File System Interface and Implementation

The Concept of a File:

Contiguous logical address spacenTypes:
Data
- ☐numeric
- ☐character
- ☐binary

Program

File Structure
None - sequence of words, bytes
- Simple record structure
- Lines
- Fixed length
- Variable length
- Complex Structures
- Formatted document
- Relocatable load file

Can simulate last two with first method by inserting appropriate control characters
Who decides:
- Operating system
- Program

File Attributes
Name – only information kept in human-readable form
Identifier – unique tag (number) identifies file within file system
Type – needed for systems that support different types
Location – pointer to file location on device
Size – current file size
Protection – controls who can do reading, writing, executing
Time, date, and user identification – data for protection, security, and usage monitoring
Information about files are kept in the directory structure, which is maintained on the disk

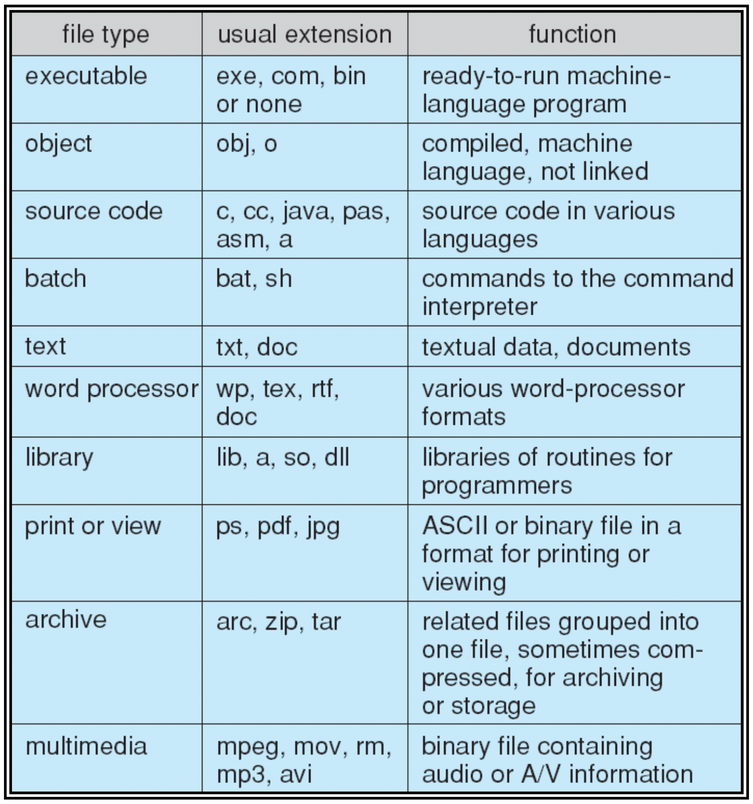
File Operations
File is an **abstract data type**
Create
Write
Read
Reposition within file
Delete
Truncate
Open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory

Close (F_i) – move the content of entry F_i in memory to directory structure on disk

Open Files
Several pieces of data are needed to manage open files:
lFile pointer: pointer to last read/write location, per process that has the file open
File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
Disk location of the file: cache of data access information
Access rights: per-process access mode information
Open File Locking
Provided by some operating systems and file systems
Mediates access to a file
Mandatory or advisory:
Mandatory – access is denied depending on locks held and requested
Advisory – processes can find status of locks and decide what to do

File Locking Example – Java API
```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
        public static final boolean EXCLUSIVE = false;
        public static final boolean SHARED = true;
        public static void main(String arsg[]) throws IOException {
                FileLock sharedLock = null;
                FileLock exclusiveLock = null;
                try {
                        RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
                        // get the channel for the file
                        FileChannel ch = raf.getChannel();
                        // this locks the first half of the file - exclusive
                        exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
                        /** Now modify the data . . . */
                        // release the lock
                        exclusiveLock.release();
// this locks the second half of the file - shared
                        sharedLock = ch.lock(raf.length()/2+1, raf.length(),
        SHARED);
                        /** Now read the data . . . */
                        // release the lock
                        sharedLock.release();
                } catch (java.io.IOException ioe) {
                        System.err.println(ioe);
                }finally {
                        if (exclusiveLock != null)
                        exclusiveLock.release();
                        if (sharedLock != null)
                        sharedLock.release();
                }
        }
}
```

