"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

- C.A.R. Hoare

(British computer scientist, winner of the 1980 Turing Award)

File-System Interface

These slides were compiled from the OSC textbook slides (Silberschatz, Galvin, and Gagne) and the instructor's class materials.

What is a File?

- Nonvolatile storage unit
- *Logically* contiguous space
- File format
  - Sequence of bits: OS view point
  - Meaningful information: application view point
    - Types identified by a file suffix (extension)

- Attributes
  - Name, Type, Size, Protection, Time, Date, and User ID
  - Location
    - Directory from a user's point of view
    - Disk location from the OS view point

File-System

The file-system resides permanently on *secondary storage*, which is designed to hold a large amount of data permanently.

File Attributes

- **Name** – used for a user to reference a file.
- **Type** – needed for an application to identify if it is reading correct file information.
- **Location** – Directory path (and disk location).
- **Size** – current file size.
- **Protection** – controls who can do reading, writing, executing.
- **Time, date, and user identification** – data for protection, security, and usage monitoring.

Directory (or folder) provides a user with such file information

File Types

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

The OS needs to recognize the type of file in order to operate on it in the appropriate way

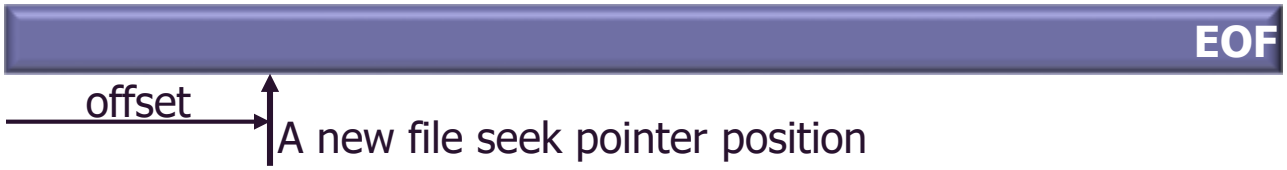
File Operations

Operations	Descriptions	Unix	Our ThreadOS
Create	Create a file with its attributes	`creat(filename,mode)`	N/A
Open	Open the specified file. (Create it if mode specified and necessary)	`open(filename, flag,mode)`	`SysLib.open(filename,mode)`
Read	Read from a file	`read(fd, buf, size)`	`SysLib.read(fd, buffer)`
Write	Write a file	`write(fd, buf, size)`	`SysLib.write(fd, buffer)`
Seek	Reposition a file access point	`lseek(fd, offset, origin)`	`SysLib.seek(fd, offset, whence)`
Close	Close the file specified with fd	`close(fd)`	`SysLib.close(fd)`
Delete	Destroy the specified file	`remove(filename)`	`SysLib.delete(filename)`
Truncate	Erase the file contents but keep its attributes, (e.g. name)	`truncate(filename, length)`	N/A
Status	Returns the specified file status	`stat(fd, statbuf)`	`SysLib.fsize(fd)`
Format	Format the disk	N/A	`SysLib.format(files)`

seek (fseek / lseek)

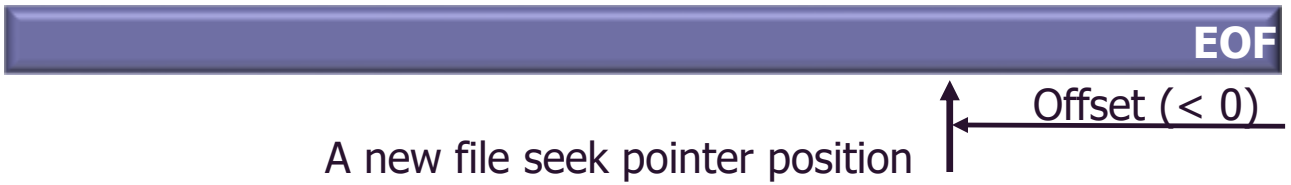
```
seek( int fd, int offset, int whence )
```

- whence == 0 (SEEK_SET)



- whence == 1 (SEEK_CUR)

- whence == 2 (SEEK_END)



