
  + An ***advisory lock*** is informational only, and not enforced. ( A "Keep Out" sign, which may be ignored. )
  + A ***mandatory lock*** is enforced. ( A truly locked door. )
  + UNIX used advisory locks, and Windows uses mandatory locks.

**File Types**

* Windows ( and some other systems ) use special file extensions to indicate the type of each file:

  
**Figure 11.3 - Common file types.**

* Macintosh stores a creator attribute for each file, according to the program that first created it with the create( ) system call.
* UNIX stores magic numbers at the beginning of certain files. ( Experiment with the "file" command, especially in directories such as /bin and /dev )

**File Structure**

* Some files contain an internal structure, which may or may not be known to the OS.
* For the OS to support particular file formats increases the size and complexity of the OS.
* UNIX treats all files as sequences of bytes, with no further consideration of the internal structure. ( With the exception of executable binary programs, which it must know how to load and find the first executable statement, etc. )
* Macintosh files have two ***forks*** - a ***resource fork***, and a ***data fork***. The resource fork contains information relating to the UI, such as icons and button images, and can be modified independently of the data fork, which contains the code or data as appropriate.

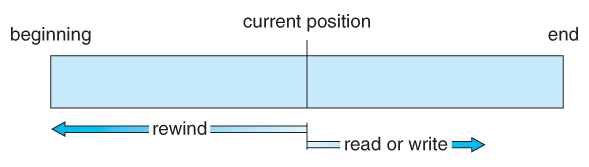
**Internal File Structure**

* Disk files are accessed in units of physical blocks, typically 512 bytes or some power-of-two multiple thereof. ( Larger physical disks use larger block sizes, to keep the range of block numbers within the range of a 32-bit integer. )
* Internally files are organized in units of logical units, which may be as small as a single byte, or may be a larger size corresponding to some data record or structure size.
* The number of logical units which fit into one physical block determines its ***packing***, and has an impact on the amount of internal fragmentation ( wasted space ) that occurs.
* As a general rule, half a physical block is wasted for each file, and the larger the block sizes the more space is lost to internal fragmentation.

**Access Methods**

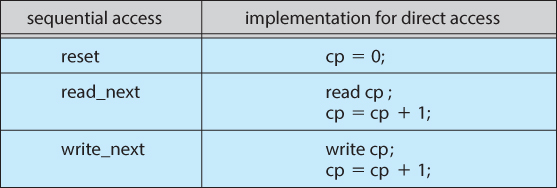
**Sequential Access**

* A sequential access file emulates magnetic tape operation, and generally supports a few operations:
  + read next - read a record and advance the tape to the next position.
  + write next - write a record and advance the tape to the next position.
  + rewind
  + skip n records - May or may not be supported. N may be limited to positive numbers, or may be limited to +/- 1.

  
**Figure 11.4 - Sequential-access file.**

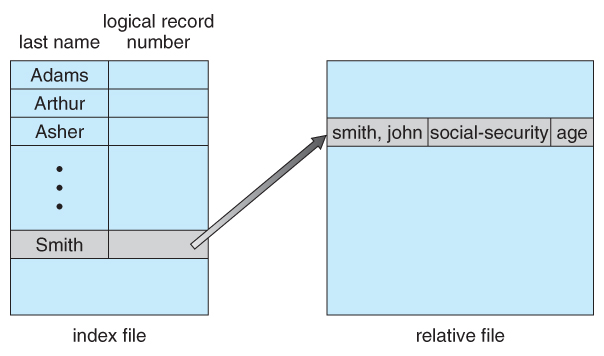
**Direct Access**

* Jump to any record and read that record. Operations supported include:
  + read n - read record number n. ( Note an argument is now required. )
  + write n - write record number n. ( Note an argument is now required. )
  + jump to record n - could be 0 or the end of file.
  + Query current record - used to return back to this record later.
  + Sequential access can be easily emulated using direct access. The inverse is complicated and inefficient.