File Types – Name, Extension



file type	usual 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

Access Methods

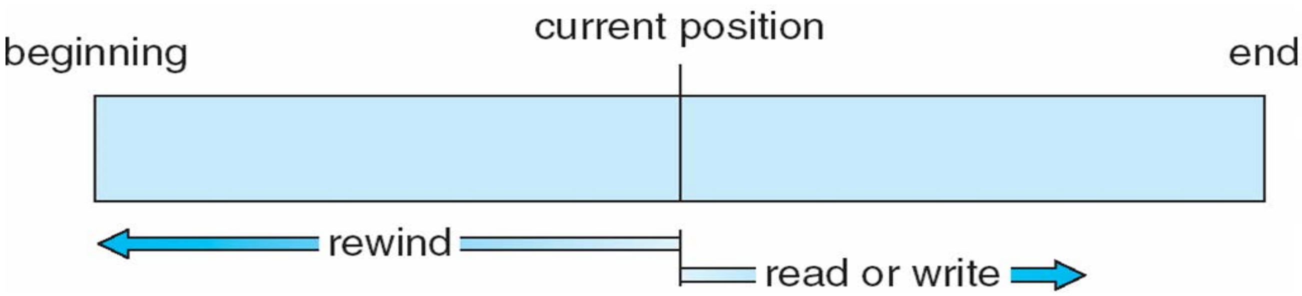
Sequential Access

read next
write next
reset
no read after last write
  (rewrite)

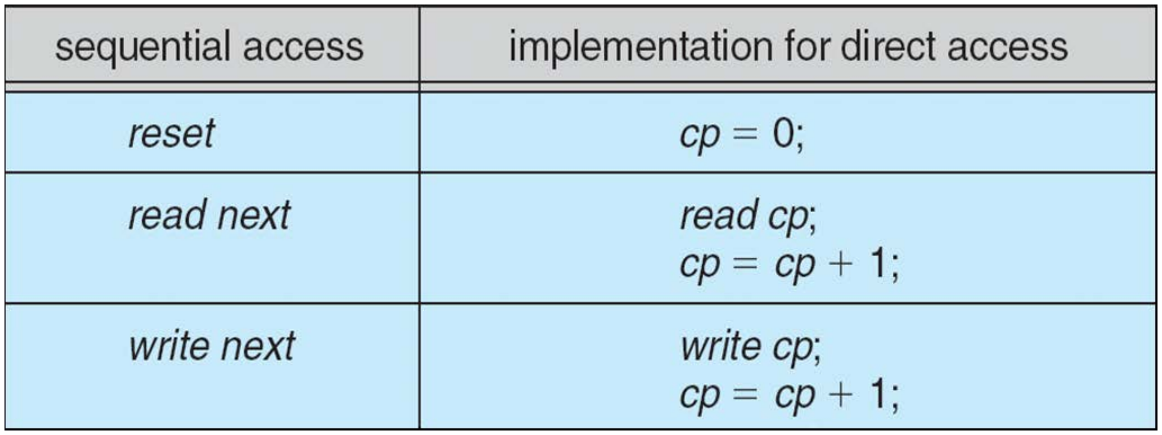
Direct Access

read *n*
write *n*
position to *n*
  read next
  write next
rewrite *n*
n = relative block number

