

# Chapter 10: File System Interface

肖卿俊

办公室：九龙湖校区计算机楼212室

电邮：[csqjxiao@seu.edu.cn](mailto:csqjxiao@seu.edu.cn)

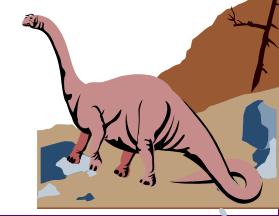
主页：<https://csqjxiao.github.io/PersonalPage>

电话：025-52091022



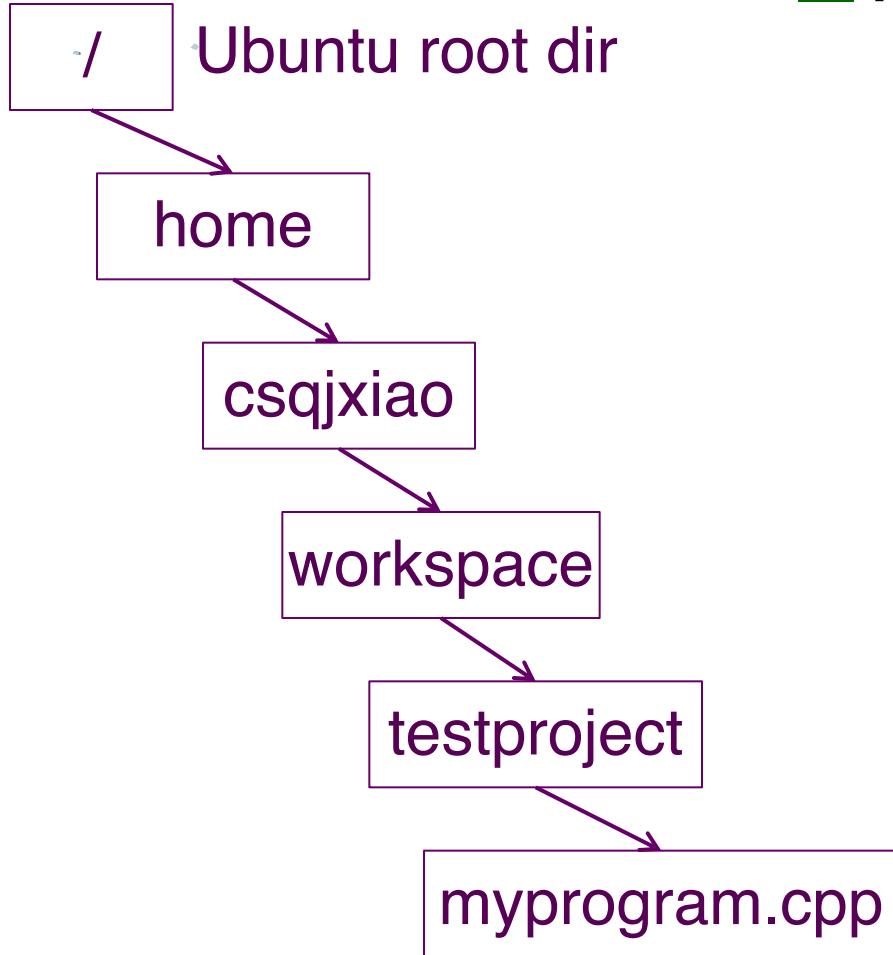
# Chapter 10: File-System Interface

- File Concept
- Access Methods
- Directory Structure
- File System Mounting
- File Sharing
- Protection



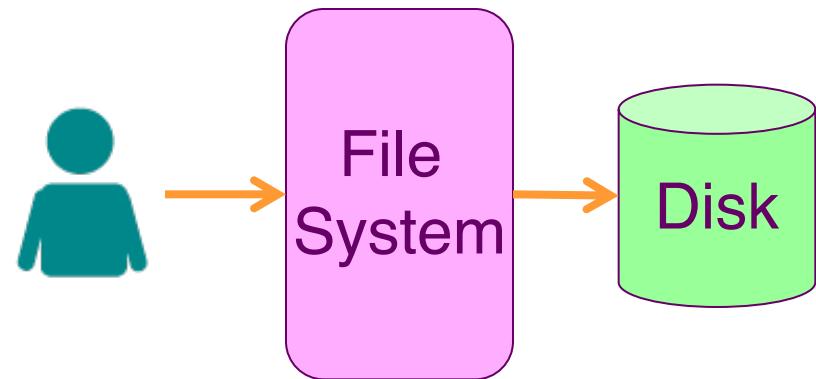


# File System Concept

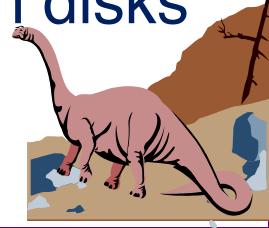


## ■ Key Abstraction

- ◆ File
- ◆ Filename
- ◆ Directory tree (folders)