File Locking
Exclusive/Shared Locks

```c
#include <sys/file.h> // for flock(2)
#include <sys/stat.h> // for S_* constants
#include <string.h>   // for strerror(3) prototype
#include <stdio.h>    // for fprintf(3),printf(3),stderr protype
#include <errno.h>    // for errno prototype
#include <unistd.h>   // for close(2) prototypes
#include <iostream>   // for C++ cin and cout
#define  FILENAME "/tmp/flock.example"

using namespace std;

int main(int argc,char **argv) {
   int fd; char buf;
   fd = open(FILENAME,O_RDWR|O_CREAT,S_IRUSR|S_IWUSR);

   // Aquire an exclusive lock
   if (flock(fd,LOCK_EX) == -1)
      return -1;
   cout << "Press enter to release the lock." << endl;
   cin >> buf;

   // Release the exclusive lock
   if (flock(fd,LOCK_UN)==-1)
      return -1;
   printf("Released!\n");
   if(close(fd)==-1)
      return 0;
}
```

U-read

Lock a file itself
(but not a file descriptor)

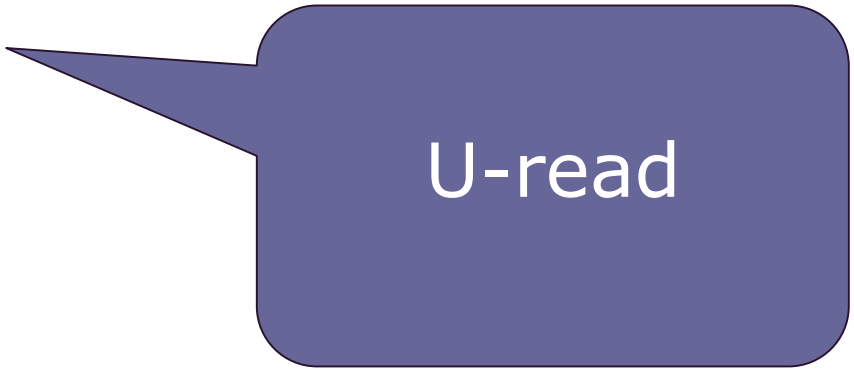
Unix flock

From: http://www.wlug.org.nz/

Java File locking

```
FileLock lock(long begin, long end, boolean shared)
```

- Requires the `FileChannel` for the file to be locked.
- `lock()` method of the FileChannel is used to acquire the lock.
- begin and end are the beginning and ending positions of the region being locked.
- shared = true for shared locks; false exclusive lock
- lock is released by the `release()` method of `FileLock`



File Locking
Exclusive/Shared Locks

```
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();
      }
   }
}
```

Java Filelock

U-read

Akin to a reader lock

Lock a portion of
a random access file object

Memory Mapped files

Bringing a file into memory

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#define PACKAGE "mmap"

int main(int argc, char *argv[]) {
   int input, output;
   size_t filesize;
   void *source, *target;

   if((input = open(argv[1], O_RDONLY)) == -1) exit(-1);
   if((output = open(argv[2], O_RDWR|O_CREAT|O_TRUNC, 0666)) == -1) exit(-1);

   if((source = mmap(0, filesize, PROT_READ, MAP_SHARED, input, 0)) == (void *) -1) exit(-1);
   if((target = mmap(0, filesize, PROT_WRITE, MAP_SHARED, output, 0)) == (void *) -1) exit(-1);

   memcpy(target, source, filesize);
   munmap(source, filesize);
   munmap(target, filesize);

   close(input);
   close(output);
   return 0;
}
```

From: http://w nux/

U-read

Memory Mapped files

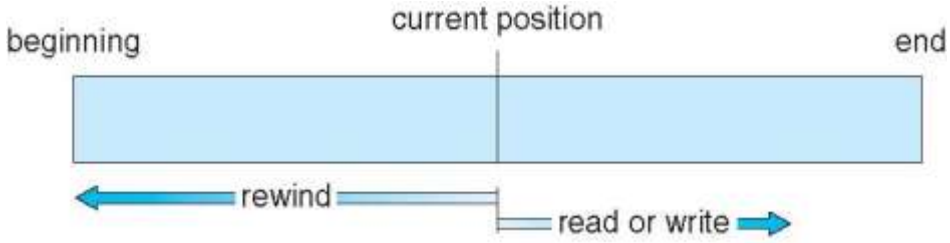
Java example

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class MemoryMapReadOnly {
   / Assume page size of 4K
   public static final int PAGE_SIZE = 4096;

   public static void main (String args[]) throws IOException {
      RandomAccessFile inFile = new RandomAccessFile(args[0], "r

      FileChannel in = inFile.getChannel();
      MappedByteBuffer mappedBuffer = in.map(FileChannel.MapMode.READ_ONLY, 0, in.size());
      long numPages = in.size() / (long)PAGE_SIZE;
      if( (in.size() % PAGE_SIZE) > 0)  ++numPages;

      / we will "touch" the first byte of every page
      int position = 0;
      for (long i=0; i < numPages; i++) {
         byte item = mappedBuffer.get(position);
         position += PAGE_SIZE;
      }
      in.close();
      inFile.close();
   }
}
```