Sequential-access File

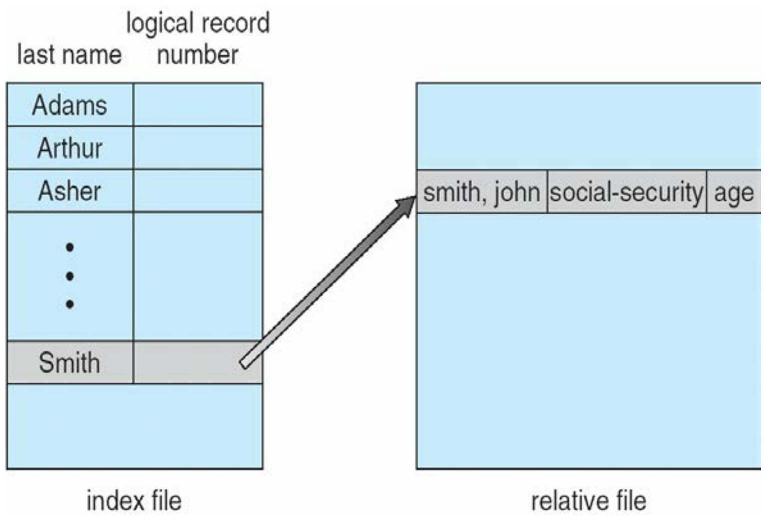


Simulation of Sequential Access on Direct-access File

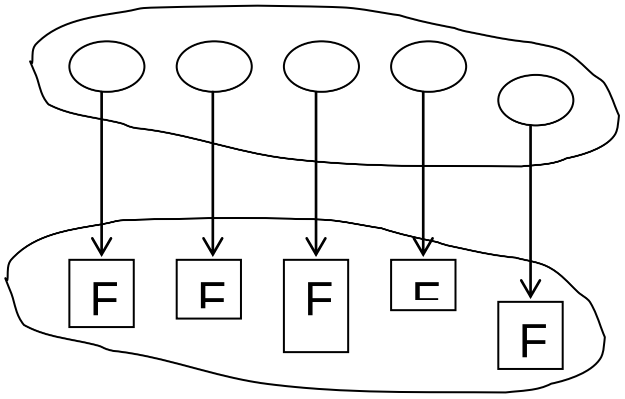


sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Example of Index and Relative Files



Directory Structure
A collection of nodes containing information about all files

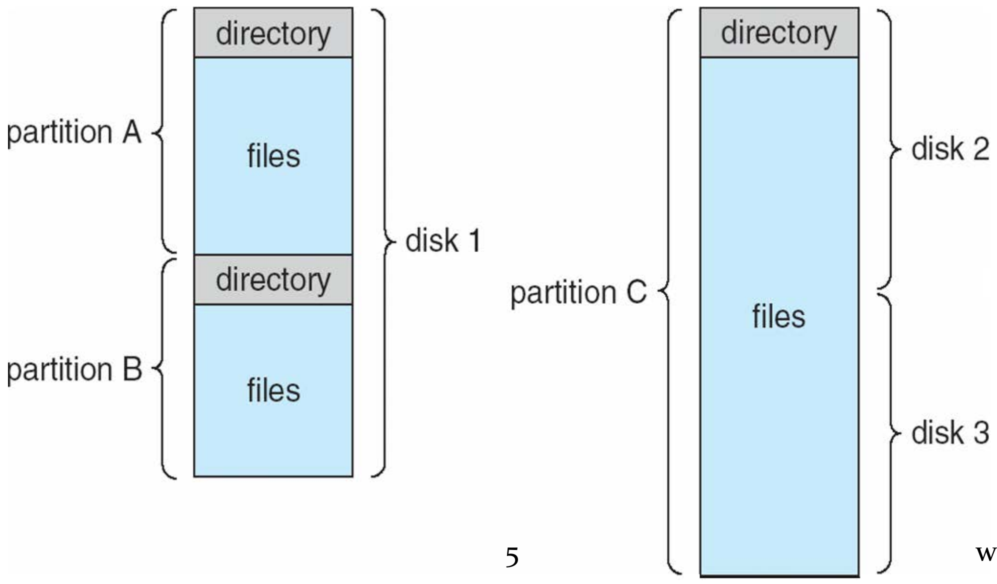


Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

Disk Structure

- Disk can be subdivided into partitions
- Disks or partitions can be RAID protected against failure
- Disk or partition can be used raw – without a file system, or formatted with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a volume
- Each volume containing file system also tracks that file system's info in device directory or volume table of contents
- As well as general-purpose file systems there are many special-purpose file systems, frequently all within the same operating system or computer
-