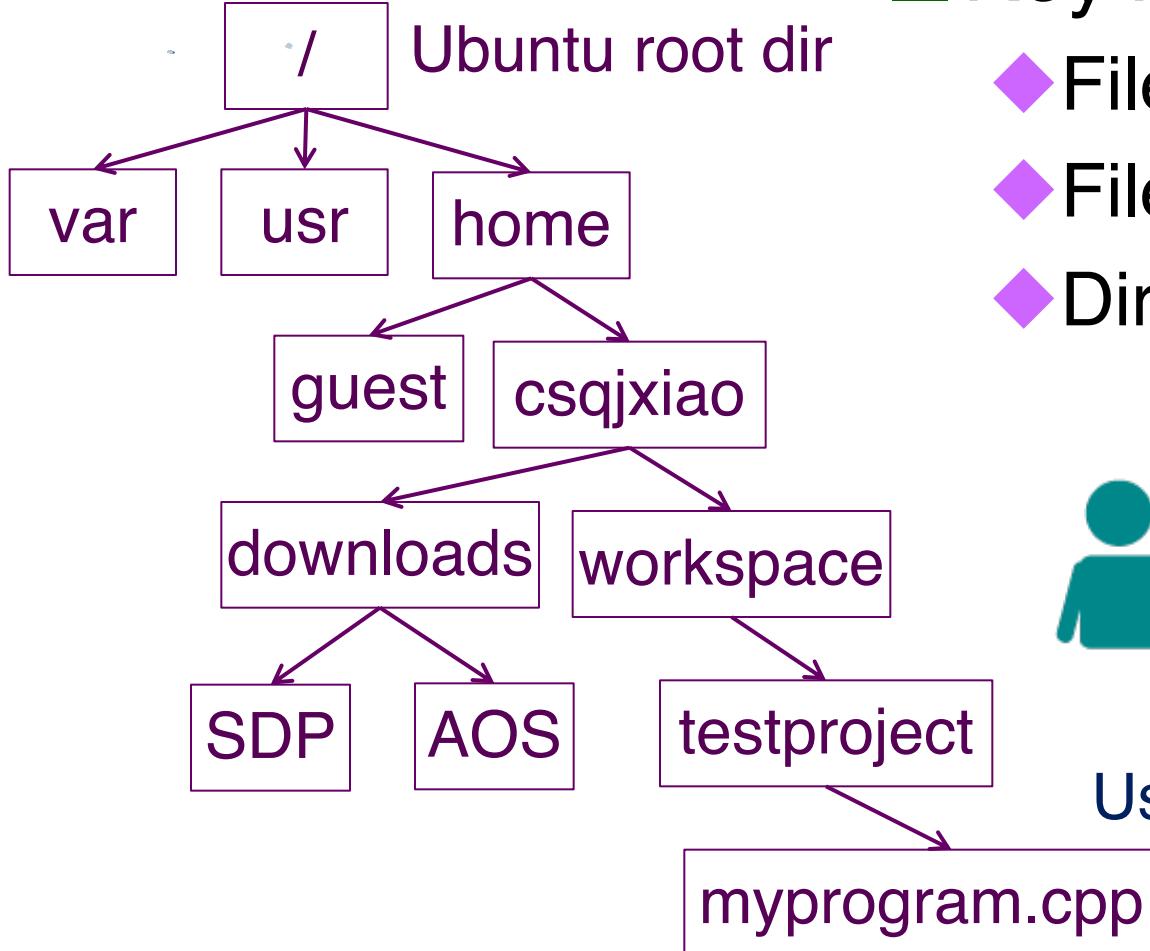


Users do not access directly  
the file blocks on disks



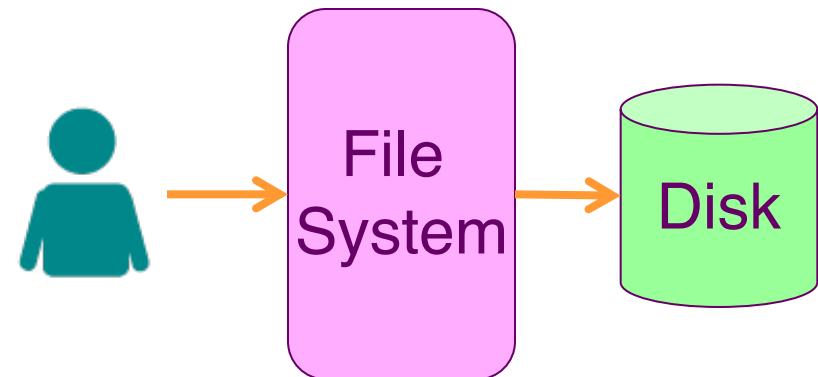


# File Path and Directory Tree

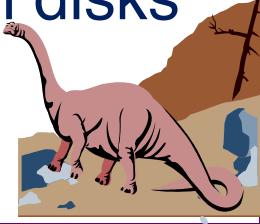


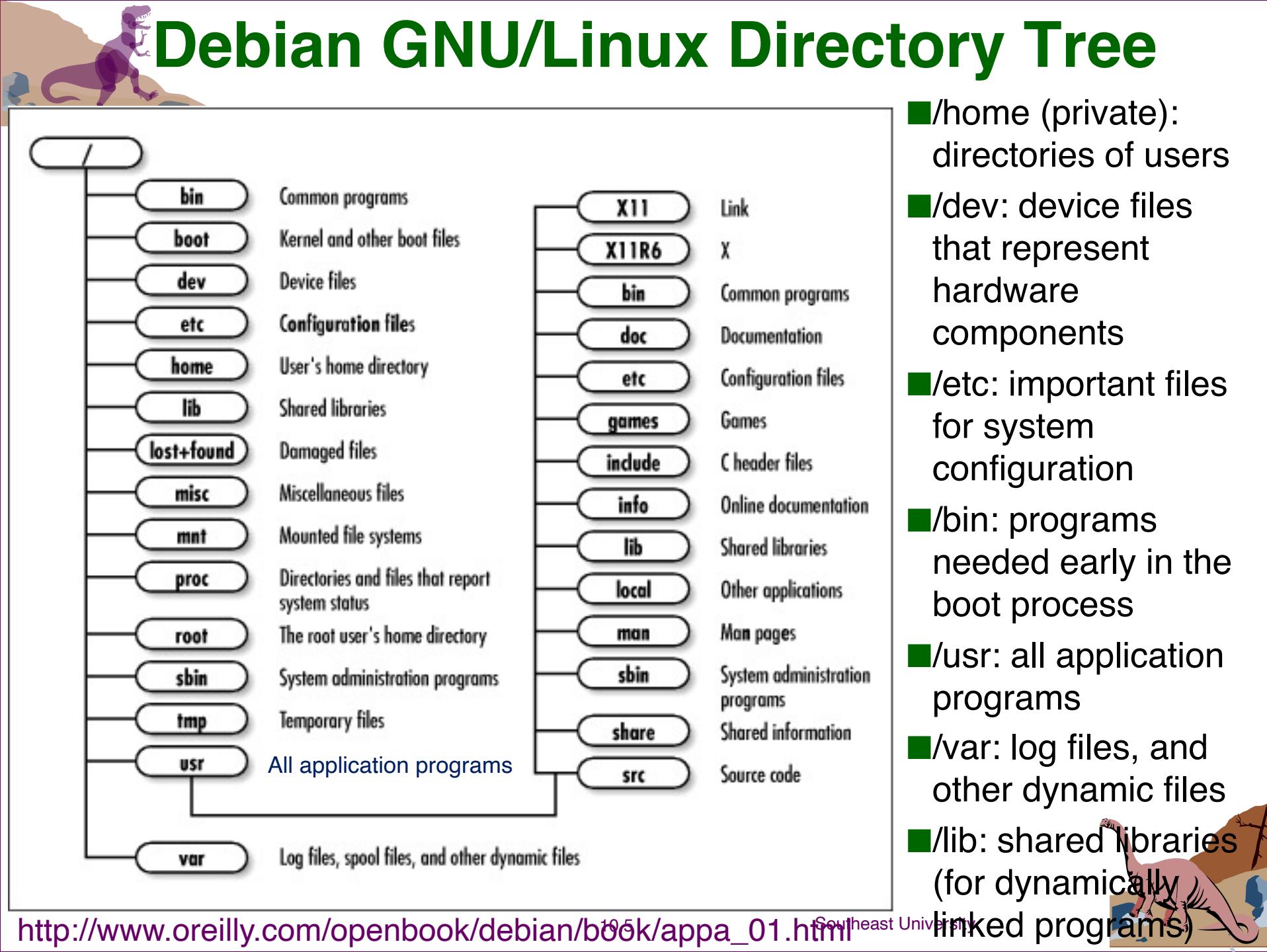
## ■ Key Abstraction

- ◆ File
- ◆ Filename
- ◆ Directory tree (folders)



Users do not access directly  
the file blocks on disks







# File Concept

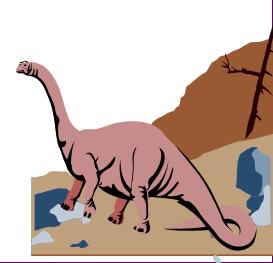
■ Contiguous logical address space

■ Types:

◆ Data

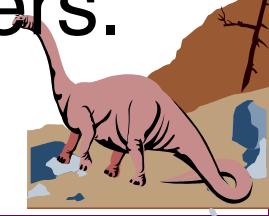
- ✓ numeric
- ✓ character
- ✓ binary

◆ 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 the first method by inserting appropriate control characters.
  - Who decides the file structure?
- 



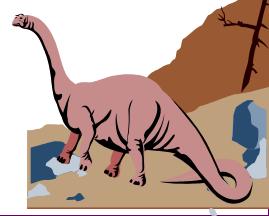
# File Attributes

- **Name** – only information kept in human-readable form.
- **Type** – needed for systems that support different types.
- **Location** – pointer to file location on device.
- **Size** – current file size.



# File Attributes (Cont.)

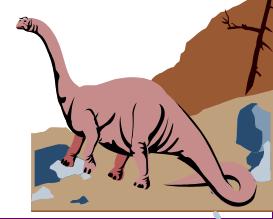
- **Protection** – controls who can do reading, writing, executing.
- **Time, date, and user identification** – data for protection, security, and usage monitoring.
- All these information about files are kept in the directory structure, which is maintained on the disk.





# File Types – Name, Extension

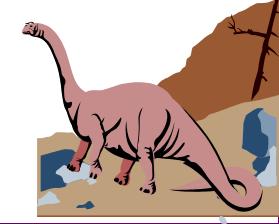
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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





# File Operations from Developer's Perspective

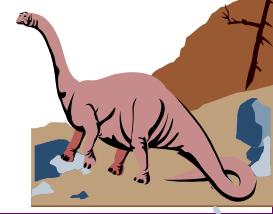
- Create
- Write
- Read
- Reposition within file – file seek
- Delete
- Truncate





# File Operations from Developer's Perspective (cont.)

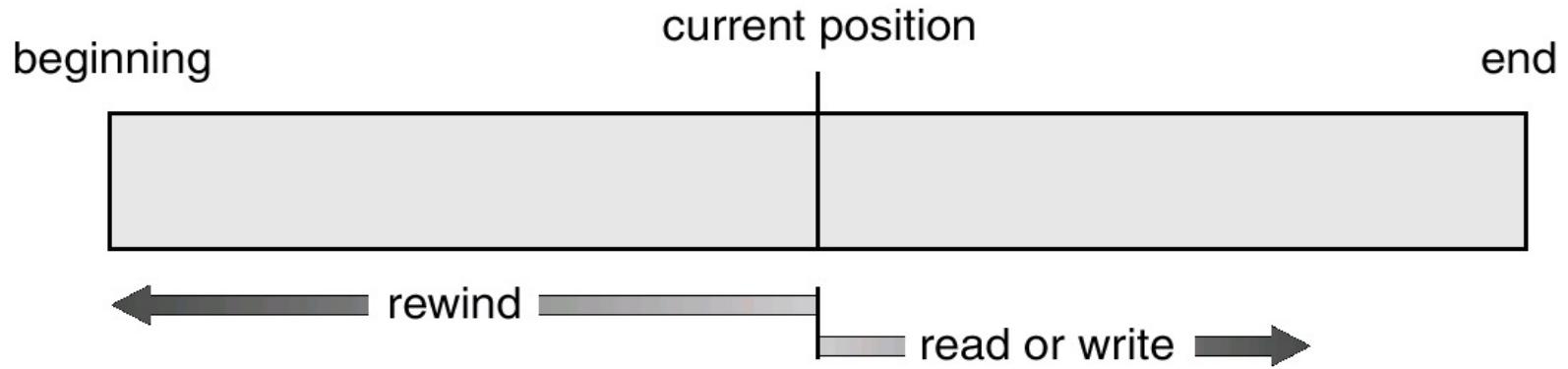
- $\text{open}(F_i)$  – search the directory structure on disk for entry  $F_i$ , and move the content of the entry from disk to memory.
- $\text{close}(F_i)$  – persist the content of entry  $F_i$  in memory to directory structure on disk.
- $\text{read}(F_i)$  – read the file content
- $\text{write}(F_i)$  – write to the file
- $\text{fseek}(F_i)$  – reposition the file cursor



