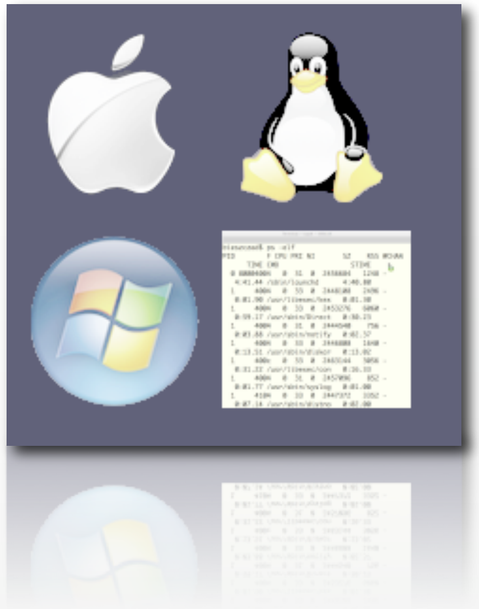


Lecture 11: File-System Interface

COMP362 Operating Systems
Prof. AJ Bieszczad

Outline

- File Concept
  - structure
  - attributes
  - access
  - operations
  - types
- Directory Structure
- File System Protection
  - UNIX File Access Lists and Groups
- File-System Mounting

File Concept

- File system is at the core of the operating system
  - however, it's not part of the kernel
- File is an abstract concept that represents a unit of storage
  - used across all operating systems
- File provides contiguous logical address space
- There are numerous types of files:
  - data files:
    - numeric,
    - character,
    - binary
    - … and many more
  - programs
    - executables,
    - libraries,
    - object code, etc.

File Structure

- Numerous possibilities for internal file structure
  - No structure
    - sequence of “words” (units), bytes
  - Simple record structure
    - e.g., lines
      - fixed length
      - variable length
  - Complex Structures
    - formatted document
    - relocatable load file
    - can be simulated with simple records by inserting appropriate control characters
- Who decides on the structure:
  - operating system (or rather it’s designers)
    - e.g., executables
  - standards
    - e.g., email, audio, video, books, documents, etc.
  - application programs

File Attributes

- Information about files is kept in the directory structure, which is maintained by the OS on the disk. It includes file attributes:
  - Name
    - only information kept in human-readable form
  - Identifier
    - unique tag (number) identifies file within the 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
    - a.k.a. access control
  - Time, date, and user identification
    - data for protection, security, and usage monitoring

File Information – POSIX API

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

#define oops(msg, errn) { perror(msg); exit(errn); }

#define fileName "test.txt"

int main(int argc, char *argv[])
{
   struct stat fileInfo;
   if (stat (fileName, &fileInfo) < 0)
      oops("Cannot stat the file.", 1);

   printf("FILE INFO: %s\n\n", fileName);
   printf("%-37s %12s = %10u\n", "Device inode resides on", "st_dev", fileInfo.st_dev);
   printf("%-37s %12s = %10d\n", "Inode's number", "st_ino", fileInfo.st_ino);
   printf("%-37s %12s = %10d\n", "Inode protection mode", "st_mode", fileInfo.st_mode);
   printf("%-37s %12s = %10d\n", "Number or hard links to the file", "st_nlink", fileInfo.st_nlink);
   printf("%-37s %12s = %10d\n", "User-id of owner", "st_uid", fileInfo.st_uid);
   printf("%-37s %12s = %10d\n", "Group-id of owner", "st_gid", fileInfo.st_gid);
   printf("%-37s %12s = %10d\n", "Device type, for special file inode", "st_rdev", fileInfo.st_rdev);
   printf("%-37s %12s = %10d\n", "Time since last access", "st_atimespec", fileInfo.st_atimespec.tv_sec);
   printf("%-37s %12s = %10d\n", "Time since last data modification", "st_mtimespec", fileInfo.st_mtimespec.tv_sec);
   printf("%-37s %12s = %10d\n", "Time since last file status change", "st_ctimespec", fileInfo.st_ctimespec.tv_sec);
   printf("%-37s %12s = %10d\n", "File size, in bytes", "st_size", fileInfo.st_size);
   printf("%-37s %12s = %10d\n", "Blocks allocated for file", "st_blocks", fileInfo.st_blocks);
   printf("%-37s %12s = %10d\n", "Optimal file sys I/O ops blocksize", "st_blksize", fileInfo.st_blksize);
   printf("%-37s %12s = %10o\n", "User defined flags for file", "st_flags", fileInfo.st_flags);
   printf("%-37s %12s = %10d\n", "File generation number", "st_gen", fileInfo.st_gen);
}
```

```
FILE INFO: test.txt

Device inode resides on                    st_dev =  234881032
Inode's number                             st_ino =      31642
Inode protection mode                     st_mode =      16877
Number or hard links to the file         st_nlink =          2
User-id of owner                           st_uid =        503
Group-id of owner                          st_gid =        503
Device type, for special file inode       st_rdev =          0
Time since last access                 st_atimespec = 1239740898
Time since last data modification      st_mtimespec = 1239740907
Time since last file status change     st_ctimespec = 1239740907
File size, in bytes                        st_size =        204
Blocks allocated for file                st_blocks =          0
Optimal file sys I/O ops blocksize       st_blksize =       4096
User defined flags for file               st_flags =          0
File generation number                      st_gen =          0
```

```
$ man 2 stat
...
look for struct stat
```

Also see:

```
$ stat test.txt
$ man stat
```

File Operations

- File can also be viewed as an abstract data type with operations such as:
  - Create
  - Open(File)
    - search the directory structure on disk for entry File, and move the content of entry to memory
  - Close (File)
    - move the content of entry File in memory to directory structure on disk
  - Write
  - Read
  - Reposition within file (e.g., fseek(), fgetpos(), fsetpos())
  - Delete
  - Truncate (e.g., truncate(), ftruncate())
  - Locking

Opening Files

- OS needs several pieces of data to manage open files:
  - File **position**
    - pointer to last read/write location
      - per each process that has the file open
  - File-open count (a.k.a. **reference 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
    - cached in memory, so it does not need to be read from file all the time
  - **Access rights**
    - per-process access mode information

The next lecture will cover the implementation details and you will work on many of them in the project #1.

Repositioning Files

```c
#include  <stdio.h>
#include <stdlib.h> // for exit

#define oops(msg, errn) { perror(msg); exit(errn); }

int main(void)
{
   FILE *file_descr;
   fpos_t pos;
   char record[20]; // for an existing file with
                    // 20 byte records

   // open file
   file_descr = fopen("test.txt", "r");
   if (file_descr == NULL)
      oops("Could not open file", 1);

   // get position of file pointer
   fgetpos(file_descr, &pos);

   // read first record
   if (!fread(record, sizeof(record), 1, file_descr))
      oops("Cannot read file", 2);
   else
      printf("1st file record: %s\n", record);

   // read second record - the value of 'pos' changes
   if (!fread(record, sizeof(record), 1, file_descr))
      oops("Cannot read file", 3);
   else
      printf("2nd file record: %s\n", record);

   // reset pointer to start of file
   // and re-read first record
   fsetpos(file_descr, &pos);
   if (!fread(record, sizeof(record), 1, file_descr))
      oops("Cannot read file", 4);
   else
      printf("1st file record again: %s\n", record);

   fclose(file_descr);
}
```

Truncating Files

```c
#include <stdio.h>  //for fopen, perror
#include <unistd.h> //for ftruncate
#include <stdlib.h> // for exit