A Typical File-system Organization



Operations Performed on Directory

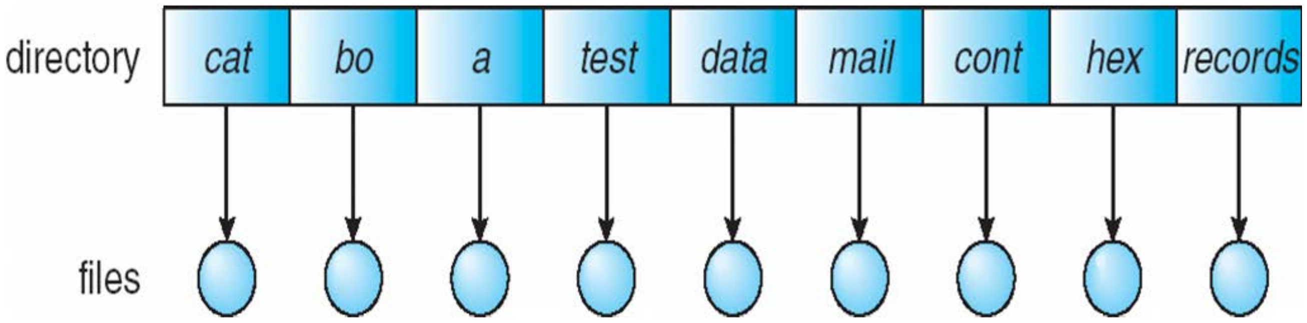
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system
-

Organize the Directory (Logically) to Obtain

Efficiency – locating a file quickly
Naming – convenient to users
Two users can have same name for different files
The same file can have several different names
nGrouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

