



File System Interface

Yajin Zhou (<http://yajin.org>)

Zhejiang University



Review

- Page replacement: FIFO, Optimal, LRU (stack, counter), Approximation (reference bit, second chance), Counting (LFU, MFU)
- Global vs local allocation
- Reclaiming pages, OOM
- MAJOR/Minor page fault, NUMA
- Thrashing, Working-Set Model, How to track working set
- Kernel memory allocation: buddy, slab
- Other consideration: Prepaging, pagesize, TLB reach ...



File Concept

- **File** is a contiguous logical address space for storing information
 - database, audio, video, web pages...
- There are different types of file:
 - data: numeric, character, binary
 - program
 - special one: proc file system - use file-system interface to retrieve system information



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
- Many variations, including **extended file attributes** such as file checksum



File info Window on Mac OS X





File Operations

- OS provides file operations to
 - create:
 - space in the file system should be found
 - an entry must be allocated in the directory
 - open: most operations need to file to be opened first
 - return a handler for other operations
 - read/write: need to maintain a **pointer**



File Operations

- reposition within file - seek
- delete
 - Release file space
 - Hardlink: maintain a counter - delete the file until the last link is deleted
- truncate: delete a file but maintains its attributes
- Other operations can be implemented using these ones
 - Copying: create and read/write



Open Files

- Most of file operations need to search the directory to find the named file
 - To avoid the searching, OS maintains a table - **open-file table** contains information about all open files
 - Then following operation is specified via an index to the table - no searching is required
- For os that several processes may open the file simultaneously
 - **Per-process table:** current location pointer, access rights
 - **System-wide table:** location on the disk ...



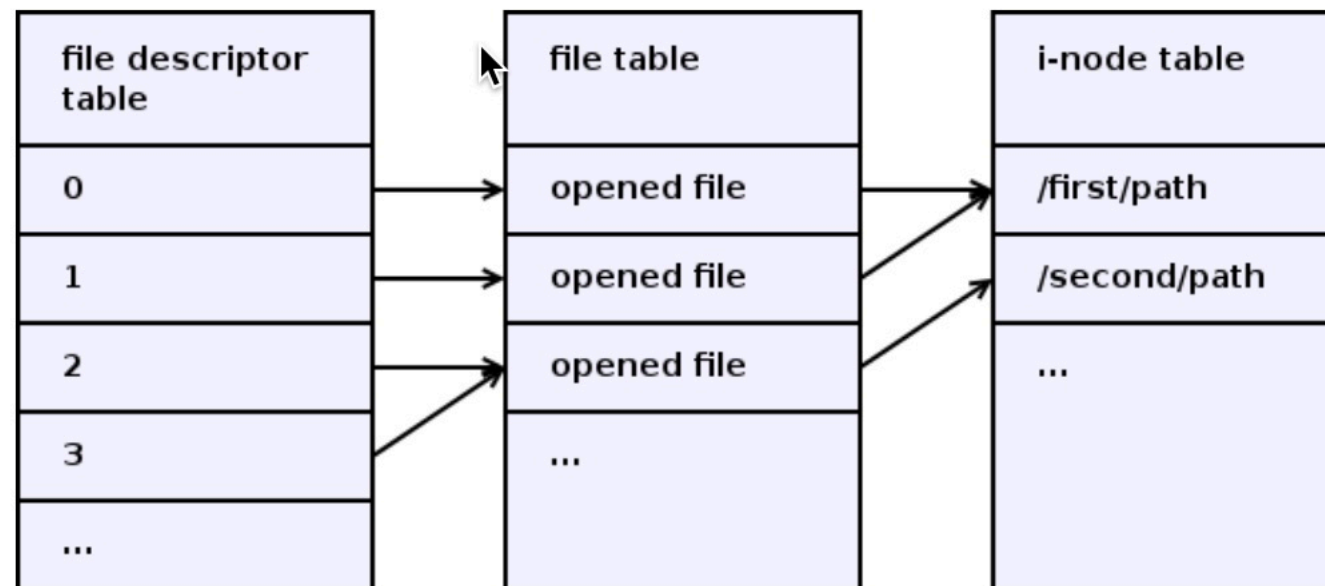
Information with Open Files

- Several pieces of information are associated with an open file
 - file position: pointer to last read/write location
 - file position is **per-process** that has the file open - since file could be opened by multiple processes
 - **file-open count**: the number of a file is open
 - to allow removal of data from open-file table when last processes closes it
 - disk location: cache of data access information
 - access rights: per-process access mode information