#define oops(msg, errn) { perror(msg); exit(errn); }

int main()
{
   // without opening
   if (truncate("test.txt", 100000) == -1)
   {
      oops("Could not truncate", 1);
   }

   // when file is already opened
   FILE *file = fopen("test.txt", "r+");
   if (file == NULL)
   {
      oops("Could not open file", 1);
   }

   //do something with the contents of file
   if (ftruncate(fileno(file), 100000) == -1)
   {
      oops("Could not truncate", 2);
   }

   fclose(file);
   return 0;
}
```

Locking Files

- Open **file locking** might be provided by an operating systems and file systems
- Mediates access to a file
- Can be:
  - **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 – POSIX API

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#define oops(msg, errn) { perror(msg); exit(errn); }
int main(int argc, char *argv[])
{
   /* see: man fcntl */
   struct flock filelock = { 0, 0, 0, F_WRLCK, SEEK_SET };

   int fd;
   filelock.l_pid = getpid();
   if (argc > 1)
      filelock.l_type = F_RDLCK;

   if ((fd = open("test.txt", O_RDWR)) == -1)
      oops("open", 1);

   printf("Press <RETURN> to try to get lock: ");
   getchar();
   printf("Trying to get lock...");

   if (fcntl(fd, F_SETLK, &filelock) == -1)
      oops("fcntl", 1);

   printf("got lock\n");
   printf("Press <RETURN> to release lock: ");
   getchar();

   filelock.l_type = F_UNLCK;  /* set to unlock */

   if (fcntl(fd, F_SETLK, &filelock) == -1)
      oops("fcntl", 1);

   printf("Unlocked.\n");
   close(fd);
}
```

NOTES:
- possible to lock part of a file

```
$ man fcntl
```

- there are other ways to lock

```
$ man flock
```

- in pthreads:

```
$ man flockfile
```

Terminal 1

```
$ ./flock
Press <RETURN> to try to get lock:
Trying to get lock...got lock
Press <RETURN> to release lock:


Unlocked.
$
```

Terminal 2

```
$ ./flock
Press <RETURN> to try to get lock:
fcntl: Resource temporarily unavailable
Trying to get lock…


$ ./flock
Press <RETURN> to try to get lock:
Trying to get lock...got lock
Press <RETURN> to release lock:
Unlocked.
```

File Types – Name, Extension

- Plenty of file types

  http://en.wikipedia.org/wiki/List_of_file_formats

- Some systems use file extensions to indicate file type
  - e.g., Windows
- Others, have internal meta-data associated with files
  - Unix
    - magic number indicates the type
      - stored at the beginning of a file
  - macOS
    - keeps a reference to the creating application
      - can be re-assigned

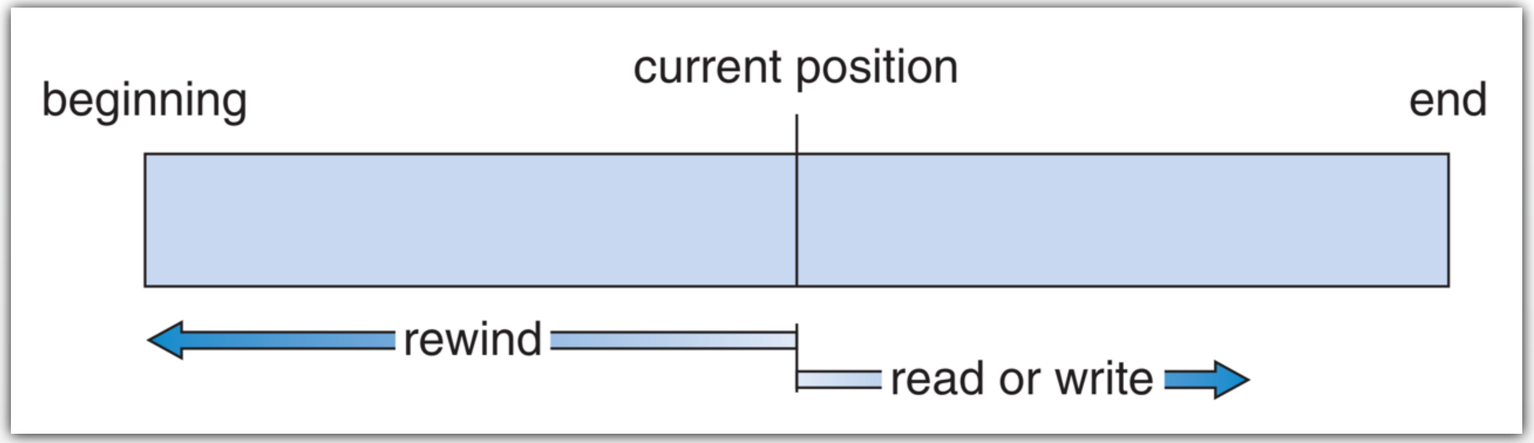
DIGRESSION:
To keep everybody on the same page on the Internet we use MIME

http://en.wikipedia.org/wiki/MIME

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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Logical Access Methods

- **Sequential Access**
  - read next
  - write next
  - rewind
  - reset
  - rewrite



- **Direct Access**
  - read n
  - write n
  - go to n
    - set pointer to n

block number == index relative to the beginning of the file

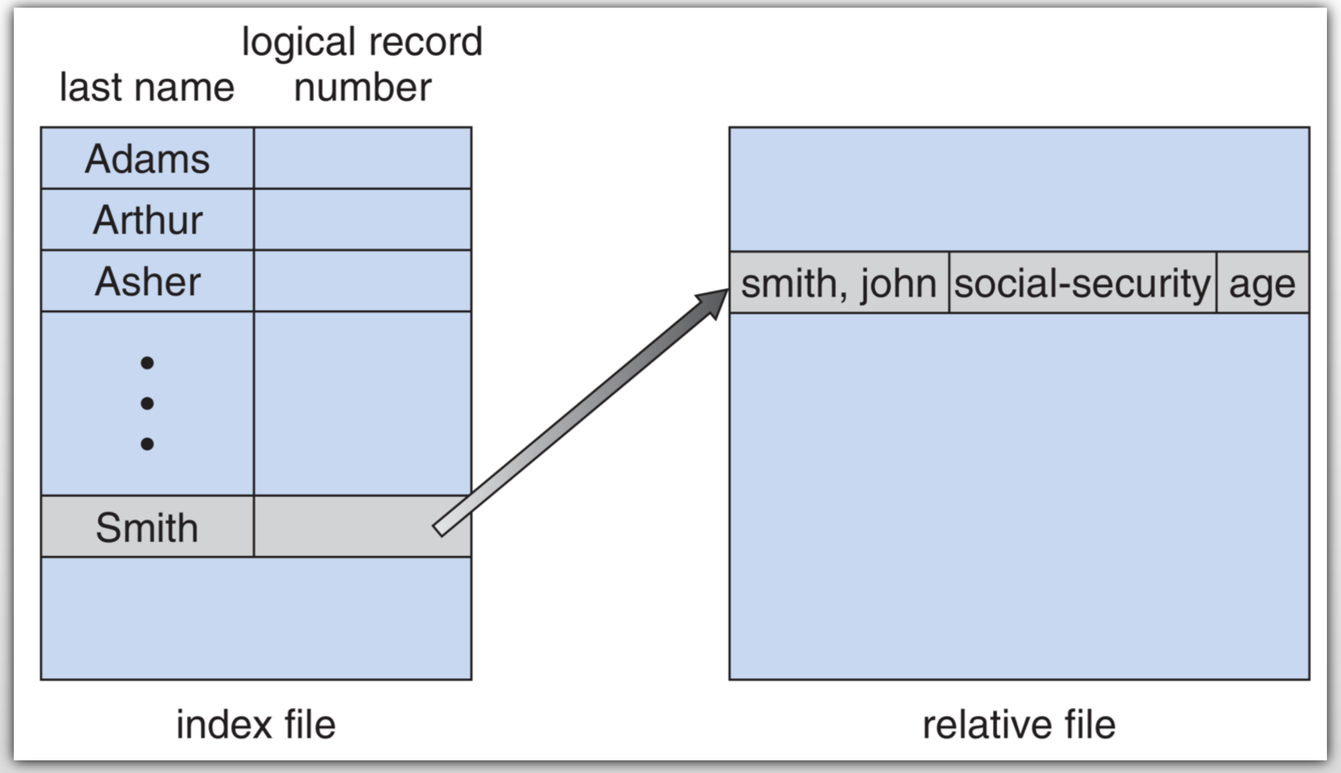
- We can easily simulate sequential access using direct access