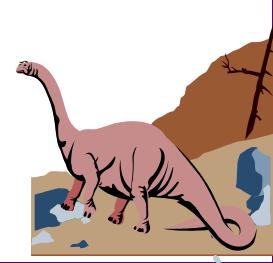


# File Content Access Methods

## ■ Sequential Access



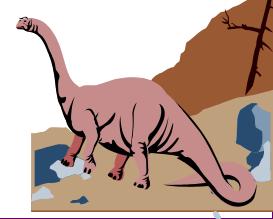
## ■ Direct Access





# Simulation of Sequential Access on a Direct-Access File

sequential access	implementation for direct access
<i>reset</i>	$cp = 0;$
<i>read next</i>	$read cp;$ $cp = cp+1;$
<i>write next</i>	$write cp;$ $cp = cp+1;$

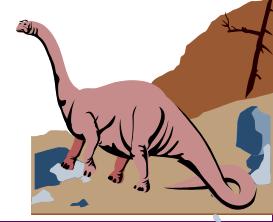




# Code Modifying a Key-Value Pair

KEY	VALUE
integer	integer

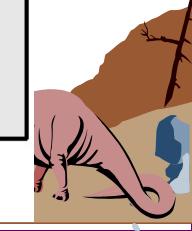
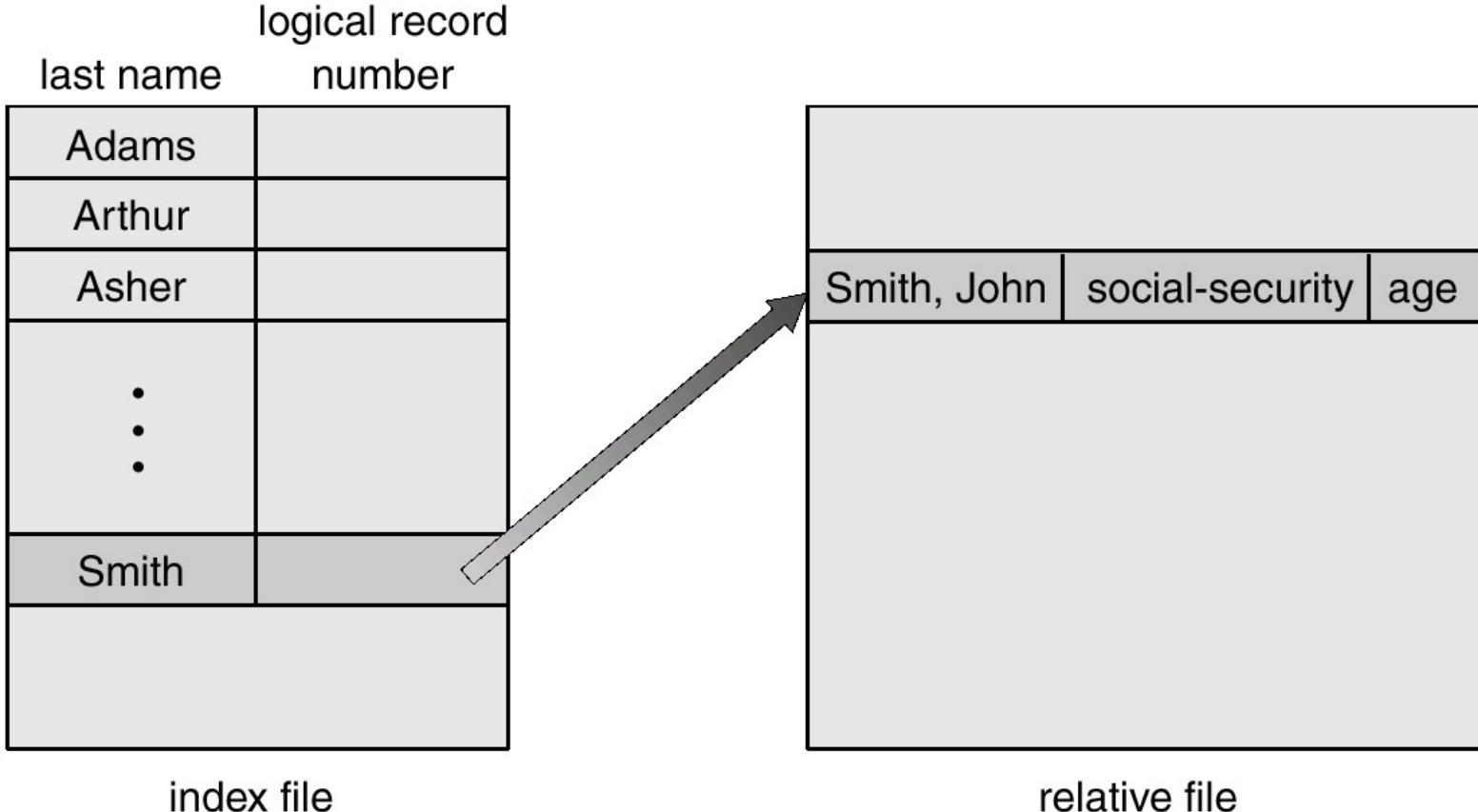
```
ssize_t len;
char * filename;
int key, srch_key, new_value;
filename = argv[1];
srch_key = strtol(argv[2], NULL, 10);
new_value = strtol(argv[3], NULL, 10);
int fd = open(filename, O_RDWR);
while(sizeof(int) == read(fd, &key, sizeof(int))) {
    if(key != srch_key)
        lseek(fd, sizeof(int), SEEK_CUR);
    else {
        write(fd, &new_value, sizeof(int));
        close(fd);
        return EXIT_SUCCESS;
    }
}
fprintf(stderr, "key not found!");
return EXIT_FAILURE;
```





# Example of Index and Relative Files

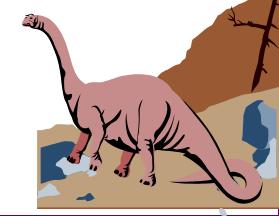
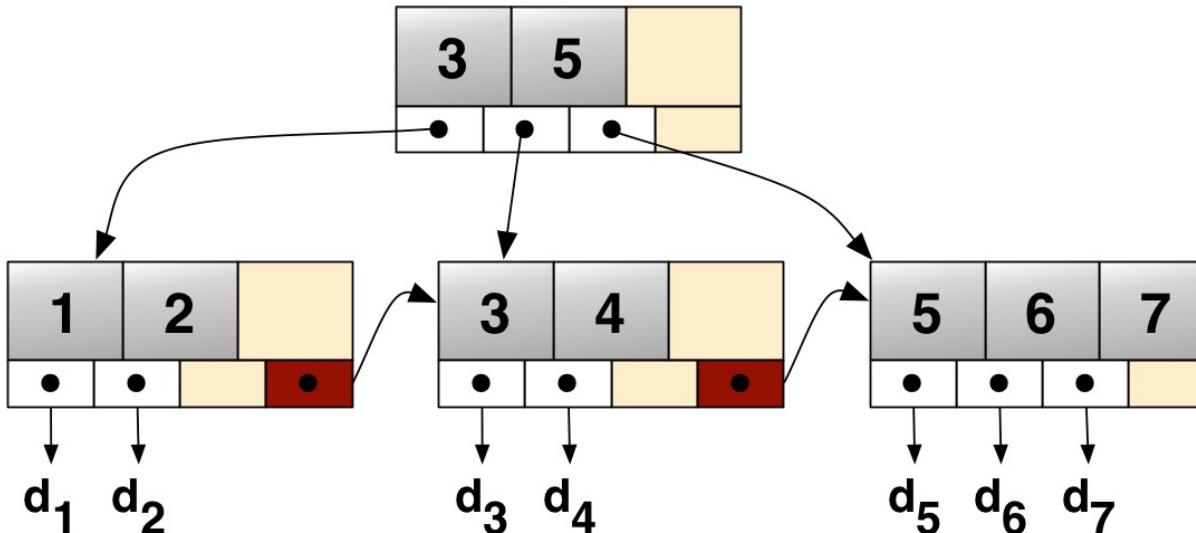
- Store keys in the index file
- Store values (or records) in the relative file
- How to quickly locate the record of John Smith





# B+ Tree for Index File

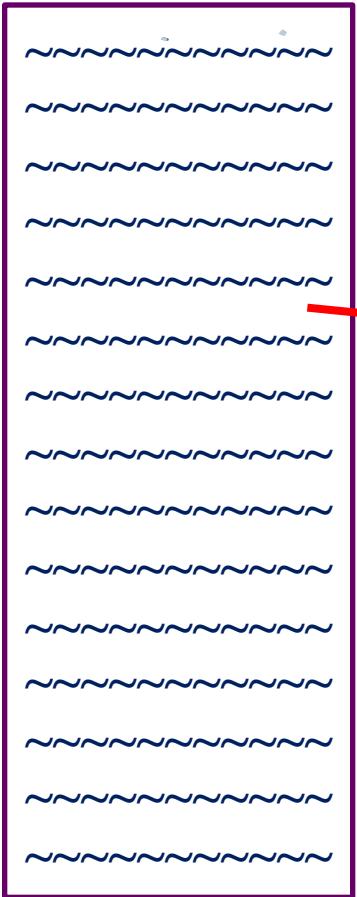
- The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context – in particular, filesystems. Unlike binary search trees, B+ trees have very high fanout (number of pointers to child nodes in a node, typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.