Open Files

- Some file systems provide file lock to mediate access to a file
- Two types of lock
 - **Shared lock** - multiple processes can acquire the lock concurrently
 - **Exclusive lock** - one process can acquire such a lock
- Two locking mechanisms
 - **mandatory lock**: access is denied depending on locks held and requested
 - **advisory lock**: processes can find status of locks and decide what to do



A file descriptor is an index in the per-process file descriptor table (in the left of the picture). Each file descriptor table entry contains a reference to a file object, stored in the file table (in the middle of the picture). Each file object contains a reference to an i-node, stored in the i-node table (in the right of the picture).

A file descriptor is just a number that is used to refer a file object from the user space. A file object represents an opened file. It contains things like current read/write offset, non-blocking flag and another non-persistent state. An i-node represents a filesystem object. It contains things like file meta-information (e.g. owner and permissions) and references to data blocks.

File descriptors created by several `open()` calls for the same file path point to different file objects, but these file objects point to the same i-node. Duplicated file descriptors created by `dup2()` or `fork()` point to the same file object.

A BSD lock and an Open file description lock is associated with a file object, while a POSIX record lock is associated with an `[i-node, pid]` pair. We'll discuss it below.



BSD locks (flock)

- Features:
 - not specified in POSIX, but widely available on various Unix systems
 - always lock the entire file
 - associated with a **file object**
 - **duplicated file descriptors**, e.g. created using dup2 or fork, share the lock acquisition
 - **independent file descriptors**, e.g. created using two open calls (even for the same file), don't share the lock acquisition
 - This means that with BSD locks, threads or processes **can't be synchronized on the same or duplicated file descriptor**, but nevertheless, both can be synchronized on **independent file descriptors**.
 - Work in both locking modes (exclusive and shared)
- up to Linux 2.6.11, didn't work on NFS; since Linux 2.6.12, flock() locks on NFS are emulated using fcntl() POSIX record byte-range locks on the entire file (unless the emulation is disabled in the NFS mount options)



Name

flock - apply or remove an advisory lock on an open file

Synopsis

#include <sys/file.h>

int flock(int *fd*, int *operation*);

Description

Apply or remove an advisory lock on the open file specified by *fd*. The argument *operation* is one of the following:

LOCK_SH

Place a shared lock. More than one process may hold a shared lock for a given file at a given time.

LOCK_EX

Place an exclusive lock. Only one process may hold an exclusive lock for a given file at a given time.

LOCK_UN

Remove an existing lock held by this process.



File Types

- File types: as part of the file names - file extension
- File type: magic number of the file - elf



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 com- pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