  
**Figure 11.5 - Simulation of sequential access on a direct-access file.**

**Other Access Methods**

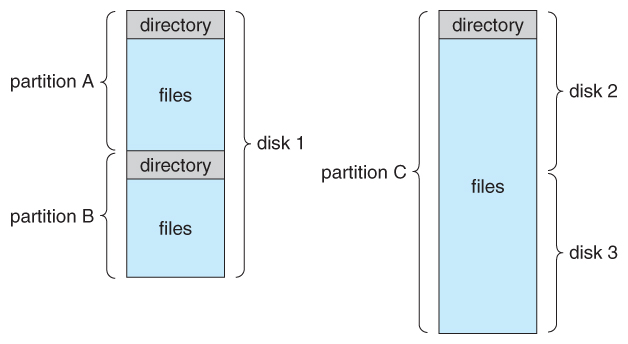
* An indexed access scheme can be easily built on top of a direct access system. Very large files may require a multi-tiered indexing scheme, i.e. indexes of indexes.

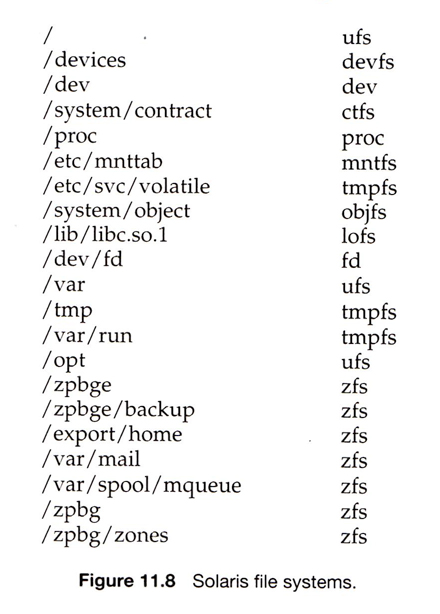
  
**Figure 11.6 - Example of index and relative files.**

**Directory Structure**

**Storage Structure**

* A disk can be used in its entirety for a file system.
* Alternatively a physical disk can be broken up into multiple ***partitions, slices, or mini-disks***, each of which becomes a virtual disk and can have its own filesystem. ( or be used for raw storage, swap space, etc. )
* Or, multiple physical disks can be combined into one ***volume***, i.e. a larger virtual disk, with its own filesystem spanning the physical disks.

  
**Figure 11.7 - A typical file-system organization.**

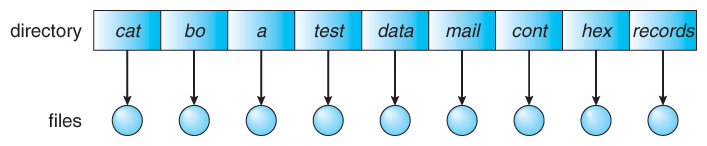


**Directory Overview**

* Directory operations to be supported include:
  + Search for a file
  + Create a file - add to the directory
  + Delete a file - erase from the directory
  + List a directory - possibly ordered in different ways.
  + Rename a file - may change sorting order
  + Traverse the file system.

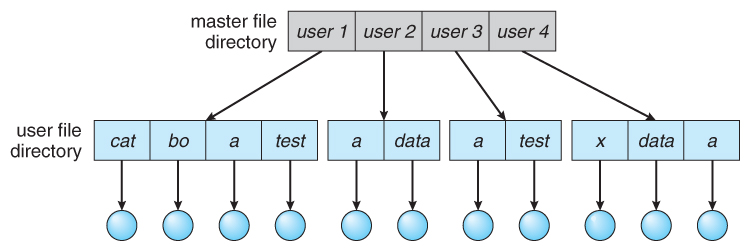
**Single-Level Directory**

* Simple to implement, but each file must have a unique name.