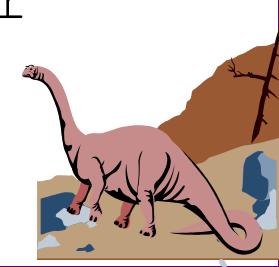
# File Content Direct Access by Memory Mapped File

file.txt



- **mmap()** creates a new mapping in the virtual address space of the calling process
- **munmap()** system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references

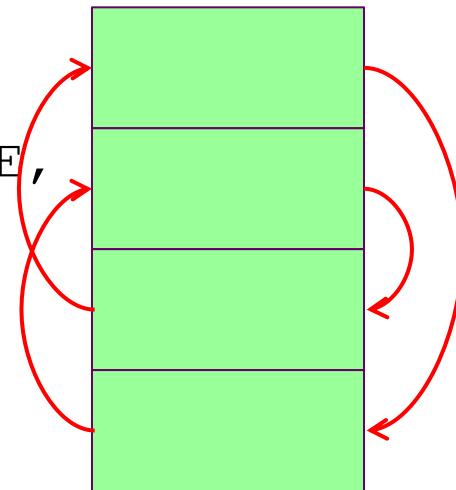
```
fd = open("file.txt", ...);  
buffer = mmap(..., fd, ...);  
  
// manipulate the buffer  
  
munmap(buffer, ...);  
close(fd);
```





# An Example of Memory Mapped File: Shuffle Blocks within a File

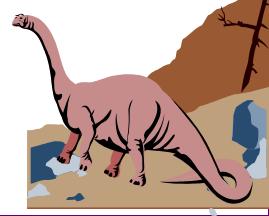
```
filename = argv[1];
card_size = strtol(argv[2], NULL, 10);
fd = open(filename, O_RDWR);
len = lseek(fd, 0, SEEK_END);
lseek(fd, 0, SEEK_SET);
buf = mmap(NULL, len, PROT_READ | PROT_WRITE,
           MAP_FILE | MAP_SHARED, fd, 0);
if( buf == (void*) -1) {
    fprintf(stderr, "mmap failed.\n");
    exit(EXIT_FAILURE);
}
memshuffle(buf, len, card_size);
munmap(buf, len);
close(fd);
return EXIT_SUCCESS;
```





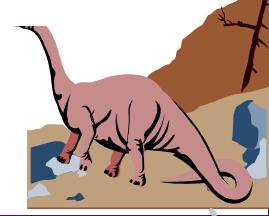
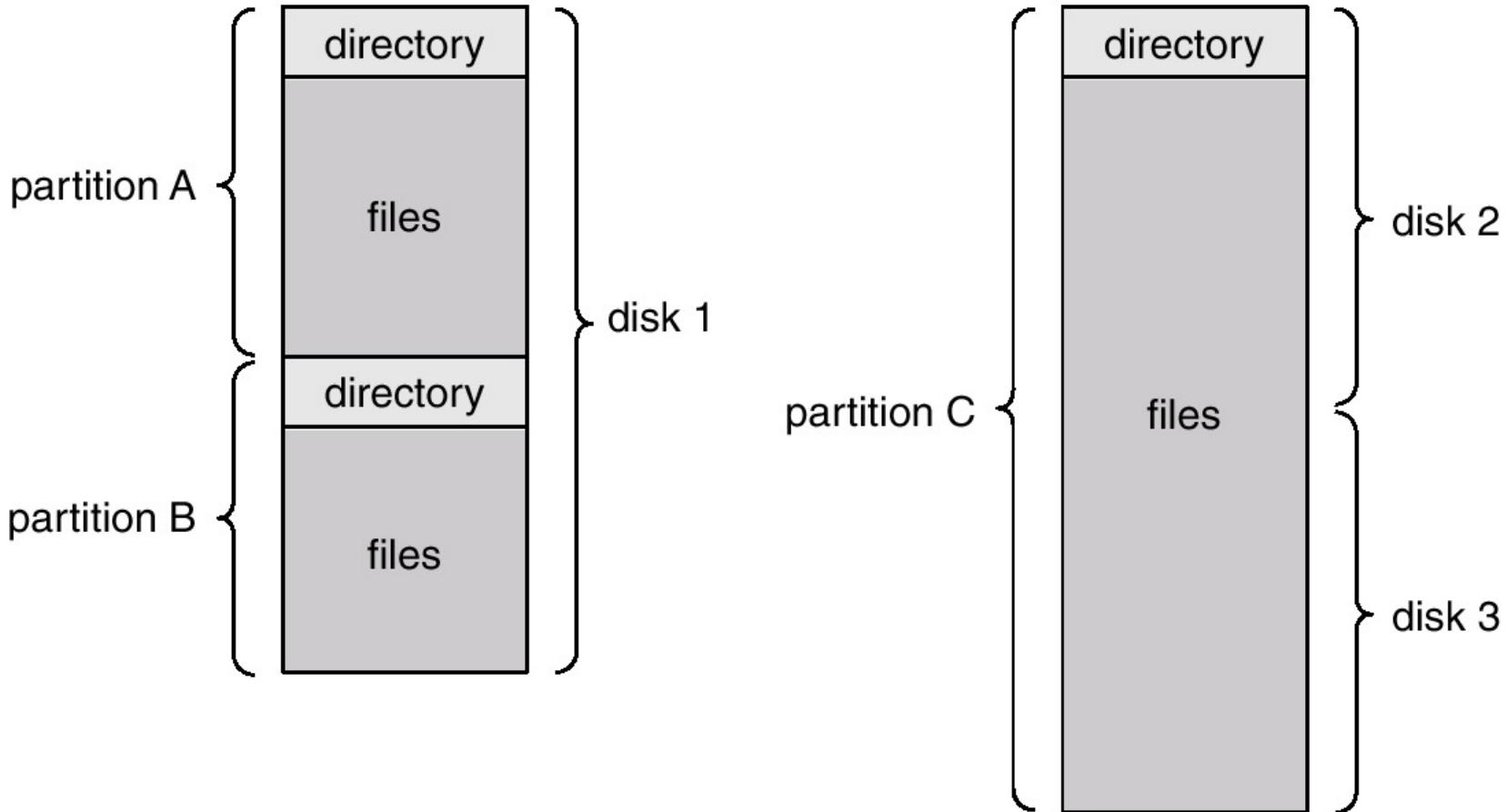
# Directory Structure

- disks are split into one or more partitions.
- each partition contains information about files within it
- The information is kept in entries in a device directory or volume table of contents





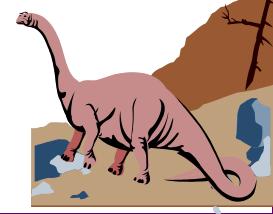
# A Typical File-system Organization





# Operations Performed on Directory

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





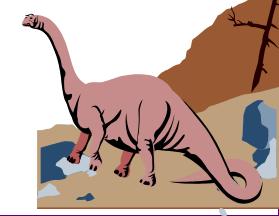
# Organize the Directory (Logically) to Obtain

■ **Efficiency** – locating a file quickly.

■ **Naming** – convenient to users.

- ◆ Two users can have the same name for different files.
- ◆ The same file can have several different names.

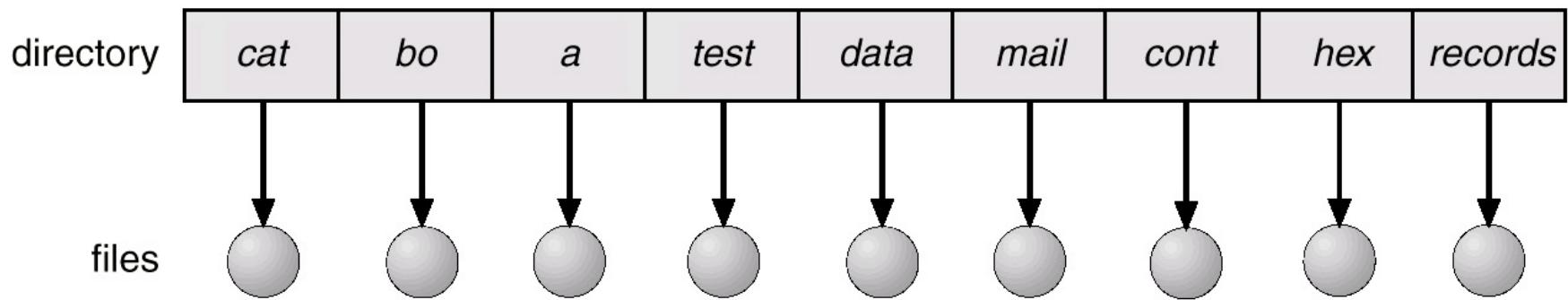
■ **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, ...)





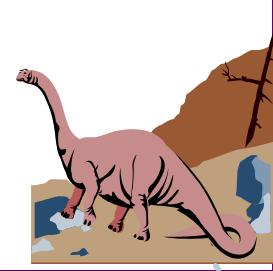
# Single-Level Directory

- A single directory for all users.