U-read

Access Methods

- Sequential Access
  ```
  read next
  write next
  reset
  no read after last write
       (rewrite)
  ```



U-read

- Direct Access
  ```
  read n
  write n
  position to n
     read next
      write next
  rewrite n
  ```

n = relative block number

Directory and Disk Structure

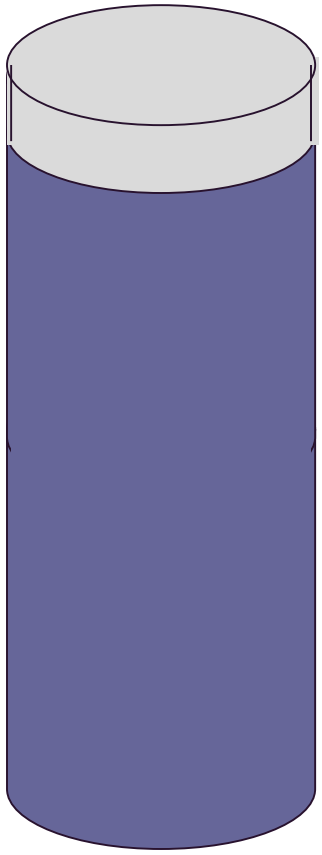
Device Directory and Disk Partition

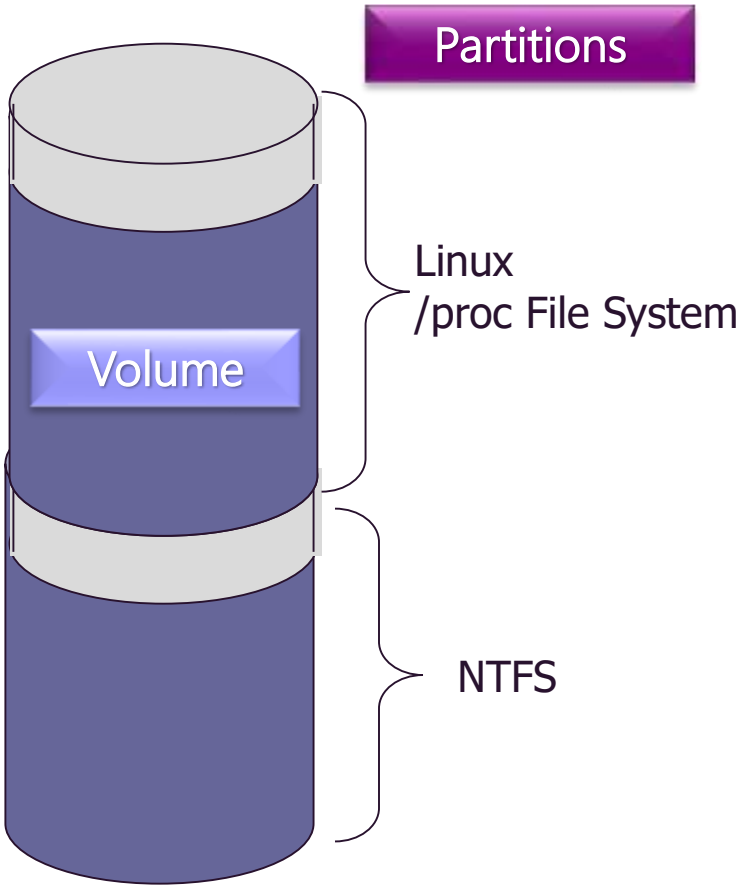
- **Device Directory**

  Super block or volume control block

  - Layout of the file system
    - File system size
    - The number of free blocks
  - Index of files stored on the device

    <fileName, fileDescriptorLocaiton>





Directory and Disk Structure

- File System
  - Millions of files stored in a computer
  - Files stored on random-access storage devices
  - Contained in a volume
  - Divide into finer-grain: e.g., partitions
    - Partitions limit the size of individual file
  - Device directory keeps information
- Directory
  - Symbol table that ***translates*** file names into directory entries