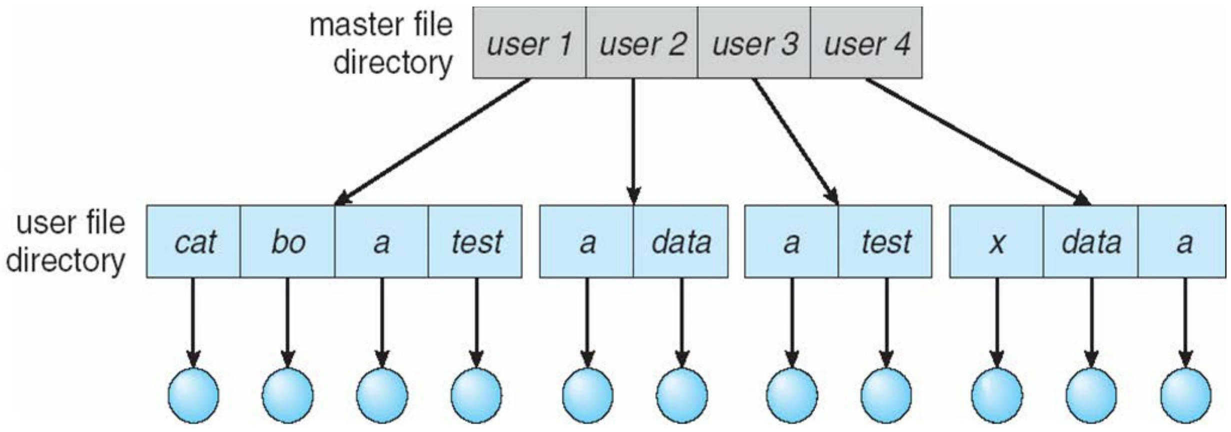
A single directory for all users



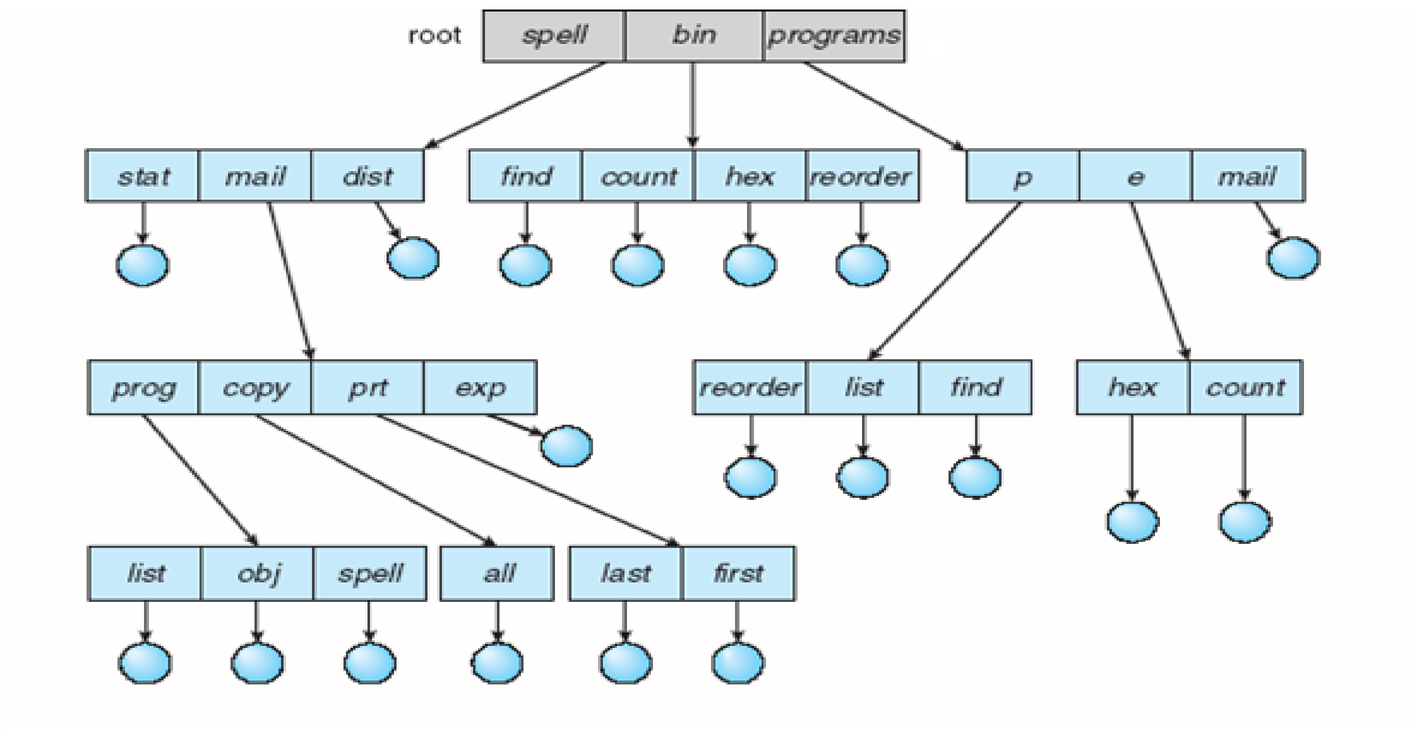
Naming problem
Grouping problem

Two-Level Directory

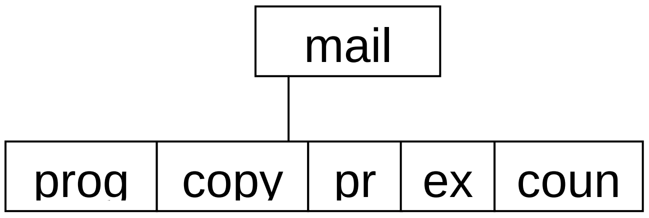
Separate directory for each user



Path name
Can have the same file name for different user
Efficient searching
No grouping capability
Tree-Structured Directories



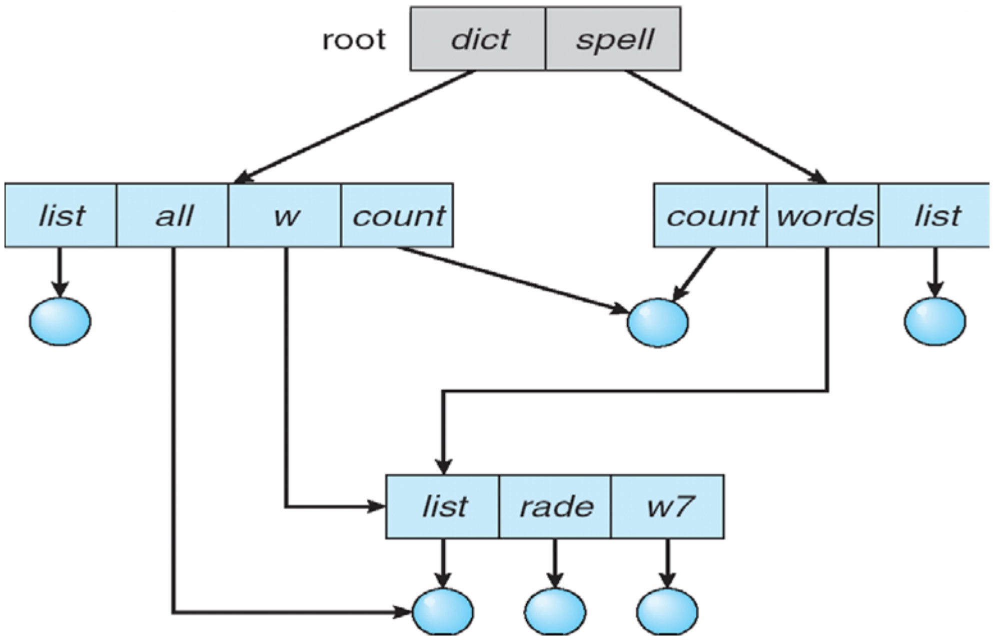
Efficient searchingnGrouping CapabilitynCurrent directory (working directory)
cd /spell/mail/prog
type list
Absolute or **relative** path name
Creating a new file is done in current directory
Delete a file
rm <file-name>
Creating a new subdirectory is done in current directory
mkdir <dir-name>
Example: if in current directory /mail
mkdir count