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Index Files

- More complex organization may include an **index table**
  - requires a **key -> index** mapping
    - e.g., name in the example
  - can use binary search to find entries quickly
  - still: logical!
    - i.e., independent of the implementation

- Size of the table might be an issue
  - they need to be in memory for speed
  - solution: **multi-level indexing**
  - two-level:
    - two keys necessary; e.g.,
      - level 1: last name
      - level 2: first name
  - multiple levels
    - e.g., a level for each character in a name

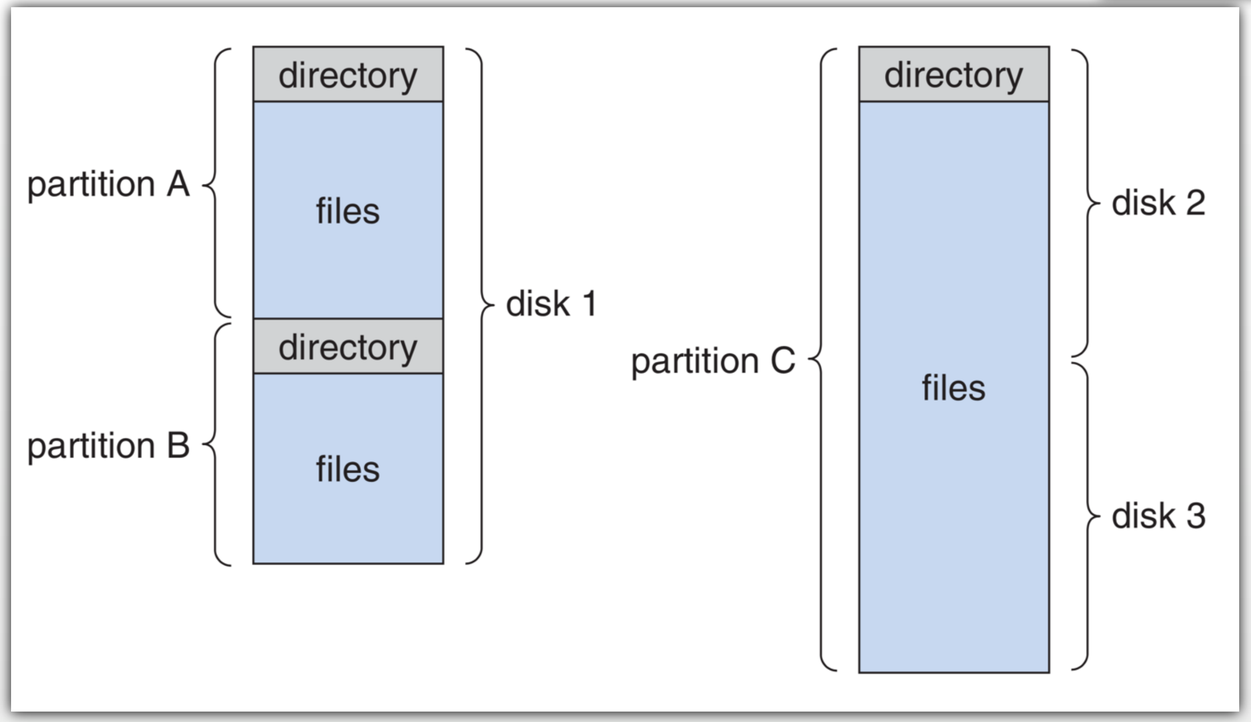


Directory Structure

- Every file system has a concept of a **directory** (a.k.a. **folder**)
  - a data structure (e.g., a collection of nodes) containing information about all files

NOTE:
pay attention to the distinction between the directory and a folder in the project!

- Operations on directories
  - Search for a file
  - Create a file
  - Delete a file
  - List the content of a directory
  - Rename a file
  - Traverse the file system



- Both the directory structure and the files reside on disk
  - the directory information might be scattered
    - e.g., in zfs directories are created “on-fly” in memory (you will do the same in the project)

Directory Access – POSIX API

```c
#include <stdio.h>
#include <dirent.h>
#include <errno.h>
#include <stdlib.h>

#define oops(msg, errn) { perror(msg); exit(errn); }

int main(int argc, char *argv[])
{
   DIR *dir;
   struct dirent *dir_entry;

   if ((dir = opendir(".")) == NULL)
      oops("Cannot open the directory.", 1);

   printf("DIRECTORY LISTING:\n");
   while ((dir_entry = readdir(dir)) != NULL)
   {
      printf("%-6s %s\n", "Name:", dir_entry->d_name);
      printf("%-7s", "Type:");
      switch(dir_entry->d_type)
      {
         default:
         case DT_UNKNOWN:
            printf("UNKNOWN\n");
            break;
         case DT_FIFO:
            printf("FIFO\n");
            break;
         case DT_CHR:
            printf("DT_CHR\n");
            break;
         case DT_DIR:
            printf("DT_DIR\n");
            break;
         case DT_BLK:
            printf("DT_BLK\n");
            break;
         case DT_REG:
            printf("DT_REG\n");
            break;
         case DT_LNK:
            printf("DT_LNK\n");
            break;
         case DT_SOCK:
            printf("DT_SOCK\n");
            break;
         case DT_WHT:
            printf("DT_WHT\n");
            break;
      }
   }

   closedir(dir);
}
```