Operations Performed on Directory

- *Search* for a file
  - `find dirName –n fileName –print`
  - `whereis` or where filename
- *Create* a file
  - Manual: Create a file by text editors
  - Command: `touch fileName`… create a 0-le
  - System call: `open(filename,O_CREAT,`
- *Delete* a file
  - `rm [-fr] fileName`
- *List* a directory
  - `ls [-al]`
- *Rename* a file
  - Command: `mv oldName newName`
  - System call: `rename( oldname, newname)`
- *Traverse* the file system
  - `find / -n -print`

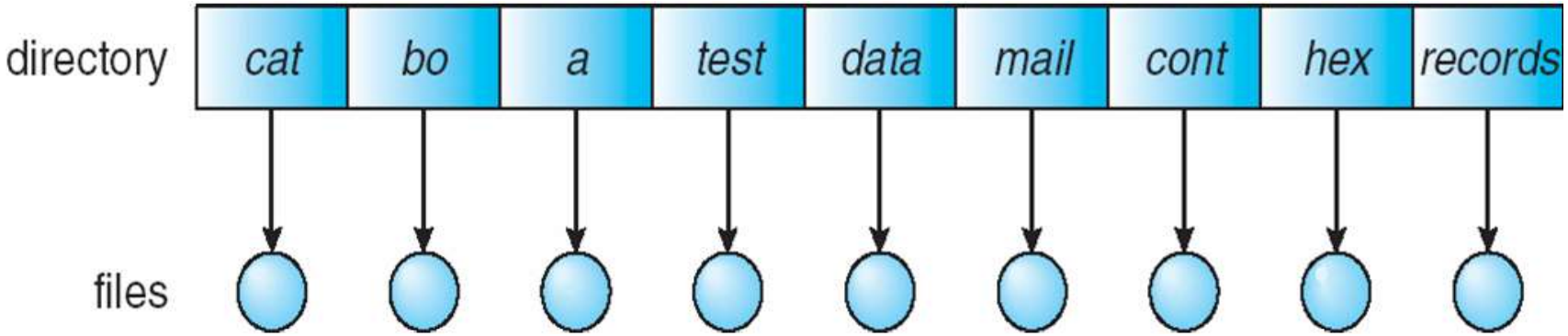
U-read