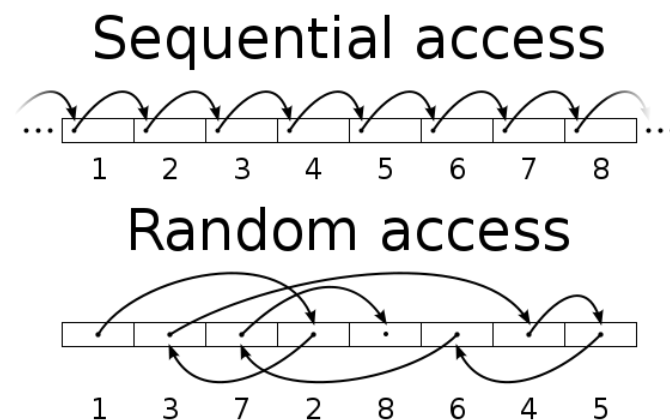


File Structure

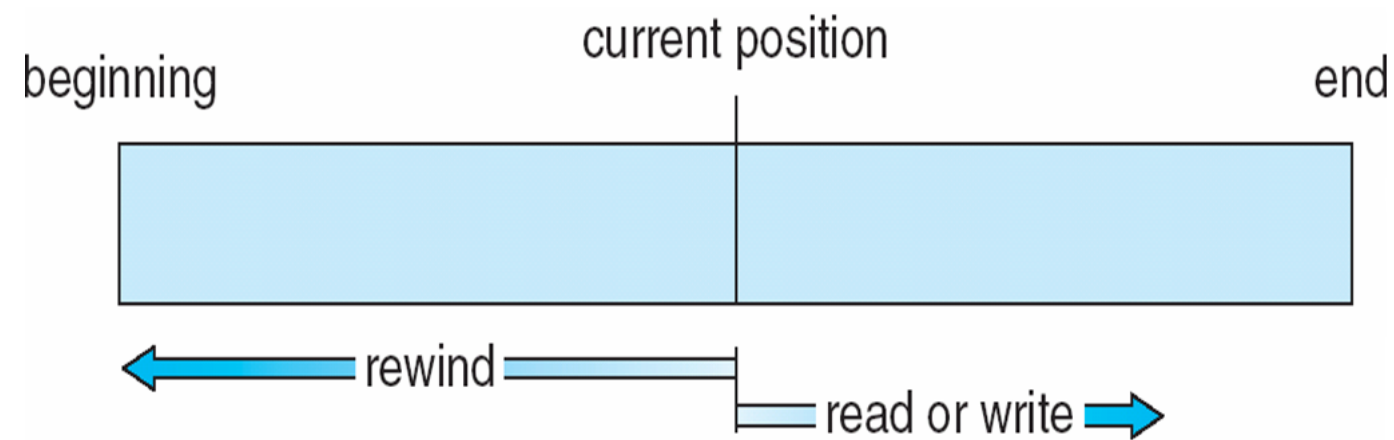
- A file can have different structures, determined by OS or program
 - **no structure**: a stream of bytes or words
 - linux files
 - **simple record structure**
 - lines of records, fixed length or variable length
 - e.g., database
 - **complex structures**
 - e.g., word document, relocatable program file
 - simple and complex structure can be encoded in the first method
- Usually user programs are responsible for identifying file structure

Access Methods

- Sequential access
 - a group of elements is access **in a predetermined order**
 - for some media types, the only access mode (e.g., **tape**)
- Direct access
 - access an element at an **arbitrary position** in a sequence in (roughly) **equal time**, independent of sequence size
 - it is possible to emulate random access in a tape, but access time varies
 - sometime called random access



Sequential-access File





Sequential Access on Direct-access File

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

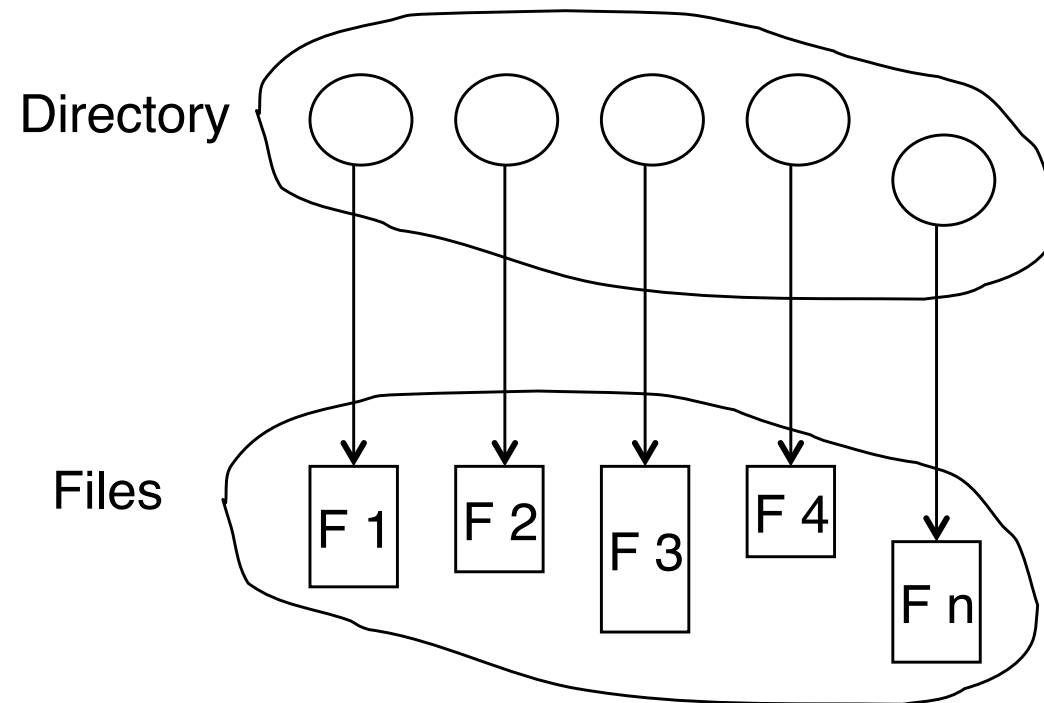


Other methods

- Based on direct-access method
- An index for the file points to blocks
 - Find a record in the file, first search the index and then use the pointer to access the block
 - We may use multiple layers of index