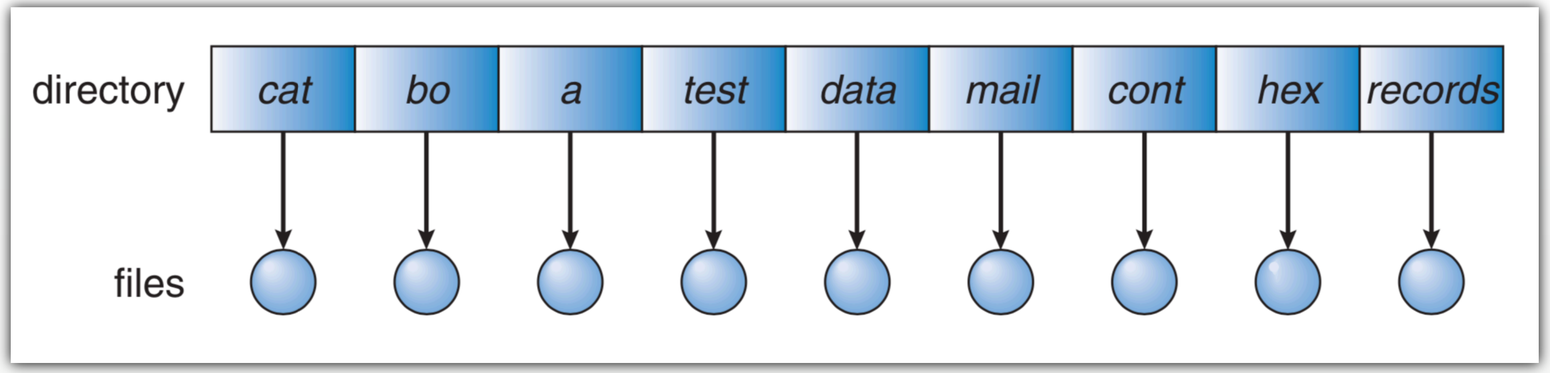
```
DIRECTORY LISTING:
Name:   .
Type:   DT_DIR
Name:   ..
Type:   DT_DIR
Name:   dir
Type:   DT_REG
Name:   flock
Type:   DT_REG
Name:   fstats
Type:   DT_REG
Name:   test.txt
Type:   DT_REG
```

Directory Organization

- We want to organize the directory (logically) to optimize:
  - 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
        - aliases (links in Unix jargon)
  - Grouping
    - logical grouping of files by properties
      - e.g., all Java programs, all games, …

Single-Level Directory

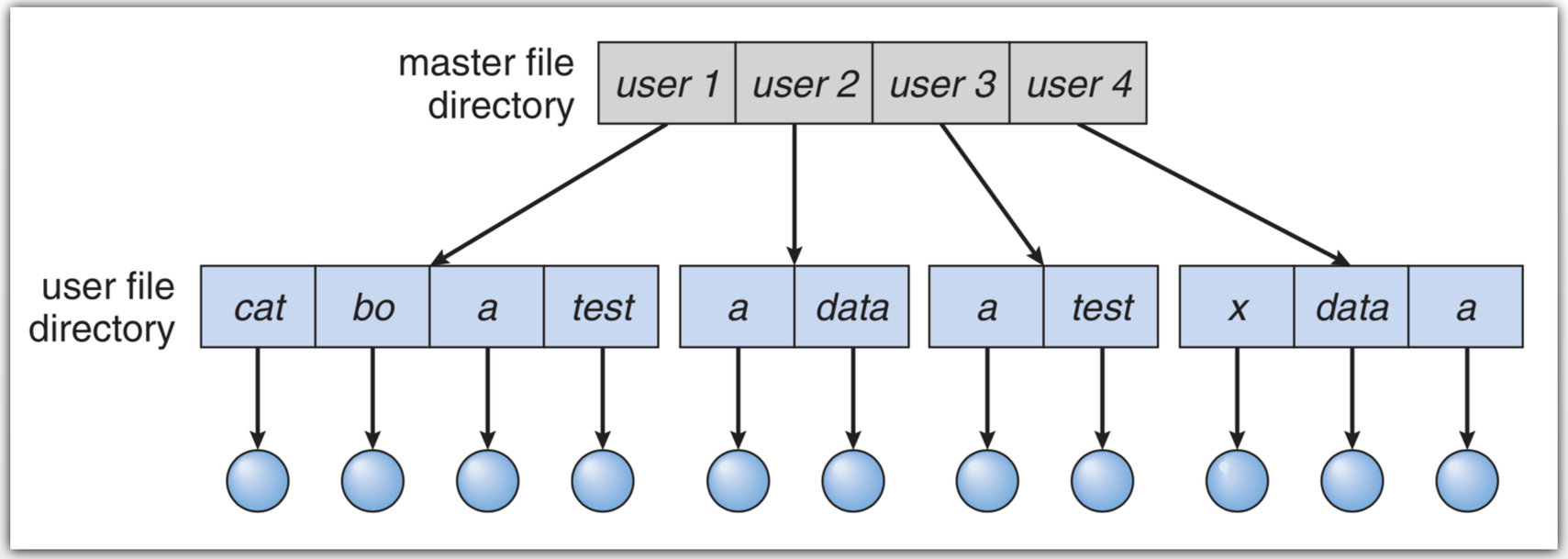
- A single directory for all users



- Many problems with:
  - searching
    - plenty of files
  - naming
    - no duplicates
  - grouping
    - no support other as by name
  - security
    - shared space

Two-Level Directory Structure

- Separate directory for each user

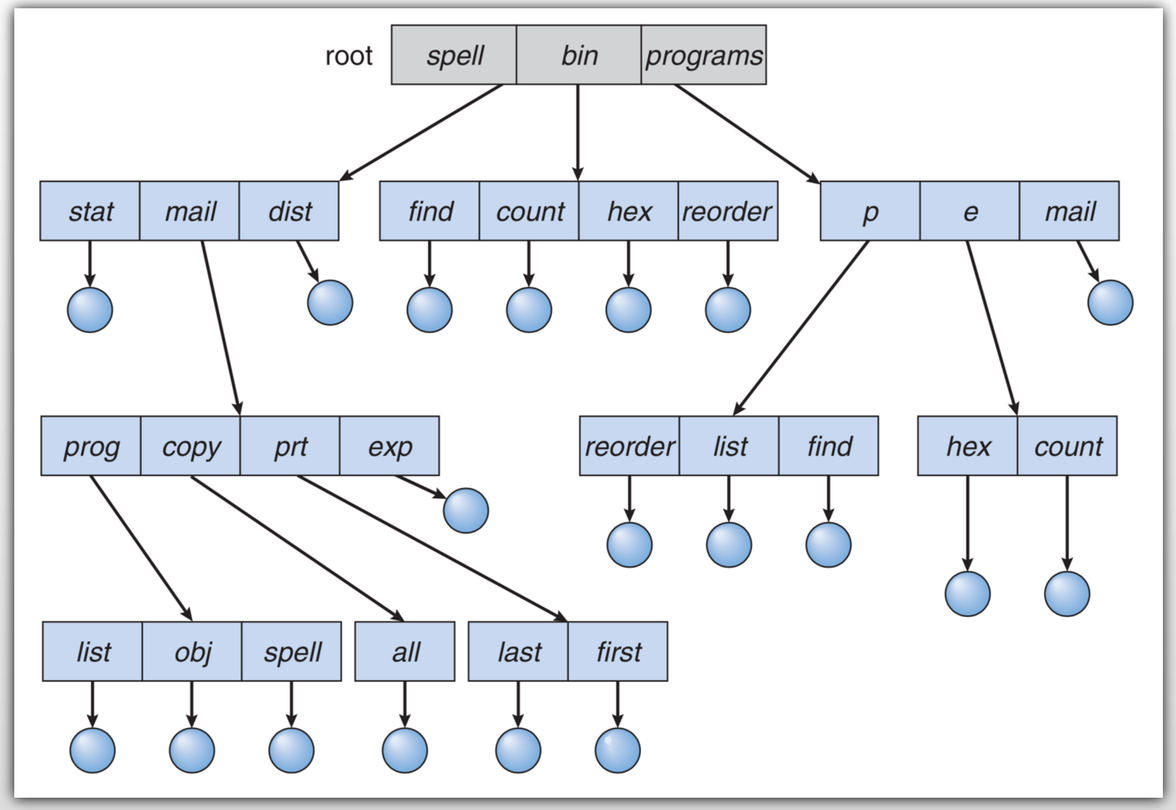


- Use a path name rather than just a file name
- Can have the same file name for different user
- Efficient searching per user