Directory Logical Structure Schemes

- Single-Level
- Two-Level
- Tree-Structure
- Acyclic-Graph

Single Level

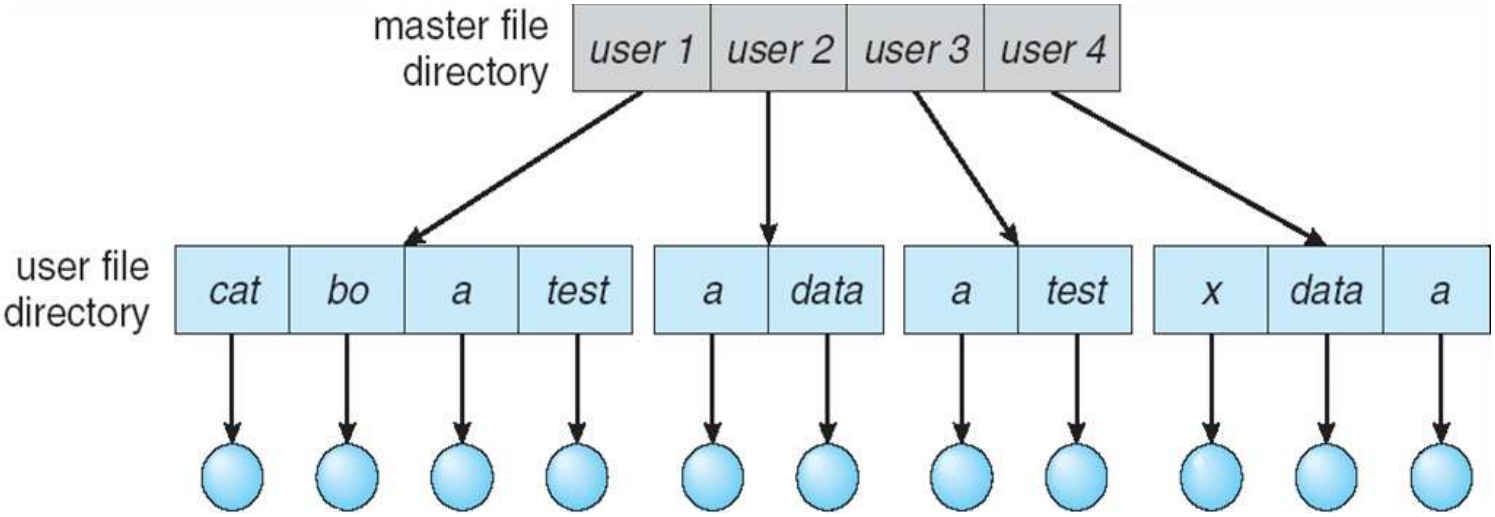
Single-Level Directory



- All files in the same directory
- Problems:
  - Naming: Files must have a unique name.
  - Grouping: All files are visible to all users.
  - What else?
- Example: CP/M

Two-Level

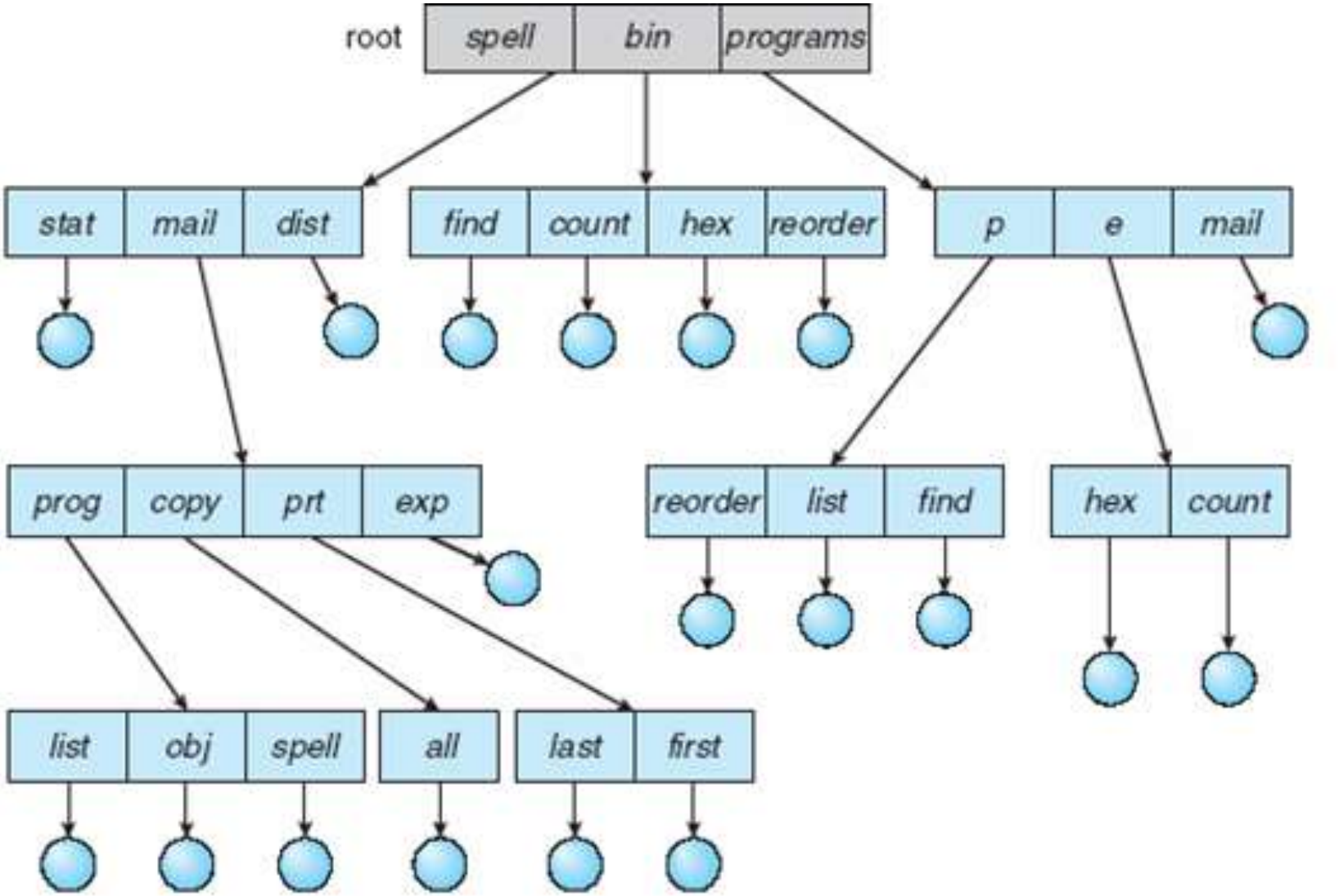
Two-Level Directory



- Each user has his/her own directory
  - Naming problems resolved
  - Special user file directories shared among users.
  - Search path needed
- Example: MVS/VM