Directory Structure

- Directory is a collection of nodes containing information about all files



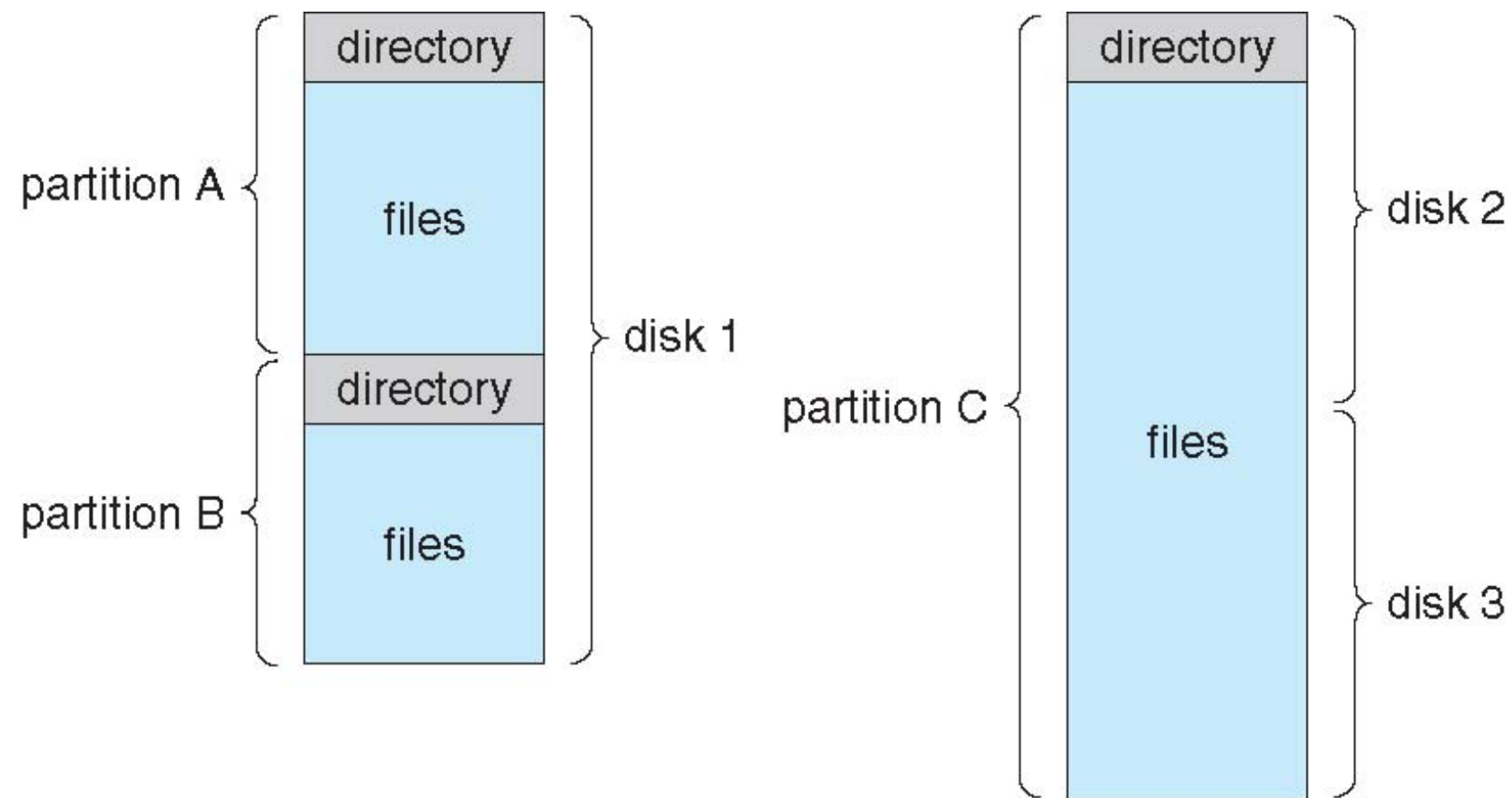
both the directory structure and the files reside on disk



Disk Structure

- Disk can be subdivided into **partitions**
 - partitions also known as **minidisks, slices**
 - different partitions can have different file systems
 - a partition containing file system is known as a **volume**
 - each volume tracks file system info in the volume's table of contents
 - a file system can be general purpose or special purpose
- disk or partition can be used **raw** (without a file system)
 - applications such as database prefer raw disks

A Typical File-system Organization





Operations Performed on Directory

- Create a file: new files need to be created and added to directory
- delete a file: remove a file from directory
- List a directory: list all files in directory
- Search for a file: pattern matching
- Traverse the file system: access every directory and file with a directory
- ...