- ...but, still not enough flexibility
  - e.g., no grouping capability

Tree-Structured Directories

- One root
- Efficient searching
- Grouping capability



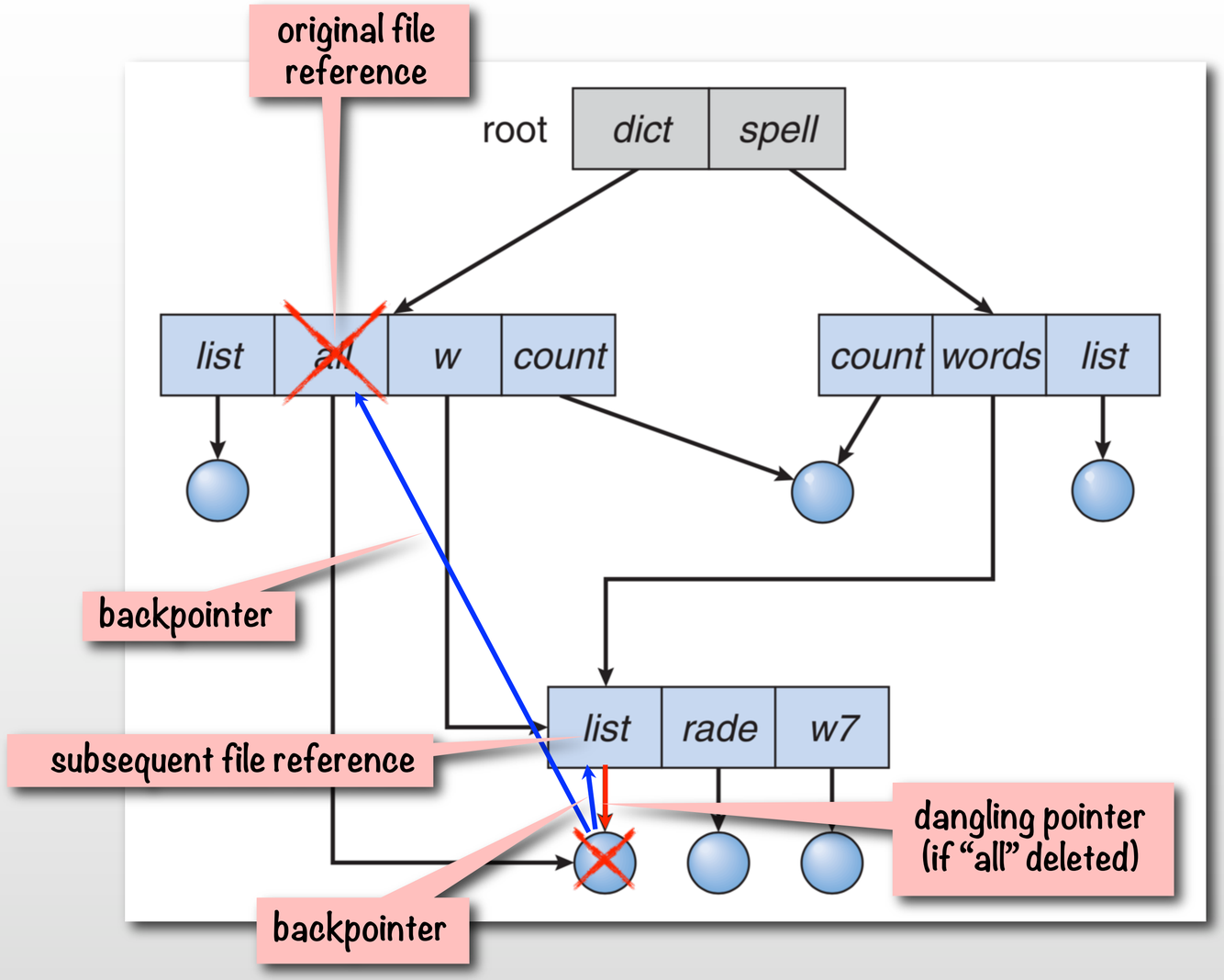
- Current **(working) directory** concept associated with every process
  - set to user's **home directory** though the login shell
    - each user session starts in here
  - inherited by all children process
    - including all programs started from the user shell

```
$ pwd
/home/aj
$ cd Desktop
$ pwd
/home/aj/Desktop
$ bash
$ pwd
/home/aj/Desktop
$ exit
$
```

entering child shell

Acyclic-Graph Directories

- Allow for shared subdirectories and files
  - two different names (**aliasing**)
- Problems!
  - e.g., if all and list are aliases (see the figure) and dict deletes all then list points to nowhere ⇒ **dangling pointer**
- Solutions:
  - **backpointers**, so we can delete all pointers (symbolic links)
    - adding backpointers makes the size of the supporting file data structure variable
      - not good, since fixed-size structures simplify access
        - could apply daisy chained backpointers
  - entry-hold-count (**reference count**)
    - add a counter that indicates the number of active pointers to a file
    - much better!



Symbolic Links (Aliases) in Unix

- New directory entry type
  - link – another name (pointer) to an existing file (alias)
  - resolve the link – follow pointer to the directory to locate the file

```
$ cat > fileToLink.txt
This is a file to link.<Cntl-D>
$ ls -la
total 8
drwxr-xr-x   3 aj  aj   102 Nov  4 13:58 .
drwxr-xr-x@ 96 aj  aj  3264 Nov  4 13:57 ..
-rw-r--r--   1 aj  aj    24 Nov  4 13:57 fileToLink.txt
$ cat fileToLink.txt
This is a file to link.
$ ln -s fileToLink.txt linkedFile.txt
$ ls -la
total 16
drwxr-xr-x   4 aj  aj   136 Nov  4 13:58 .
drwxr-xr-x@ 96 aj  aj  3264 Nov  4 13:57 ..
-rw-r--r--   1 aj  aj    24 Nov  4 13:57 fileToLink.txt
lrwxr-xr-x   1 aj  aj    14 Nov  4 13:58 linkedFile.txt -> fileToLink.txt
$ cat linkedFile.txt
This is a file to link.
$ rm fileToLink.txt
remove fileToLink.txt? y
$ ls -la
total 8
drwxr-xr-x   3 aj  aj   102 Nov  4 14:00 .
drwxr-xr-x@ 96 aj  aj  3264 Nov  4 13:57 ..
lrwxr-xr-x   1 aj  aj    14 Nov  4 13:58 linkedFile.txt -> fileToLink.txt
```

Houston, we have a probem!