Deleting “mail” Þ deleting the entire subtree rooted by “mail”

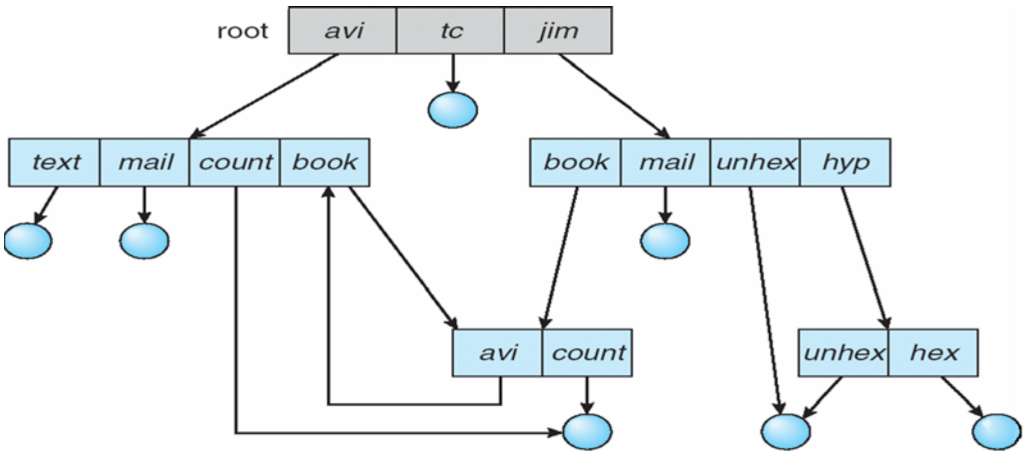
Acyclic-Graph Directories

Have shared subdirectories and files



Two different names (aliasing)nIf *dict* deletes *list* Þ dangling pointer
Solutions:
Backpointers, so we can delete all pointers
Variable size records a problem
Backpointers using a daisy chain organization
Entry-hold-count solution
New directory entry type
Link – another name (pointer) to an existing file
Resolve the link – follow pointer to locate the file

General Graph Directory

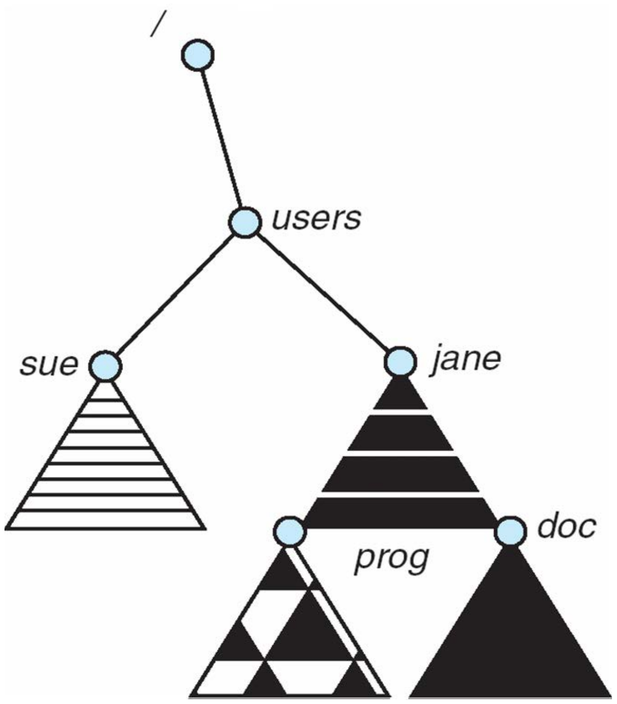


How do we guarantee no cycles?
Allow only links to file not subdirectories
Garbage collection
Every time a new link is added use a cycle detection algorithm to determine whether it is OK