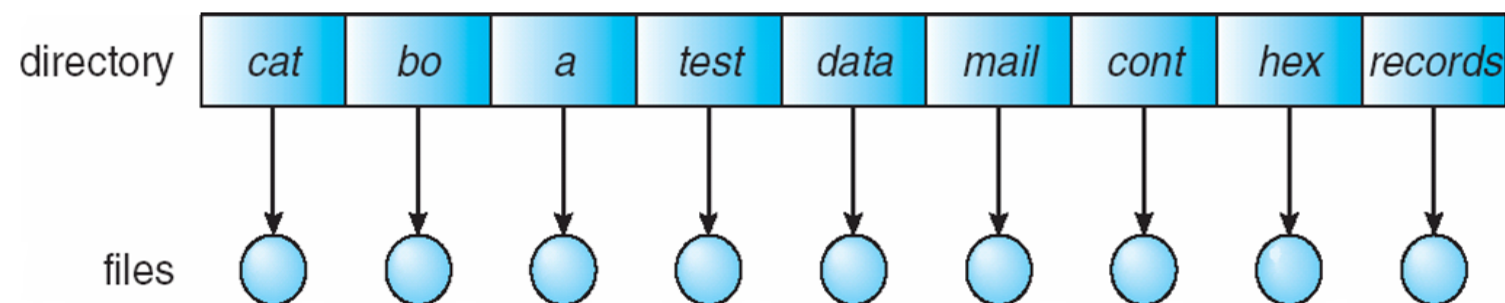


Directory Organization

- Organize directories to achieve
 - **efficiency**: to locate a file quickly
 - **naming**: organize the directory structure to be convenient to users
 - two users can have same name for different files
 - the same file can have several different names
 - **grouping**: provide a way to logically group files by properties
 - e.g., all Java programs, all games, ...
 - ...