Naming problem

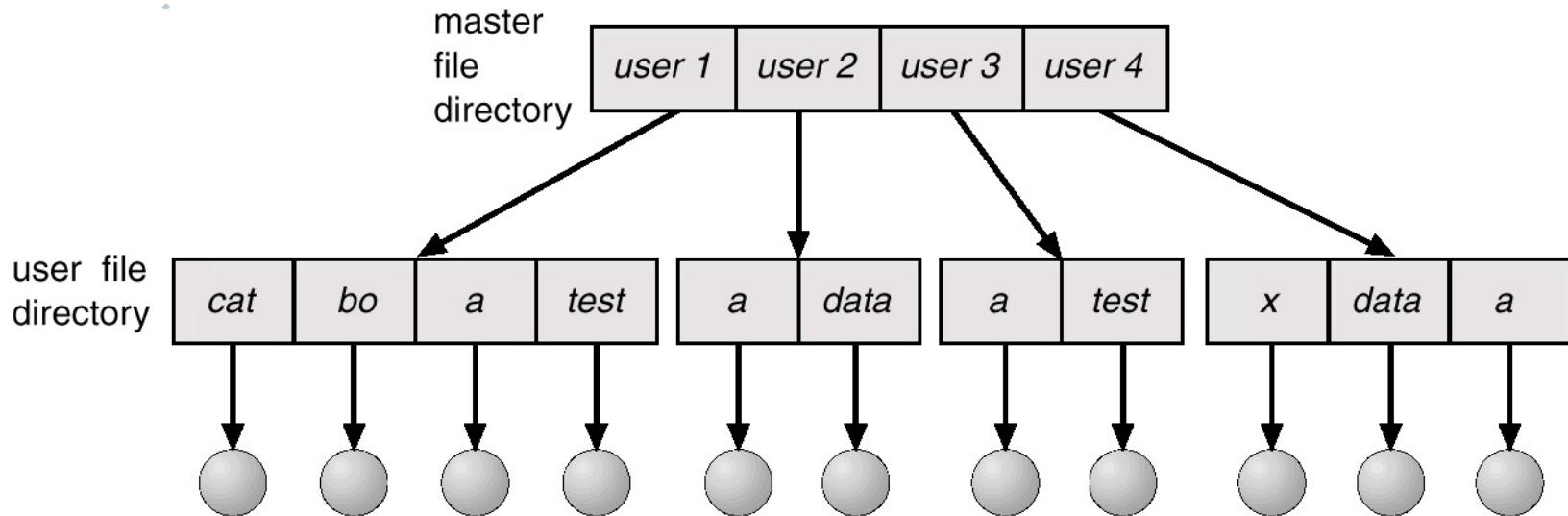
Grouping problem



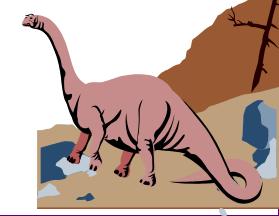


# Two-Level Directory

- Separate directory for each user.

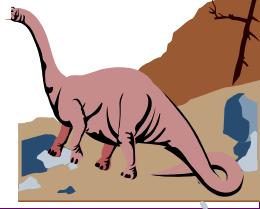
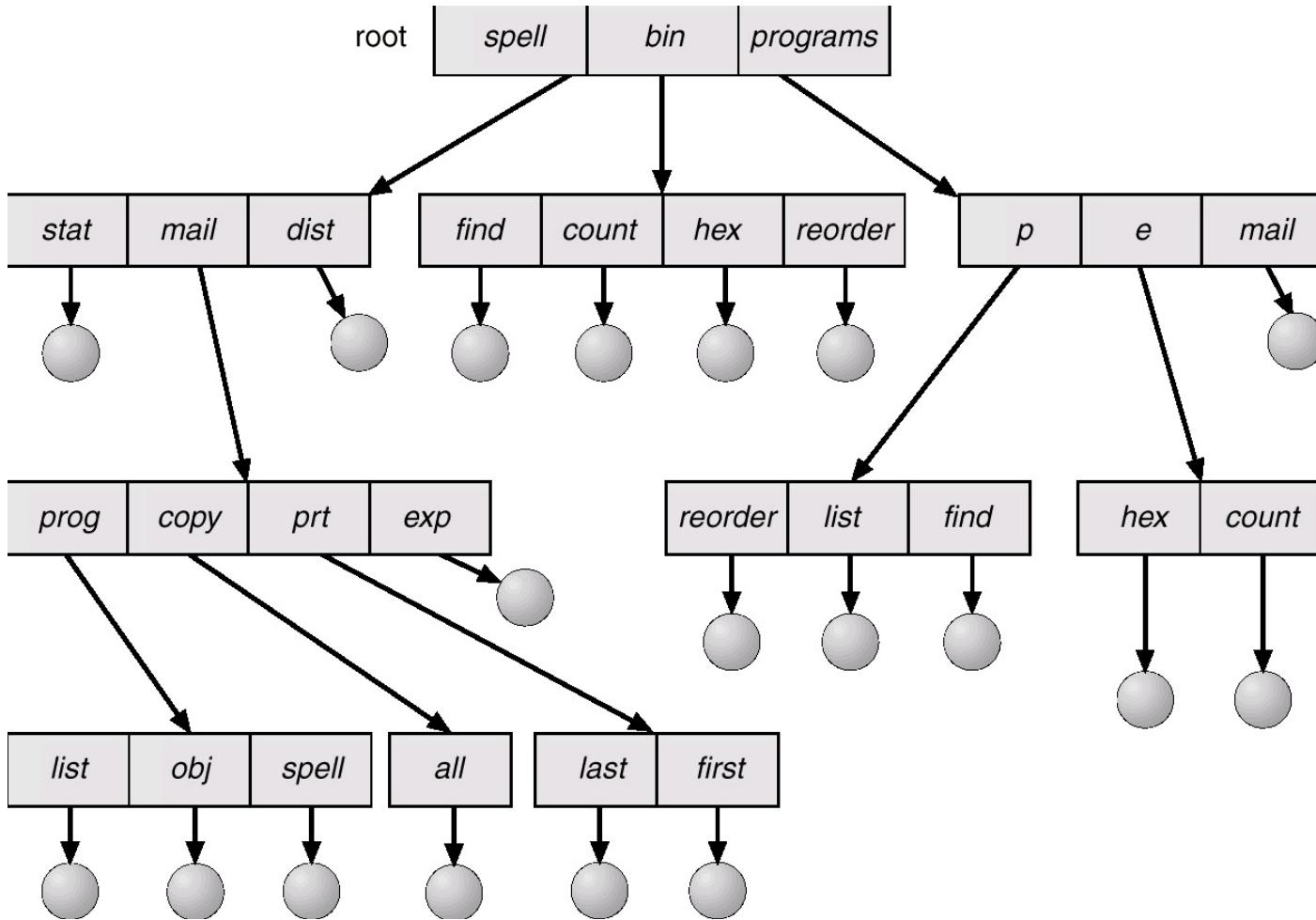


- Efficient searching
- Support path name, so can have the same file name for different users
- No grouping capability





# Tree-Structured Directories





# Tree-Structured Directories (cont.)

## ■ Efficient searching

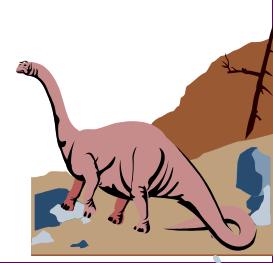
## ■ Convenient naming

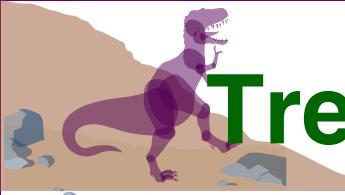
- ◆ Two users can have the same name for different files.

## ■ Grouping capability

## ■ Current directory (working directory)

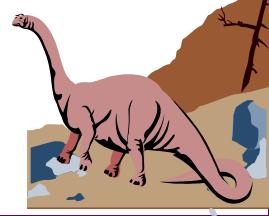
- ◆ `cd /spell/mail/prog`
- ◆ `type list`





# Tree-Structured Directories (cont.)

- Absolute or relative path name
- Creating a new file can be done in current directory.
- Delete a file
  - rm <file-name>**





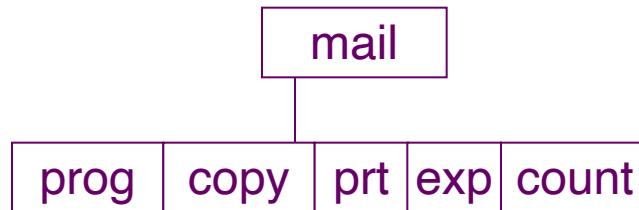
# Tree-Structured Directories (cont.)

- Creating a new subdirectory is done in current directory.