File System Mounting

A file system must be **mounted** before it can be accessed
A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**
(a) Existing. (b) Unmounted Partition

Mount Point



File Sharing

Sharing of files on multi-user systems is desirablenSharing may be done through a **protection** schemenOn distributed systems, files may be shared across a networknNetwork File System (NFS) is a common distributed file-sharing method

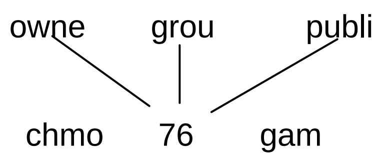
File Sharing – Multiple Users
User IDs identify users, allowing permissions and protections to be per-usernGroup IDs** allow users to be in groups, permitting group access rights

File Sharing – Remote File Systems
Uses networking to allow file system access between systems
Manually via programs like FTP
Automatically, seamlessly using **distributed file systems**
Semi automatically via the **world wide web**
Client-server model allows clients to mount remote file systems from servers
Server can serve multiple clients
Client and user-on-client identification is insecure or complicated
NFS is standard UNIX client-server file sharing protocol
CIFS is standard Windows protocol
Standard operating system file calls are translated into remote calls
Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes
Remote file systems add new failure modes, due to network failure, server failure
Recovery from failure can involve state information about status of each remote request
Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security
File Sharing – Consistency Semantics
Consistency semantics specify how multiple users are to access a shared file simultaneously
Similar to Ch 7 process synchronization algorithms
☐Tend to be less complex due to disk I/O and network latency (for remote file systems
Andrew File System (AFS) implemented complex remote file sharing semantics
Unix file system (UFS) implements:
☐Writes to an open file visible immediately to other users of the same open file
☐Sharing file pointer to allow multiple users to read and write concurrently

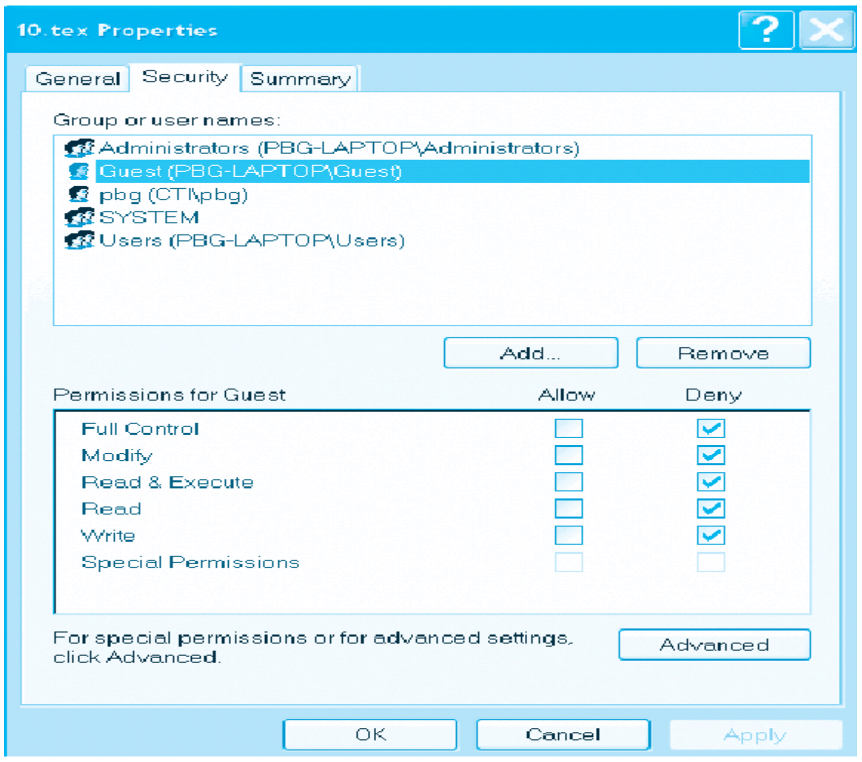
File Protection
File owner/creator should be able to control:
what can be done
by whomnTypes of access
Read
Write
Execute
Append
Delete
List