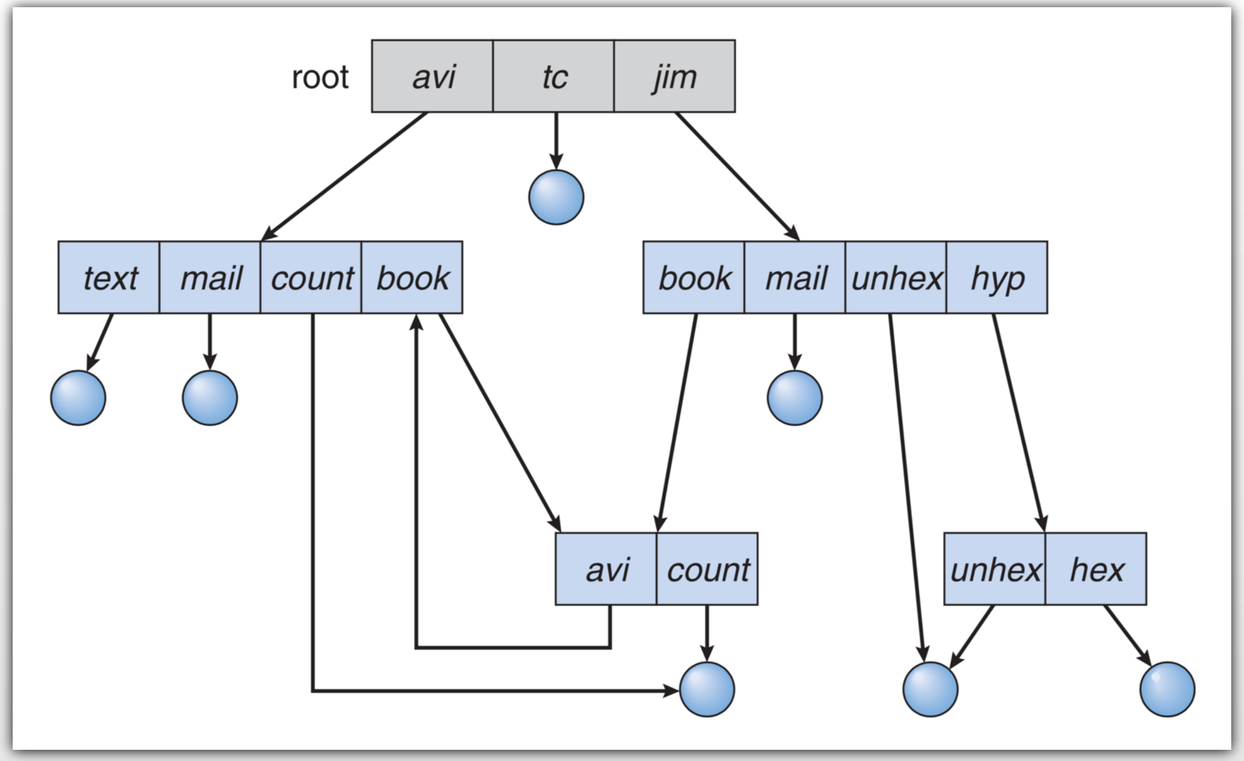
Hard Links in Unix

- Point to the same file content, but have fully-fledged directory entries
  - hard link count shows how many links exist
  - the file is not removed until the hard link count goes to zero

```
$ ls -li
total 8
27121954 -rw-r--r--@ 1 aj  staff  28 Mar 29 18:09 fileToLink.txt
$ cat fileToLink.txt
This is a file to link to.
$ ln -s fileToLink.txt softLinkedFile.txt
$ ln fileToLink.txt hardLinkedFile.txt
$ ls -li
total 24
27121954 -rw-r--r--@ 2 aj  staff  28 Mar 29 18:09 fileToLink.txt
27121954 -rw-r--r--@ 2 aj  staff  28 Mar 29 18:09 hardLinkedFile.txt
27123651 lrwxr-xr-x  1 aj  staff  14 Mar 29 18:18 softLinkedFile.txt ->
fileToLink.txt
$ find . -inum 27121954
./fileToLink.txt
./hardLinkedFile.txt
$ rm fileToLink.txt
$ ls -li
total 16
27121954 -rw-r--r--@ 1 aj  staff  28 Mar 29 18:09 hardLinkedFile.txt
27123651 lrwxr-xr-x  1 aj  staff  14 Mar 29 18:18 softLinkedFile.txt ->
fileToLink.txt
$ cat hardLinkedFile.txt
This is a file to link to.
$ cat softLinkedFile.txt
cat: softLinkedFile.txt: No such file or directory
```

General Graph Directory

- We allow for any graph to represent a directory
- Problem with possibility of introducing cycles
- How can we guarantee no cycles?
  - allow only links to files, and not directories
    - files are terminal nodes in the graph (still problem with dangling pointers)
  - every time a new link is added use a cycle detection algorithm to determine whether it is OK
    - expensive!



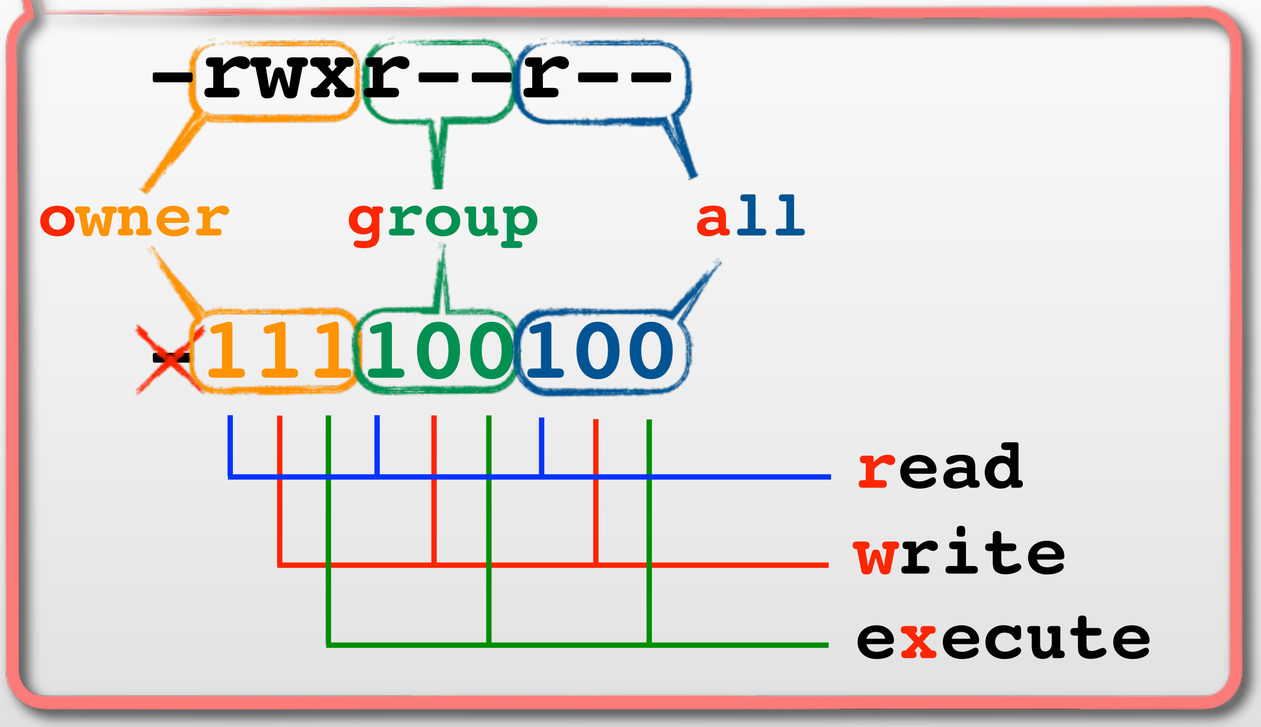
File Protection (POSIX)