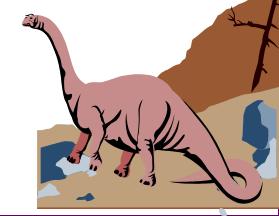
**mkdir <dir-name>**

Example: if in current directory **/mail**

**mkdir count**



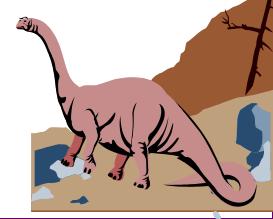
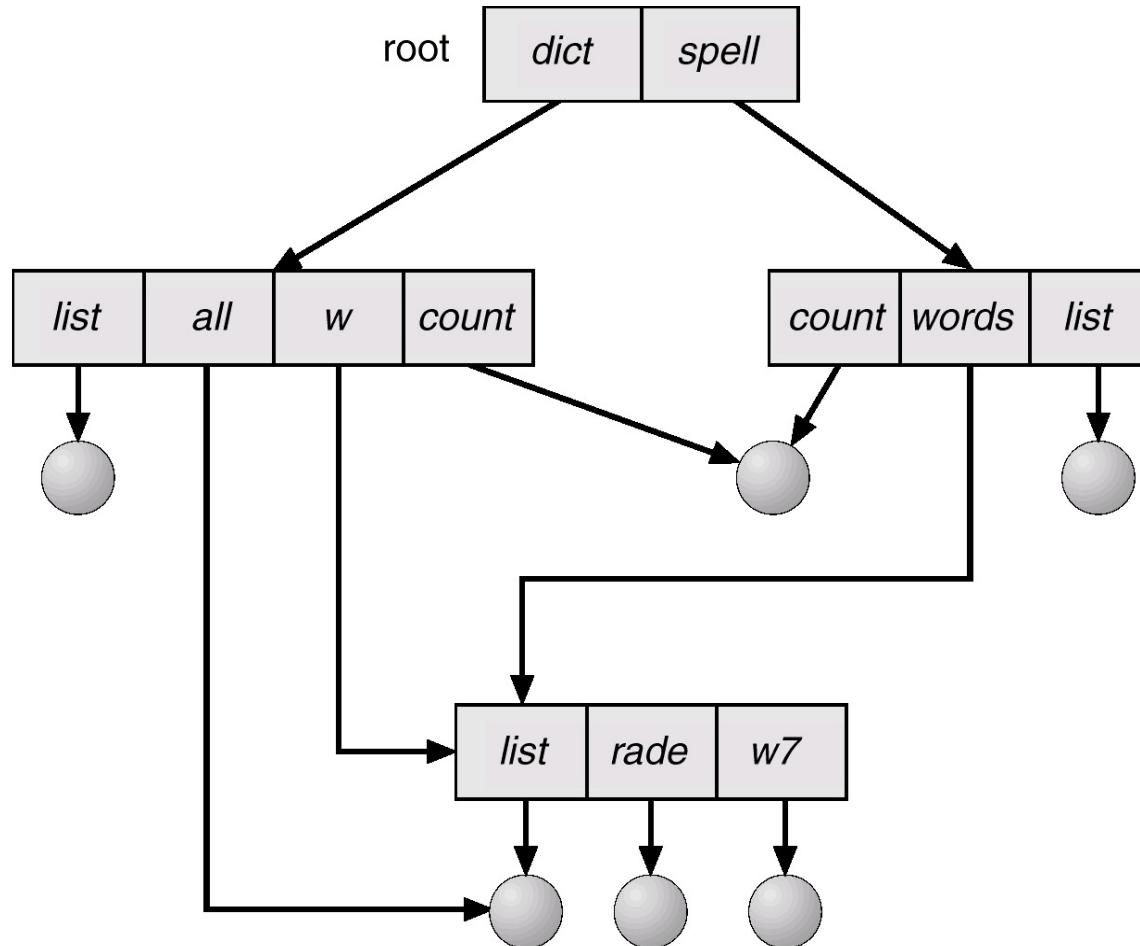
Deleting “mail”  $\Rightarrow$  deleting the entire subtree rooted by “mail”.





# Acyclic-Graph Directories

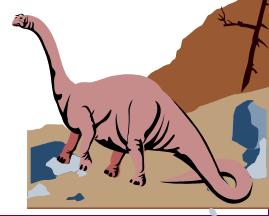
- Have shared subdirectories and files.  
The same file can have several different paths.





# Acyclic-Graph Directories (cont.)

- Two different names (aliasing)
- If *dict* deletes *count* ⇒ dangling pointer.  
Solutions:
  - ◆ Backpointers, so we can delete all pointers.
  - ◆ Entry-hold-count solution.
- These links we talked about are hard links in UNIX/Linus





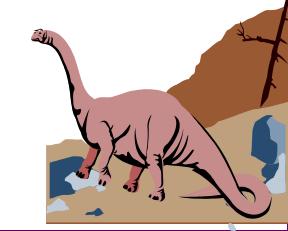
# In Linux/Unix Shortcuts are known as Link

## ■ Soft Links (symbolic links )

- ◆ You can make a link for either a file or a folder
- ◆ You can create link (shortcut) on different partition
- ◆ You got a different inode number from original.
- ◆ If real copy is deleted the link will not work.

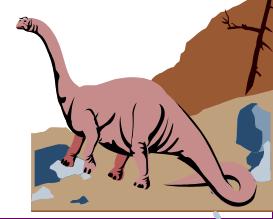
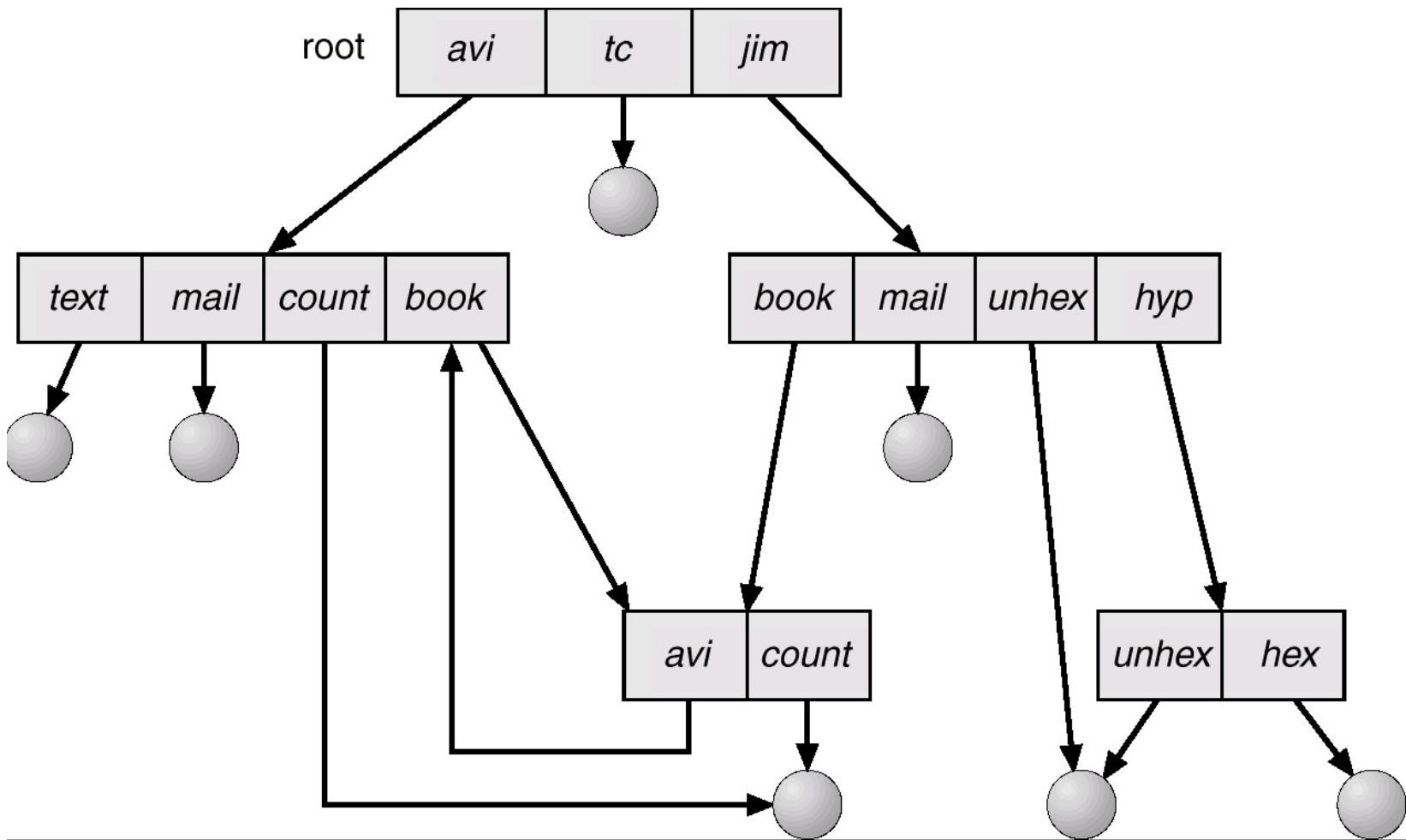
## ■ Hard Links

- ◆ For files only, and you cannot create a hard link on different partition (it should be on same partition)
- ◆ You got the same inode number as original
- ◆ If the real copy is deleted the link will work (because it act as original file )





# General Graph Directory

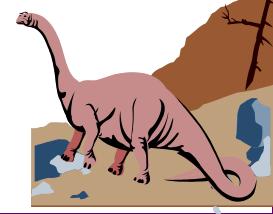




# General Graph Directory (cont.)

## ■ How do we guarantee no cycles?

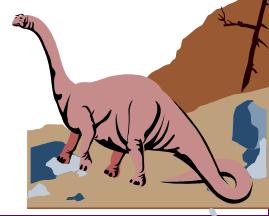
- ◆ Allow only links to file not subdirectories.
- ◆ Garbage collection.
- ◆ Every time a new link is added, use a cycle detection algorithm to determine whether it is OK.