Access Lists and Groups
Mode of access: read, write, execute
Three classes of users
RWX
a) **owner access** 7 Þ 1 1
1
RWX
b) **group access** 6 Þ 1 1
0
RWX
c) **public access** 1 Þ 0 0
1
Ask manager to create a group (unique name), say G, and add some users to the group.
For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file
chgrp G game

Windows XP Access-control List Management



Group or user names:
Administrators (PBG-LAPTOP\Administrators)
Guest (PBG-LAPTOP\Guest)
pbg (CTI\pbg)
SYSTEM
Users (PBG-LAPTOP\Users)

Permissions for Guest	Allow	Deny
Full Control		✓
Modify		✓
Read & Execute		✓
Read		✓
Write		✓
Special Permissions		

For special permissions or for advanced settings, click Advanced.

A Sample UNIX Directory Listing

```
-rw-rw-r--    1 pbg   staff     31200  Sep 3 08:30    intro.ps
drwx------    5 pbg   staff       512  Jul 8 09.33    private/
drwxrwxr-x    2 pbg   staff       512  Jul 8 09:35    doc/
drwxrwx---    2 pbg   student     512  Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg   staff      9423  Feb 24 2003    program.c
-rwxr-xr-x    1 pbg   staff     20471  Feb 24 2003    program
drwx--x--x    4 pbg   faculty     512  Jul 31 10:31   lib/
drwx------    3 pbg   staff      1024  Aug 29 06:52   mail/
drwxrwxrwx    3 pbg   staff       512  Jul 8 09:35    test/
```

File System Structure:

File structure is a structure, which is according to a required format that operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files:

ORDINARY FILES

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

DIRECTORY FILES

- These files contain list of file names and other information related to these files.

SPECIAL FILES:

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types

- **Character special files** - data is handled character by character as in case of terminals or printers.
- **Block special files** - data is handled in blocks as in the case of disks and tapes.

File System Implementation:

- **Boot control block** contains info needed by system to boot OS from that volume
- **Volume control block** contains volume details
- **Directory structure** organizes the files
- **Per-file File Control Block** (**FCB**) contains many details about the file

Directory Implementation:

The simplest form of directory implementation is linear list ,it has a linear list of file names with pointer to the data blocks. To add a new new file first a full checking of the linear list is made so that there would not be any duplicate entries. If there is no duplicate entry the a new entry is made at the end of the directory.

Deletion is simple ,we search the directory and release the space associated with it. To reuse a directory entry we can mark the entry as unused or we can attach it to the list of free directories.

The real disadvantage of linear lists is that its operations are time consuming. It because we have to traverse the entire list to find an entry. A sorted list reduces this problem but sorting it self is a problem and the list are constantly updated, so sorting will be needed all ways.

Hash table:

Here also a linear list stores all the directory entries. In addition to that we use a hash table too, the values in the hash table computed from the file name. All the entries in the returns a pointer to the file name in the linear list. This greatly reduces the search time ,but some mechanisms should be there to prevent collision so that two file names do not hash to the same location.

Allocation Methods:

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

CONTIGUOUS ALLOCATION

- Each file occupy a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

LINKED ALLOCATION

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

INDEXED ALLOCATION

- Provides solutions to problems of contigous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Free Space Management:

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks.
- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
- When a file is deleted, its disk space is added to the free-space list.

Efficiency and Performance:

Recent research advocates asymmetric multi-core architectures, where cores in the same processor can have different performance.

These architectures support single-threaded performance and multithreaded throughput at lower costs (e.g., die size and power). However, they also pose unique challenges to operating systems, which traditionally assume homogeneous hardware. This paper presents AMPS, an operating system scheduler that efficiently supports both SMP-and NUMA-style performance-asymmetric architectures.

AMPS contains three components: asymmetry-aware load balancing, faster-core-first scheduling, and NUMA-aware migration. We have implemented AMPS in Linux kernel 2.6.16 and used CPU clock modulation to emulate performance asymmetry on an SMP and NUMA system.

For various workloads, we show that AMPS achieves a median speedup of 1.16 with a maximum of 1.44 over stock Linux on the SMP, and a median of 1.07 with a maximum of 2.61 on the NUMA system. Our results also show that AMPS improves fairness and repeatability of application performance measurements.