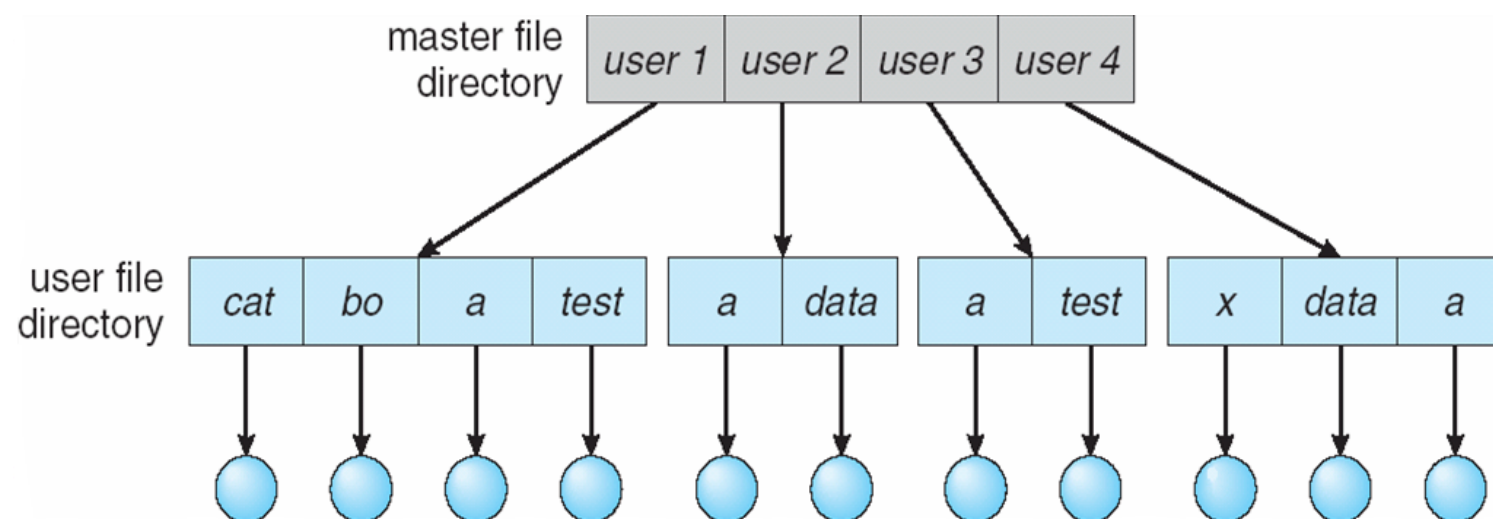
Single-Level Directory

- A single directory for all users
 - naming problems and grouping problems
 - Two users want to have same file names
 - Hard to group files



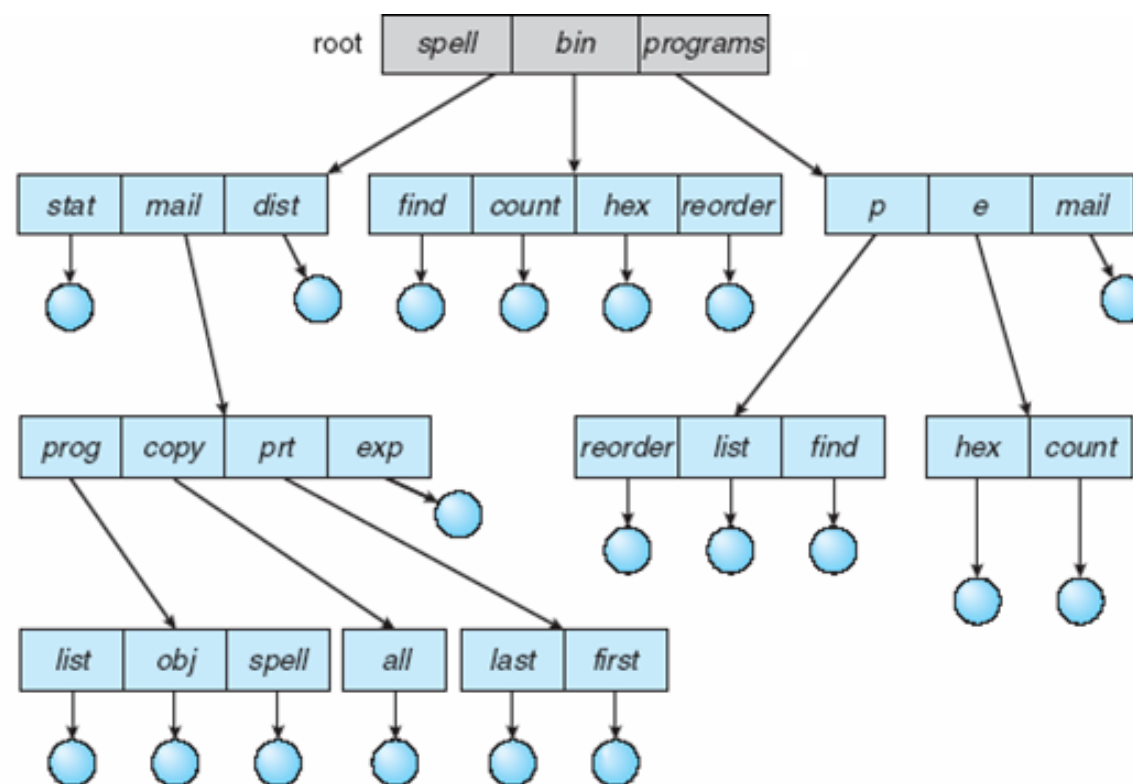
Two-Level Directory

- Separate directory for each user
 - different user can have the same name for different files
 - Each user has his own user file directory (UFD), it is in the master file directory (MFD)
- efficient to search, cannot group files
- How to share files between different users, and how to share the system files?