 **Figure 11.9 - Single-level directory.**

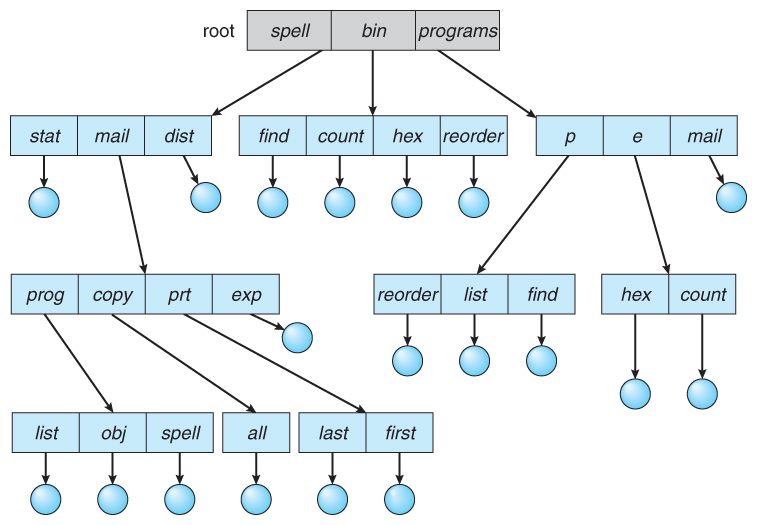
**Two-Level Directory**

* Each user gets their own directory space.
* File names only need to be unique within a given user's directory.
* A master file directory is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.
* A separate directory is generally needed for system ( executable ) files.
* Systems may or may not allow users to access other directories besides their own
  + If access to other directories is allowed, then provision must be made to specify the directory being accessed.
  + If access is denied, then special consideration must be made for users to run programs located in system directories. A ***search path*** is the list of directories in which to search for executable programs, and can be set uniquely for each user.

  
**Figure 11.10 - Two-level directory structure.**

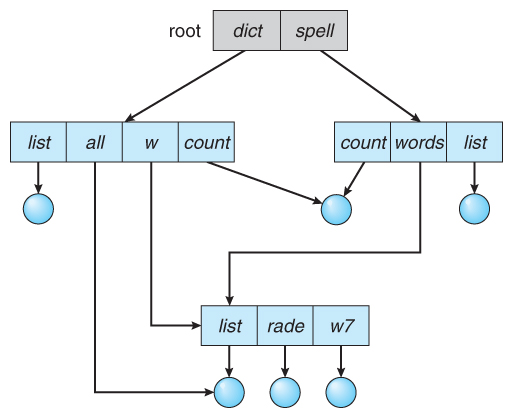
**Tree-Structured Directories**

* An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar.
* Each user / process has the concept of a ***current directory*** from which all ( relative ) searches take place.
* Files may be accessed using either absolute pathnames ( relative to the root of the tree ) or relative pathnames ( relative to the current directory. )
* Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
* One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

  
**Figure 11.11 - Tree-structured directory structure.**

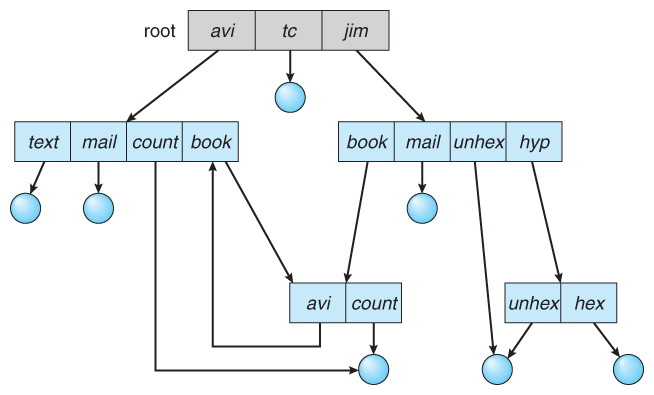
**Acyclic-Graph Directories**

* When the same files need to be accessed in more than one place in the directory structure ( e.g. because they are being shared by more than one user / process ), it can be useful to provide an acyclic-graph structure. ( Note the ***directed*** arcs from parent to child. )
* UNIX provides two types of ***links*** for implementing the acyclic-graph structure. ( See "man ln" for more details. )
  + A ***hard link*** ( usually just called a link ) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
  + A ***symbolic link***, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.
* Windows only supports symbolic links, termed ***shortcuts.***
* Hard links require a ***reference count***, or ***link count*** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.
* For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:
  + One option is to find all the symbolic links and adjust them also.
  + Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
  + What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?