- Sharing of files on multi-user systems is desirable
  - but it must be done through a protection scheme
- User IDs
  - identify users, allowing permissions and protections to be per-user
- Group IDs
  - allow users to be in groups, permitting group access rights
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

UNIX Access Lists and Groups

- On POSIX systems, each file has
  - access mode
    - read (r), write (w), execute (x)
  - three classes of users
    - owner
    - group
    - other (world / all)

```
$ ls -la
-rw-r--r--   1 aj   aj      0 Nov   4 14:26 test.file
```



Example

NOTE: **0022** octal is
000010010 binary
(first 0 – in red - indicates octal notation)

HOW?

starting with the highest permission level of 777 zero out all bits corresponding to 1s in umask (**xor**)

```
111111111
000010010
111101101
rwxr-xr-x
```

```
$ umask
0022
$ umask -S
u=rwx,g=rx,o=rx
$ touch test_1.file
$ ls -l
total 0
-rw-r--r--  1 aj  staff  0 Apr 11 15:38 test_1.file
$ chmod go-r test_1.file
$ ls -l
total 0
-rw-------  1 aj  staff  0 Apr 11 15:38 test_1.file
$ chmod a+rwx test_1.file
$ ls -l
total 0
-rwxrwxrwx  1 aj  staff  0 Apr 11 15:38 test_1.file
$ chmod 600 test_1.file
$ ls -l
total 0
-rw-------  1 aj  staff  0 Apr 11 15:38 test_1.file
$ umask 0057
$ umask -S
u=rwx,g=w,o=
$ touch test_2.file
$ ls -l
total 0
-rw-------  1 aj  staff  0 Apr 11 15:38 test_1.file
-rw--w----  1 aj  staff  0 Apr 11 15:40 test_2.file
```

determines the default permissions for creating new files

touching a file just creates it, or modifies the access date

take (**-**) away **r**eading permission from **g**roup and **o**thers

add (**+**) **r**eading, **w**riting, e**x**ecuting permissions to **a**ll

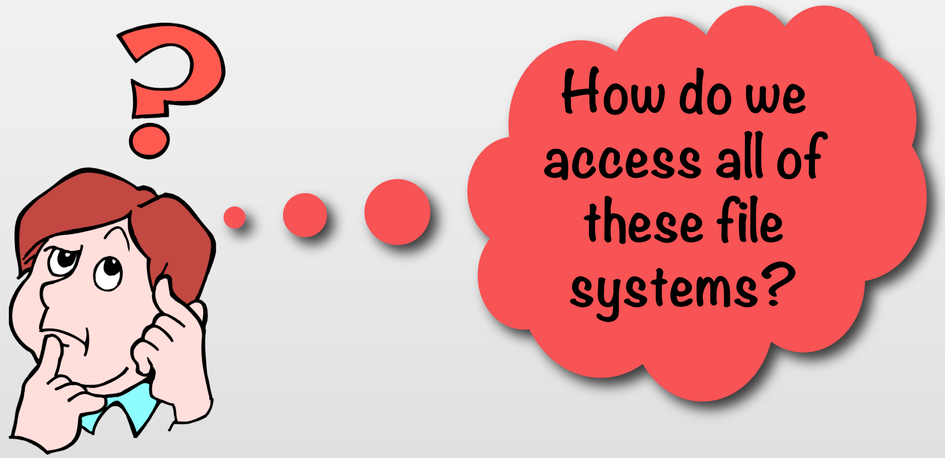
change the default mask

created with a different mask in effect

File Systems

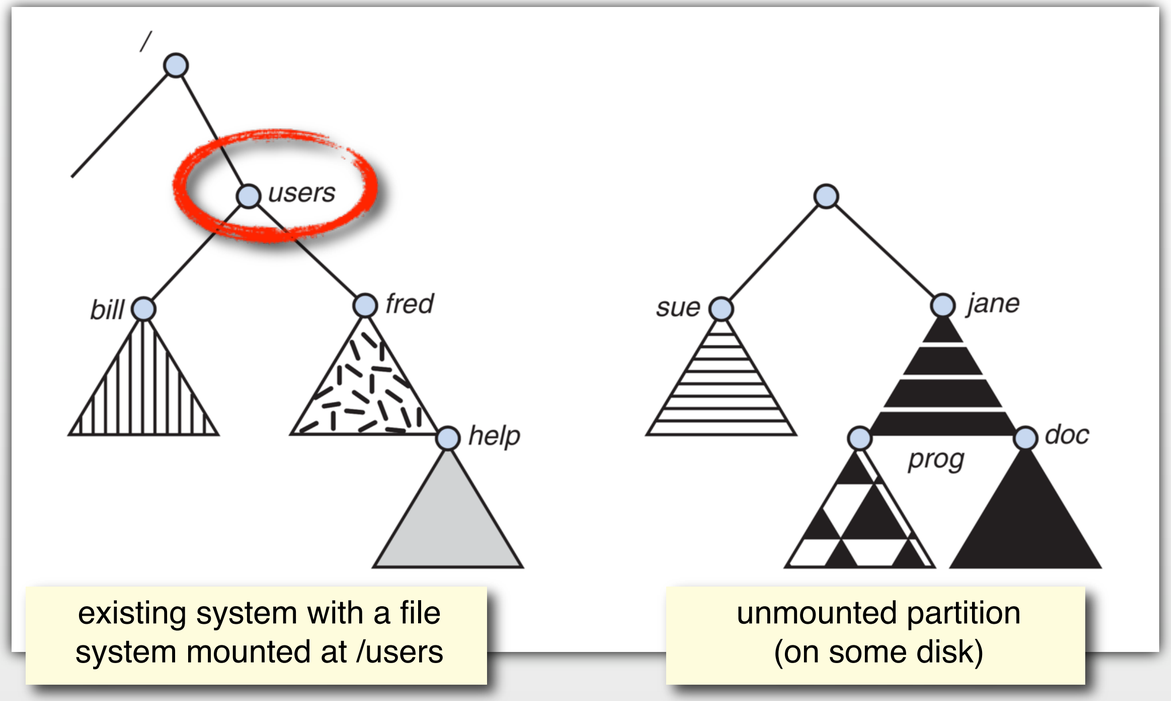
- Plenty of file system types
  - improvements through natural evolution
    - providing backward compatibility would lead to very inefficient file systems
  - protection and security necessities
  - specialization