Tree-Structured Directories

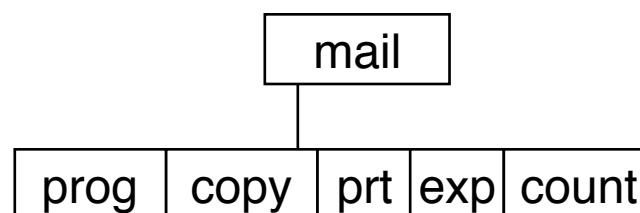
- Files organized into trees
 - efficient in searching, can group files, convenient naming





Tree-Structured Directories

- File can be accessed using **absolute** or **relative** path name
 - absolute path name: /home/alice/..
 - relative path is relative to the **current directory** (*pwd*)
 - creating a new file, delete a file, or create a sub-directory
 - e.g., if current directory is /mail, a **mkdir count** will create /mail/count
- How to share a file/directory? -> it's not allowed



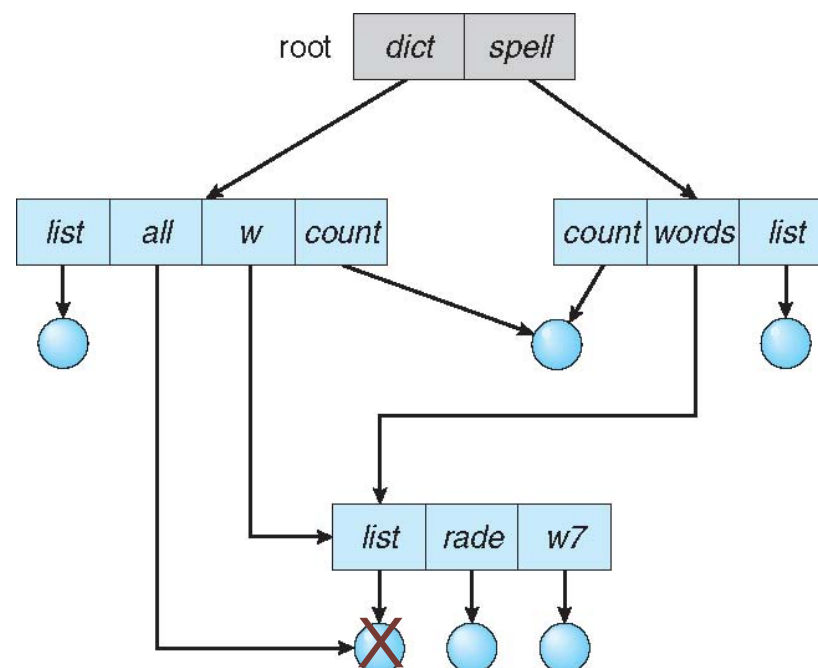


Tree-Structured Directories

- Delete directory
 - If directory is empty, then it's easy to handle
 - If not
 - Option I: directory cannot be deleted, unless it's empty
 - Option II: delete all the files, directories and sub-directories
 - `rm -rf /`

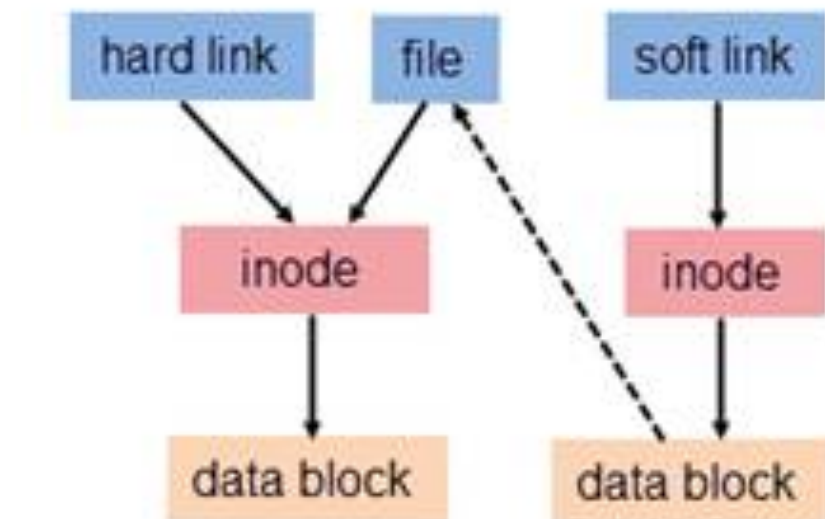
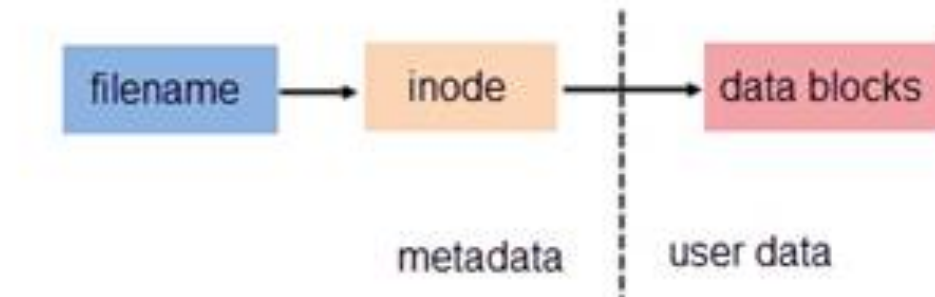
Acyclic-Graph Directories