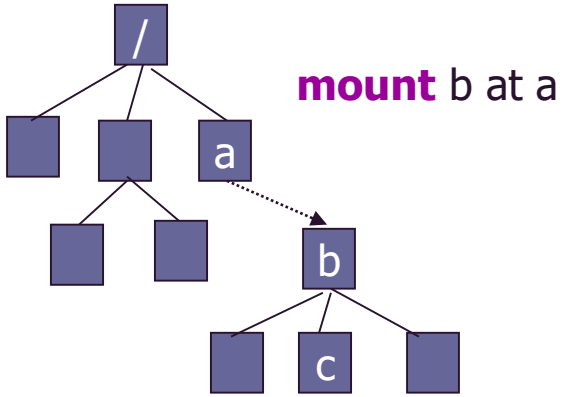
Tree Structured

Tree-Structured Directories



Tree-Structured Directories in Linux

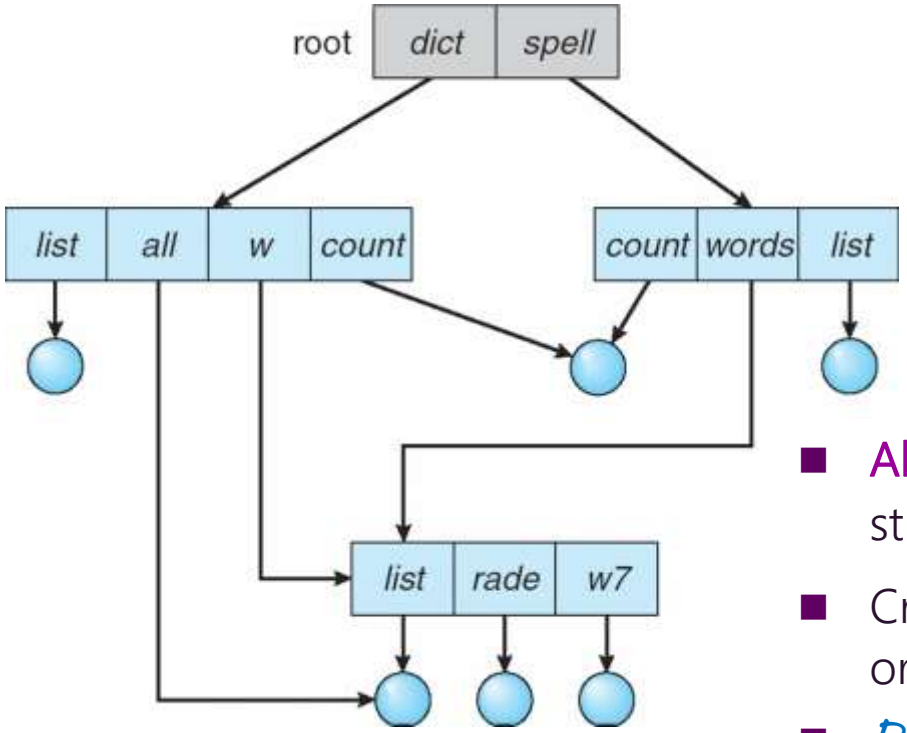
- Useful for grouping files
- *Attaching* other file systems as a subdirectory
  - mount and umount
- Current working directory
  - Files specified by a relative path
  - Process has a working directory
    - inherits to children procs
  - Subdirectories created/deleted by: `mkdir rmdir`
- Path:
  - Relative path versus absolute path
  - Setting PATH to let the shell search a command file.
- Recursive operations
  - List: `ls -R`
  - Delete: `rm -R`
  - Archive: `tar -cvf - .`