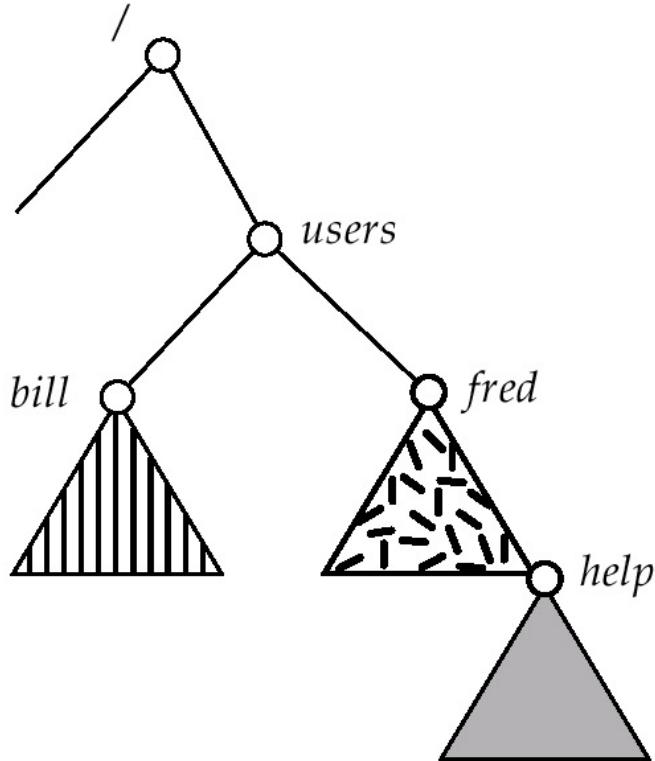
# File System Mounting

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

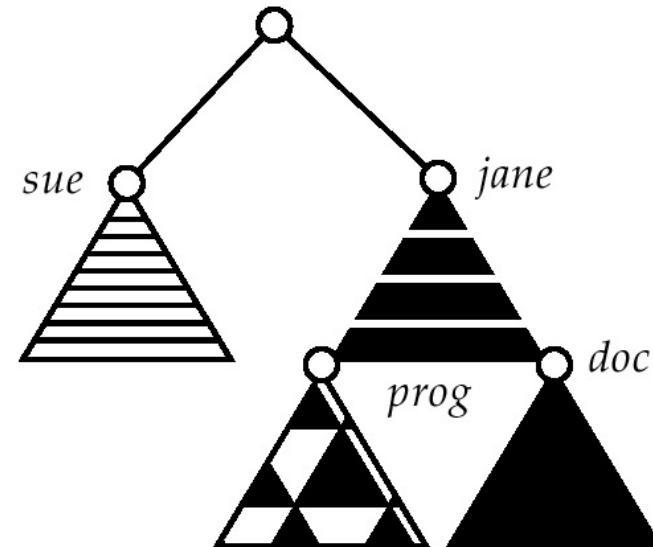




# (a) Existing (b) Unmounted Partition



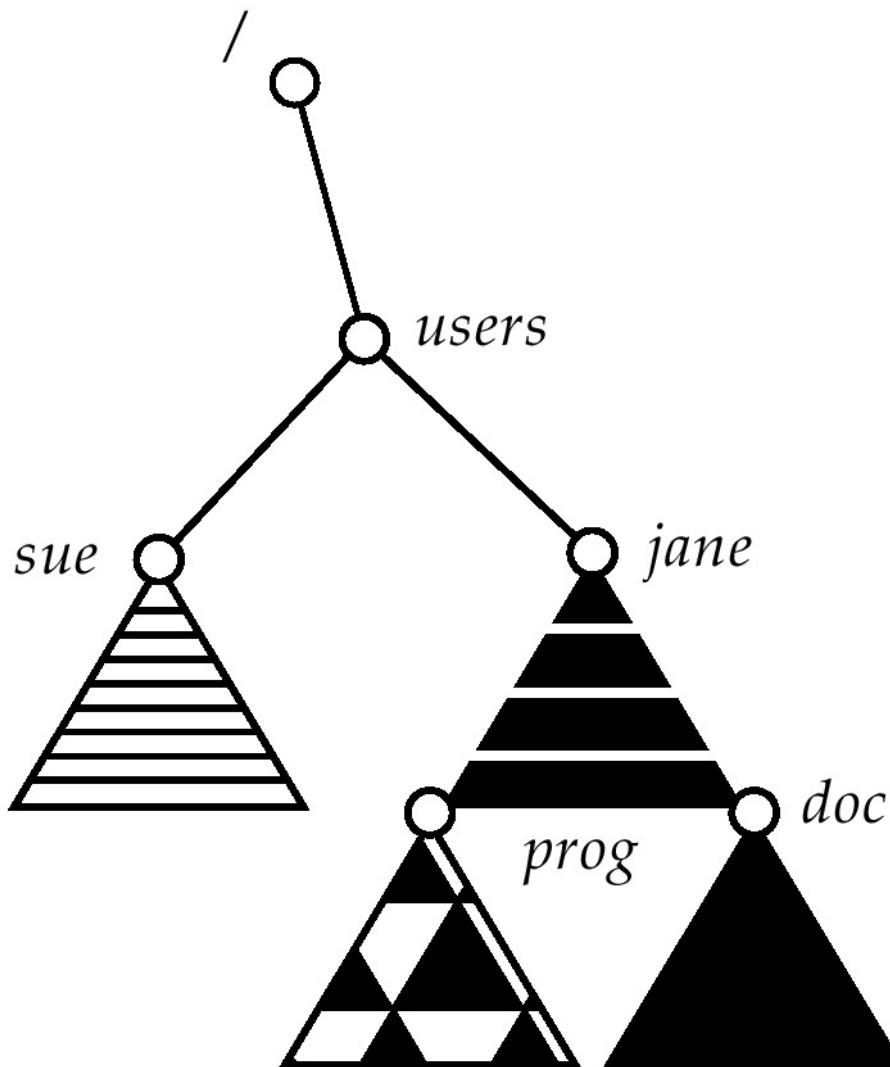
(a)



(b)



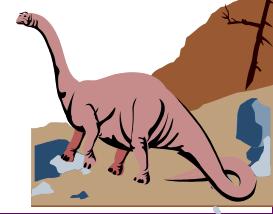
# Mount Point





# File Sharing

- 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.