http://en.wikipedia.org/wiki/List_of_file_systems



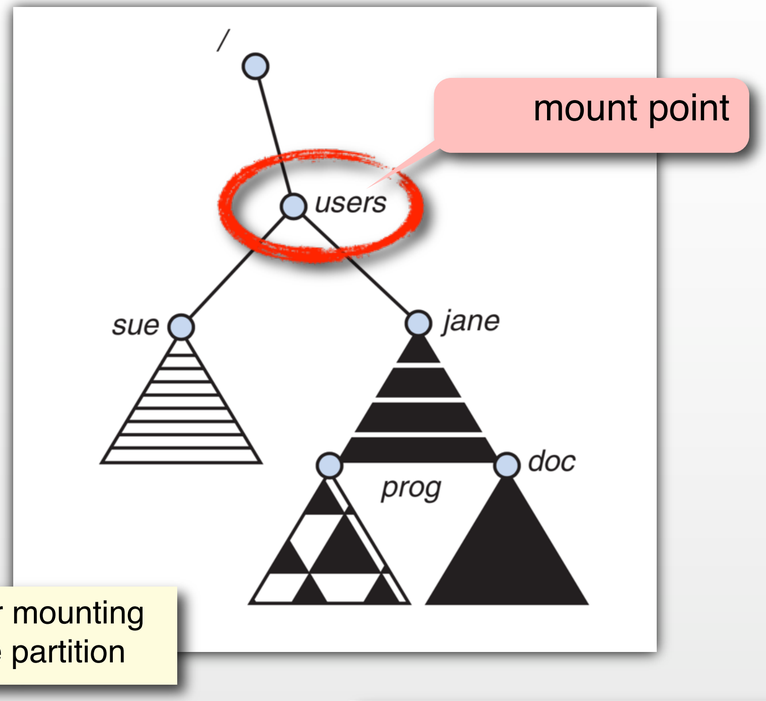
File System Mounting

- A file system must be mounted before it can be accessed



existing system with a file system mounted at /users

unmounted partition (on some disk)



after mounting the partition

- Many systems do it automatically nowadays
  - users can do it manually

```
$ mount
/dev/disk0s2 on / (hfs, local, journaled)
devfs on /dev (devfs, local)
fdesc on /dev (fdesc, union)
/dev/disk1s2 on /Volumes/AJsTimeMachine (hfs, local, nodev, nosuid, journaled)
/dev/disk2s1 on /Volumes/AJsUSBstick (hfs, local, nodev, nosuid, journaled, noowners)
$ sudo umount /Volumes/AJsUSBstick
```

you will use it in the lab to mount your own file system, so try **man mount**

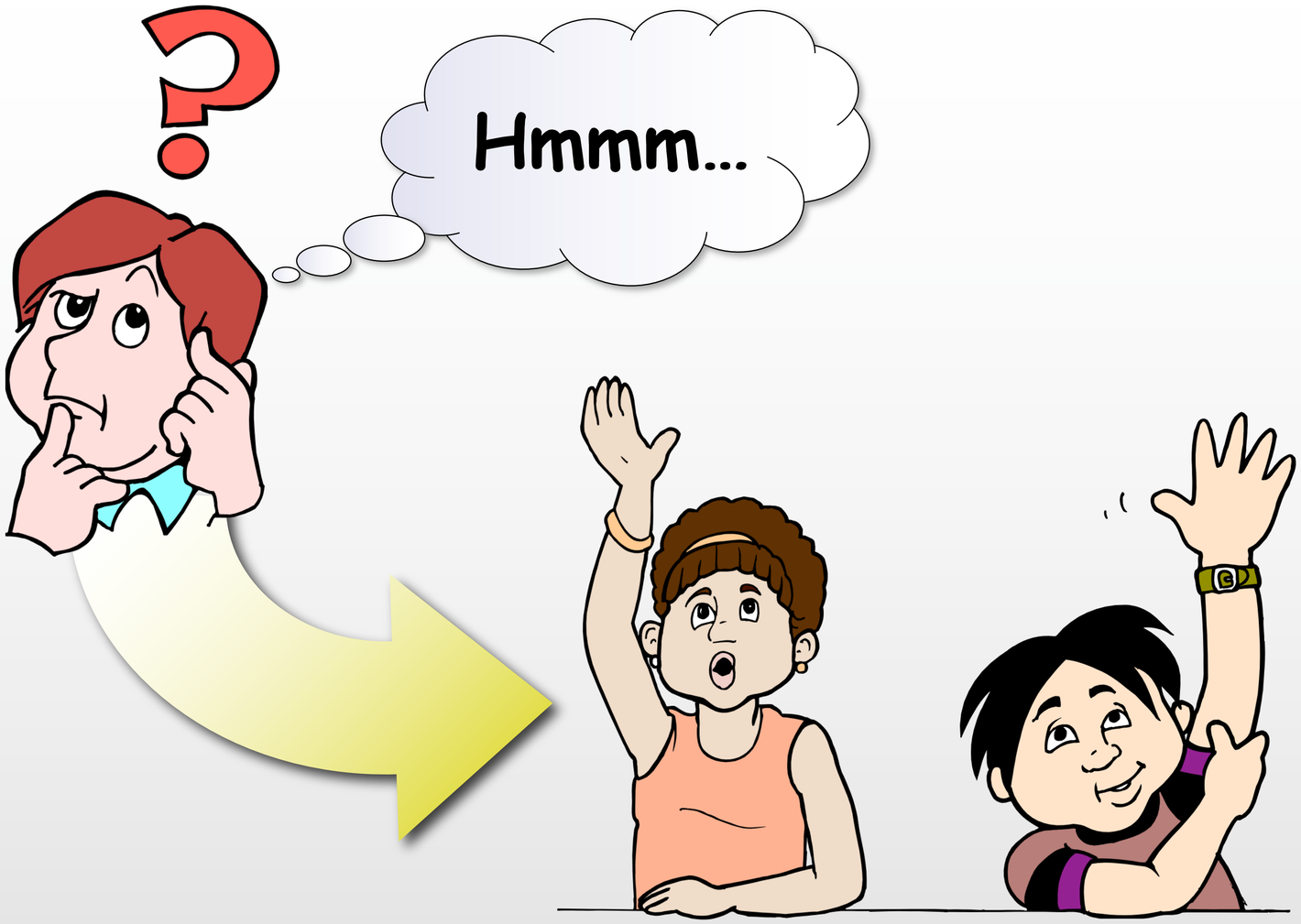
Network File Systems

- Files may be shared across networks
- **Network File System** (**NFS**) was the first a common distributed file-sharing method
- 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
  - **SMB** is an open source implementation of CIFS for non-Windows systems
  - **AFP** is standard Mac protocol
  - after mounting a network file system, all standard calls referencing files on the system are translated into remote calls

- Distributed Information Systems (distributed naming services)
  - implement unified access to information needed for remote computing
  - **LDAP** (**Lightweight Directory Access Protocol**; common Unix standard)
    - Open Directory (macOS implementation of LDAP)
  - **NIS** (**Network Information System**; new version of Sun's Yellow Pages)
  - **Active Directory** (Windows; based on LDAP)

Remote File System Issues

- Sharing files
  - remote file systems add new failure modes
    - due to network failure, server failure
  - recovery from failures may 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 (poor security; e.g., passwords in clear text!)
- **Consistency semantics**
  - specify how multiple users are to access a shared file simultaneously
  - similar to process synchronization algorithms
    - tend to be less complex due to disk I/O and network latency (for remote file systems)
  - **Andrew File System** (**AFS**) implements complex remote file sharing semantics
    - AFS has session semantics
      - writes only visible to others starting after the file is closed (on commits)
      - In contrast, **Unix File System** (**UFS**) - a precursor for many other Unix file systems like ext4 in Linux - implemented:
        - 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



COMP362 Operating Systems

Prof. AJ Bieszczad