U-read

Acyclic Graph

Acyclic-Graph Directories

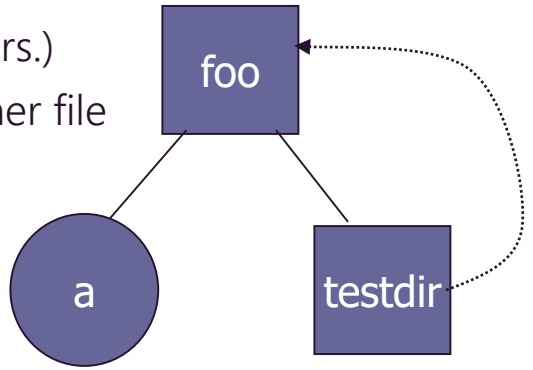


- Allow file/directory *sharing* which tree structure prohibits
- Create a `link` that is a pointer to another file or subdirectory
- *Problems*:
  - Traverse shared files more than once
  - How to **delete** shared file?
  - Ensure there are **no** cycles
- Link: shared file implementation, pointer to another file path (not a copy)
- Duplication: another implementation, problem is maintenance

Links and Functions

- Hard link
  - Refers to the specific location of physical data
  - `ln target_file_name link_name` ← file only
  - `link()` and `unlink()` system calls
    - Super-user mode only: ***avoid*** cyclic-graph directories (won't work if the super user is daft ☺)
    - When ***link count reaches 0***, the corresponding file itself is removed.

- Symbolic link
  - Serves as a reference to another file or directory (links between dirs.)
  - Refers to a symbolic path indicating the abstract location of another file
  - `ln -s target_file_name link_name`



- Linux provides two types of functions:
  - Those not following symbolic link: `chown, remove, rename, unlink`
    (To cut off an infinite loop, simply ignore symbolic links)
  - Those following symbolic link: `access, create, open, stat`
    Symbolic deletion does not affect the original file.

Hard Link Example ~~(Try it out!)~~

```
echo 'The angles got the box' > file1
ln file1 mylink

ls –l file1 mylink

   -rw-r--r-- 2 root root 15 Oct  1 15:30 file1
   -rw-r--r-- 2 root root 15 Oct  1 15:30 mylink
```

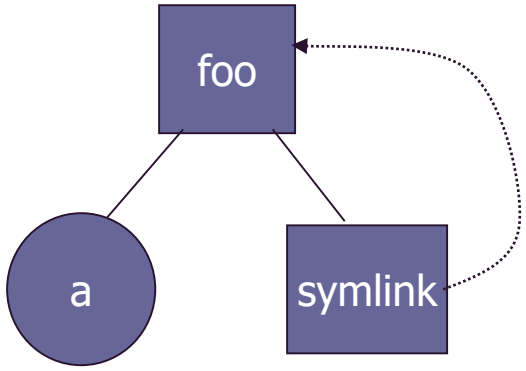
creates a file with some text in it

- The hard link is identical to the original file
- This makes it difficult to recursively traverse a directory ignoring any links
- Modern systems don't easily allow hard links on directories to prevent endless recursions

Symbolic Link Example

- Symbolic link example

```
mkdir foo
touch foo/a
ln -s /path/to/foo /path/to/symlink
ls -l foo
  -rw-rw-r—   1 chinchia 0 Oct 7 07:07 a
  lrwxrwxrwx  1 chinchia 6 Oct 7 07:07 symlink -> ../foo
cd foo
ls -l
  -rw-r--r-- 1 chinchia 0 Oct 7 07:07 a
  lrwxrwxrwx 1 chinchia 6 Oct 7 07:07 symlink -> ../foo
cd symlink
ls -l
  -rw-r--r-- 1 chinchia 0 Oct 7 07:07 a
  lrwxrwxrwx 1 chinchia 6 Oct 7 07:07 symlink -> ../foo
```



Note: This example may not work in our systems (operation not supported).