# Protection

## ■ File owner/creator should be able to control:

- ◆ what can be done
- ◆ by whom

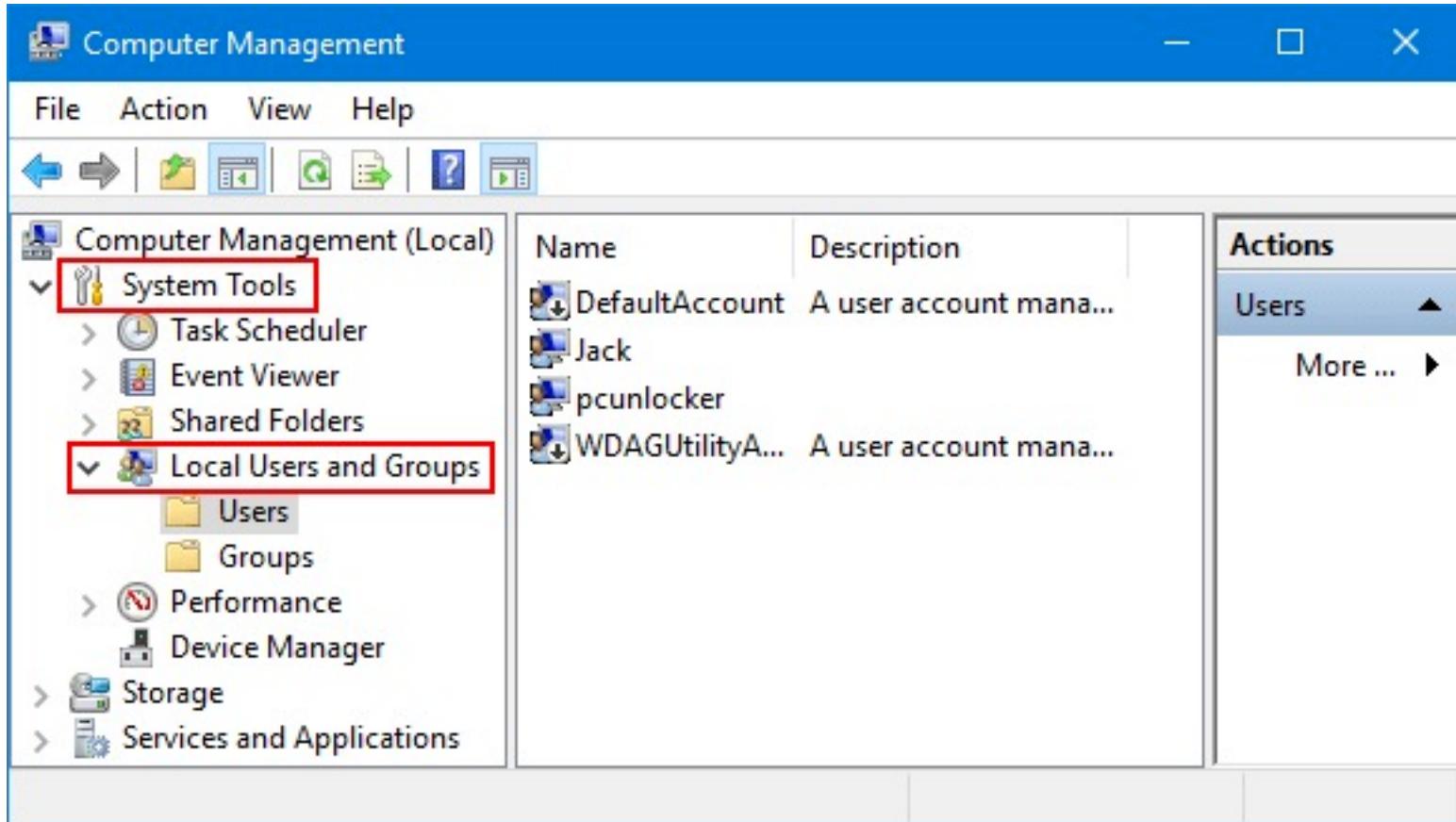
## ■ Types of access

- ◆ Read
- ◆ Write
- ◆ Execute
- ◆ Append
- ◆ Delete
- ◆ List



# User and Group on Windows

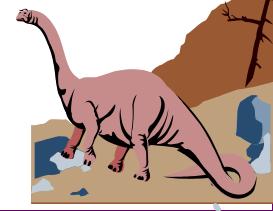
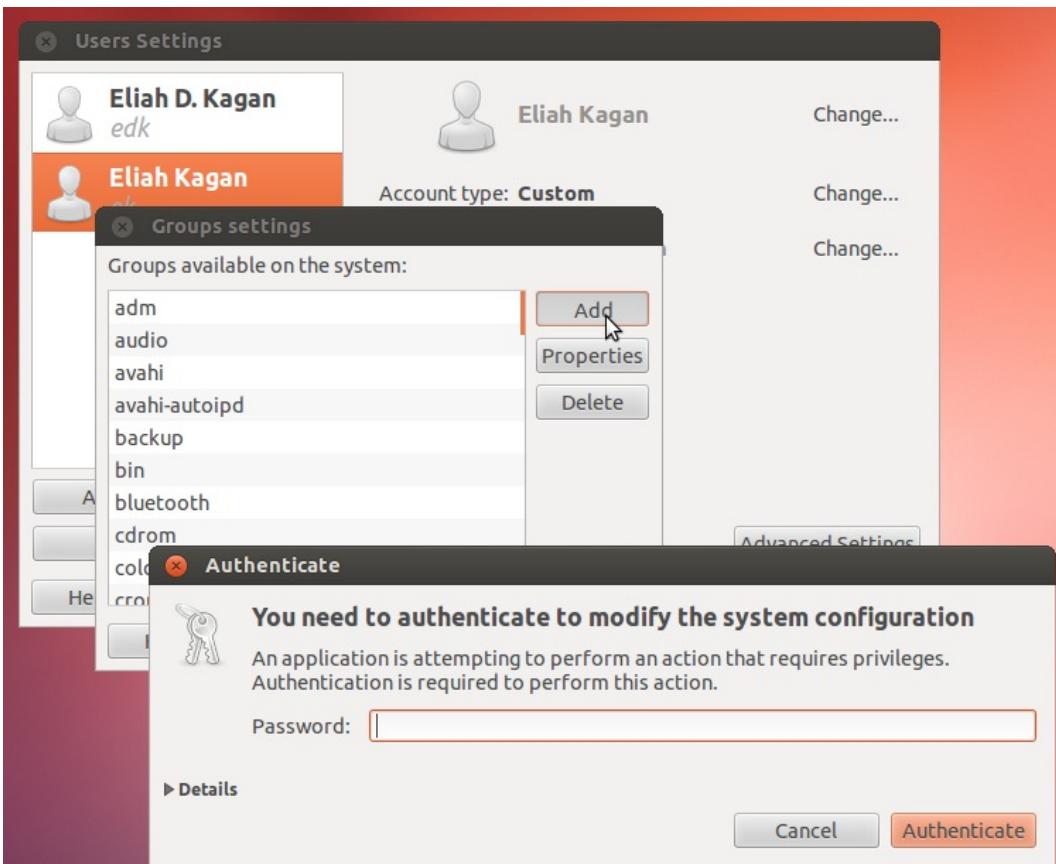
- Each user has a user ID (uid).
- A user belongs to one or multiple groups





# User and Group on Linux

- GUI is also available to manage users and groups on Ubuntu





# User File on Linux System

- The user file is located in /etc/passwd
- There is a special privileged root user (uid=0)

```
seed@VM:$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash ← root user is here
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
seed:x:1000:1000:seed,,,,:/home/seed:/bin/bash ← "seed" user for
vboxadd:x:999:1::/var/run/vboxadd:/bin/false our security experiment
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,,:/nonexistent:/bin/false
bob:x:1001:1001,,,,:/home/bob:/bin/bash ← the first command
                                                after user loginv
```

Passwd is stored elsewhere

↑ uid.   ↑ gid\_

↑ Home Dir

↑ shell

These are not real users



# Group File on Linux System

- The group file is located in /etc/group

```
Terminal
seed@VM:$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,seed
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
```

- The Linux command to add a user to a group

```
sudo usermod -a -G groupname username
```

- The command to show the IDs of current user

```
Terminal
seed@VM:$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```



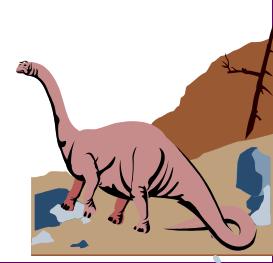
# Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users RWX
  - a) **owner access** 7  $\Rightarrow$  1 1 1
  - b) **group access** 6  $\Rightarrow$  1 1 0
  - c) **public access** 1  $\Rightarrow$  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

owner    group    public  
 $\swarrow$                $\downarrow$                $\searrow$   
`chmod 761 game`

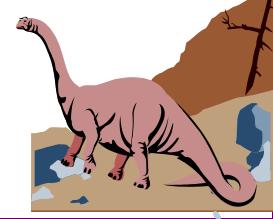
`chgrp G game`





# 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/





# User File and Shadow File on Linux

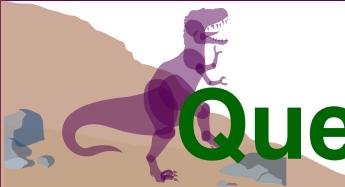
- The shadow file stores the encrypted passwords of all users. It can be modified only by root. Normal users cannot read it.

```
Terminal
seed@VM$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root 2521 Nov 1 22:37 /etc/passwd
-rw-r----- 1 root shadow 1496 Nov 1 22:37 /etc/shadow
```

```
Terminal
seed@VM$ head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
seed@VM$
seed@VM$ sudo head -5 /etc/shadow
root:$6$NrF4601p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEE0
5wj8aU3GRHW2BaodUn4K3vgyEjwPspr/kqzAqtcu.:17400:0:99999:7:::
daemon:*:17212:0:99999:7:::
bin:*:17212:0:99999:7:::
sys:*:17212:0:99999:7:::
sync:*:17212:0:99999:7:::
```

Need root user to read the shadow file

The encrypted password



# Question about File Access-Control

■ Which of the following will generate a permission error?

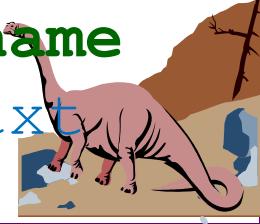
- cat foo.txt
- cat dir/bar.txt
- touch dir/new.txt

```
$ ls -l ./
```

<b>Permission</b>	<b>user</b>	<b>group</b>	....	<b>Filename</b>
drw-r--r--	me	me		dir
-rw-r--r--	other	other		foo.txt

```
$ sudo ls -l dir
```

<b>Permission</b>	<b>user</b>	<b>group</b>	....	<b>Filename</b>
-rw-r--r--	me	me		bar.txt





# Another Question

■ Which of the following will generate a permission error?

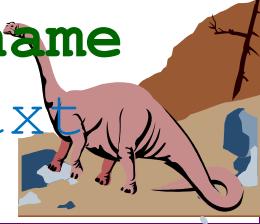
- cat foo.txt
- cat dir/bar.txt
- touch dir/new.txt

```
$ ls -l ./
```

<b>Permission</b>	<b>user</b>	<b>group</b>	....	<b>Filename</b>
d--xr--r--	me	me		dir
-rw-r--r--	other	other		foo.txt

```
$ sudo ls -l dir
```

<b>Permission</b>	<b>user</b>	<b>group</b>	....	<b>Filename</b>
-rw-r--r--	me	me		bar.txt



```
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ mkdir dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
drwxr-xr-x  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 644 dir
Password:
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
drw-r--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
touch: dir/new.txt: Permission denied
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 144 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
d--xr--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
touch: dir/new.txt: Permission denied
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 344 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
d-wxr--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$
```

MacOS的执行结果：目录dir在创建的开始，rwx权力都属于owner。后面不管是644、144权力，都会touch报错Permission denied。改成344，有x和w权力，就没问题了。`cd dir`进入dir目录的操作，一定需要dir目录的执行权。