  
**Figure 11.12 - Acyclic-graph directory structure.**

**General Graph Directory**

* If cycles are allowed in the graphs, then several problems can arise:
  + Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. ( Or not to follow symbolic links, and to only allow symbolic links to refer to directories. )
  + Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem. ( chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted. )

  
**Figure 11.13 - General graph directory.**

**Protection**

* Files must be kept safe for reliability ( against accidental damage ), and protection ( against deliberate malicious access. ) The former is usually managed with backup copies. This section discusses the latter.
* One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

**Types of Access**

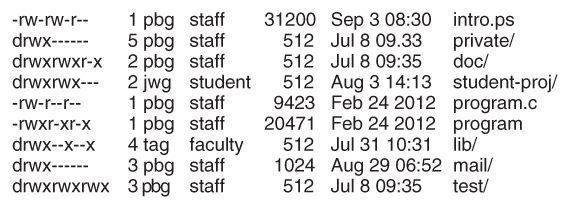
* The following low-level operations are often controlled:
  + Read - View the contents of the file
  + Write - Change the contents of the file.
  + Execute - Load the file onto the CPU and follow the instructions contained therein.
  + Append - Add to the end of an existing file.
  + Delete - Remove a file from the system.
  + List -View the name and other attributes of files on the system.
* Higher-level operations, such as copy, can generally be performed through combinations of the above.

**Access Control**

* One approach is to have complicated ***Access Control Lists, ACL,*** which specify exactly what access is allowed or denied for specific users or groups.
  + The AFS uses this system for distributed access.
  + Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown. ( AFS allows some wild cards, so for example all users on a certain remote system may be trusted, or a given username may be trusted when accessing from any remote system. )
* UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. ( See "man chmod" for full details. ) The RWX bits control the following privileges for ordinary files and directories:

|  |  |  |
| --- | --- | --- |
| **bit** | **Files** | **Directories** |
| **R** | Read ( view ) file contents. | Read directory contents. Required to get a listing of the directory. |
| **W** | Write ( change ) file contents. | Change directory contents. Required to create or delete files. |
| **X** | Execute file contents as a program. | Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access. |

* In addition there are some special bits that can also be applied:
  + The set user ID ( SUID ) bit and/or the set group ID ( SGID ) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program. This allows users running specific programs to have access to files ( ***while running that program***) to which they would normally be unable to access. Setting of these two bits is usually restricted to root, and must be done with caution, as it introduces a potential security leak.
  + The sticky bit on a directory modifies write permission, allowing users to only delete files for which they are the owner. This allows everyone to create files in /tmp, for example, but to only delete files which they have created, and not anyone else's.
  + The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for execute permission for the user, group, and others, respectively. If the letter is lower case, ( s, s, t ), then the corresponding execute permission is not also given. If it is upper case, ( S, S, T ), then the corresponding execute permission IS given.
  + The numeric form of chmod is needed to set these advanced bits.

  
**Sample permissions in a UNIX system.**

* Windows adjusts files access through a simple GUI:

**Other Protection Approaches and Issues**

* Some systems can apply passwords, either to individual files, or to specific sub-directories, or to the entire system. There is a trade-off between the number of passwords that must be maintained ( and remembered by the users ) and the amount of information that is vulnerable to a lost or forgotten password.
* Older systems which did not originally have multi-user file access permissions ( DOS and older versions of Mac ) must now be ***retrofitted*** if they are to share files on a network.
* Access to a file requires access to all the files along its path as well. In a cyclic directory structure, users may have different access to the same file accessed through different paths.
* Sometimes just the knowledge of the existence of a file of a certain name is a security ( or privacy ) concern. Hence the distinction between the R and X bits on UNIX directories.