- Examples: some external links with examples

Overview of ln -s

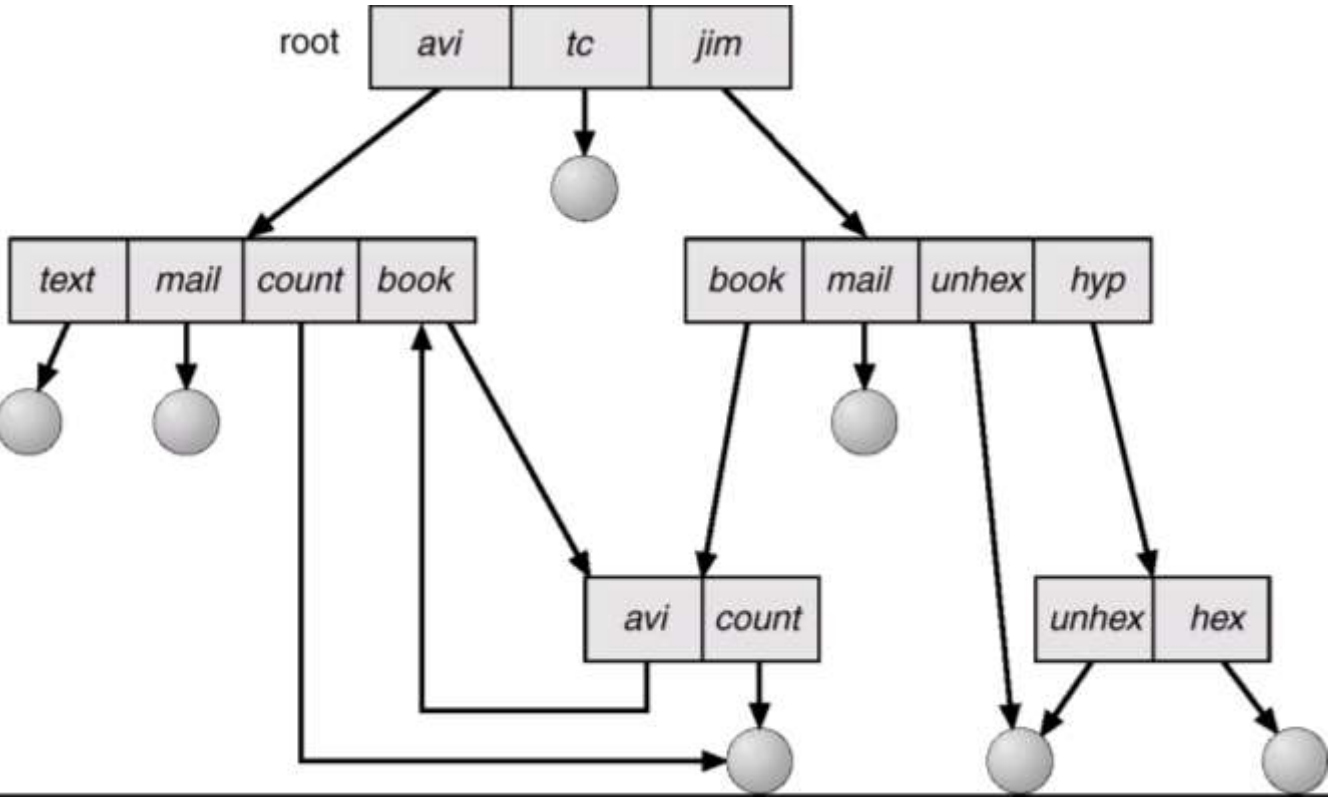
Stackoverflow discussion with some examples

General Graph

General Graph Directory

- Simple to traverse
- Cycle: reference count may ≠ 0 even when there are no references

- How do we guarantee no cycles?
  - Allow *only links to files* but not subdirectories.
  - *Garbage collection* to cut off anomaly cycles
  - Every time a new link is added, use a *cycle detection algorithm* to determine whether it is OK
- ∞ Loop risk: imit # of directories accessed during search



File Sharing

U-read!
(may be in exam)

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a protection scheme
- On distributed systems, files may be shared across a network
- Network 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-user
-
- Group IDs allow users to be in groups, permitting group access rights

File Sharing: Access Lists and Groups

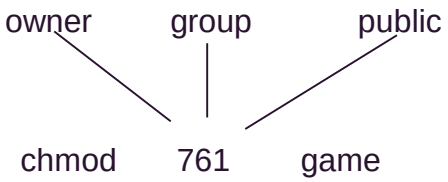
- Mode of access: **`read, write, execute`**
- Three classes of users (see **`chmod`** in Linux)

		rwx	
a) owner access	7	⇒	1 1 1
b) groups access	6	rwx ⇒	1 1 0
c) public access	1	rwx ⇒	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`*



File Sharing – Remote File Systems

U-read

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **www**

- **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

Preview of Next Chapter

FILE-SYSTEM IMPLEMENTATION

- Implementing local file systems and directory structures.
  - File System Structure and Implementation
  - Directory Structures and Implementation
- Implementation of remote file systems.
- Block allocation and free-block algorithms