- Organize directories into acyclic-graphs
 - allow links to a directory entry/files for **aliasing** (no longer a tree)
- Dangling pointer problem:
 - e.g., if delete **file** /dict/all, /dict/w/list and /spell/words/list are dangling pointers
 - Solution: **backpointers/reference counter**
 - backpointers record all the pointers to the entity, a variable size record
 - Or count # of links to it and only (physically) delete it when counter is zero



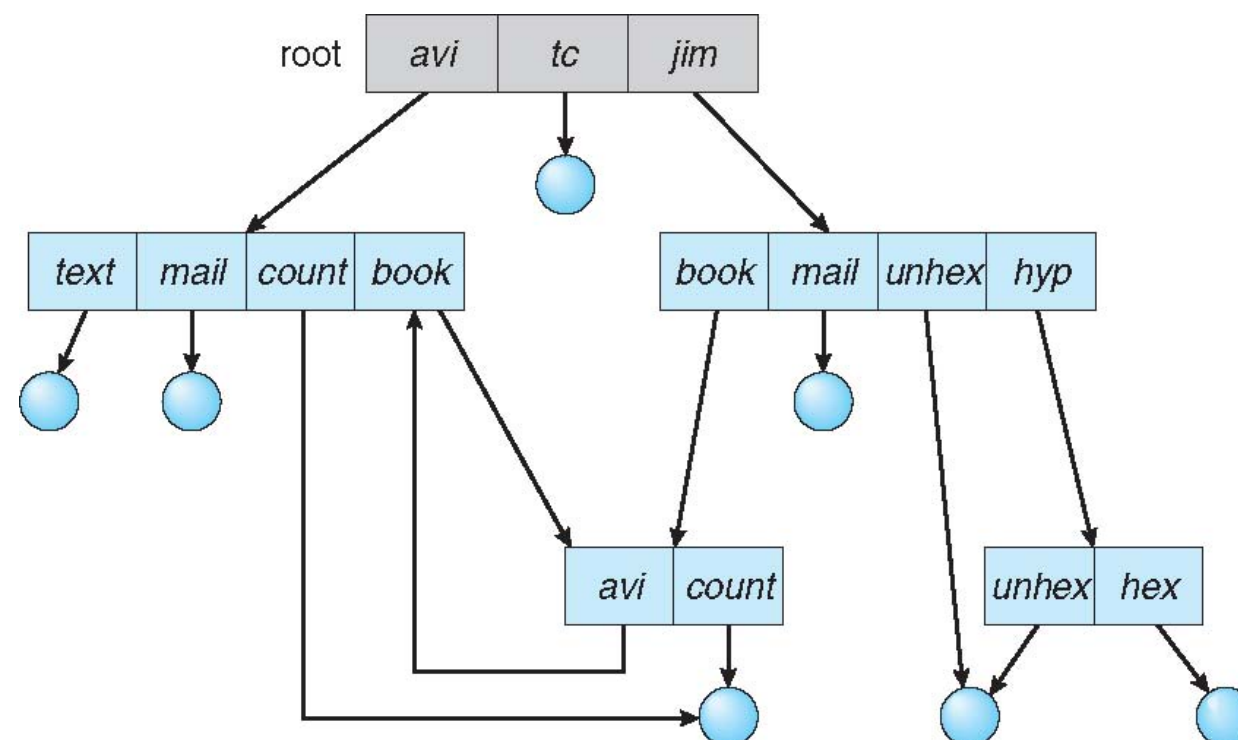
Acyclic-Graph Directories

- Share files
 - Hardlink
 - Reference count
 - Softlink



General Graph Directory

- Allowing arbitrary links may generate cycles in the directory structure
- Solution
 - allow cycles, but use **garbage collection** to reclaim disk spaces
 - every time a new link is added use a **cycle detection** algorithm



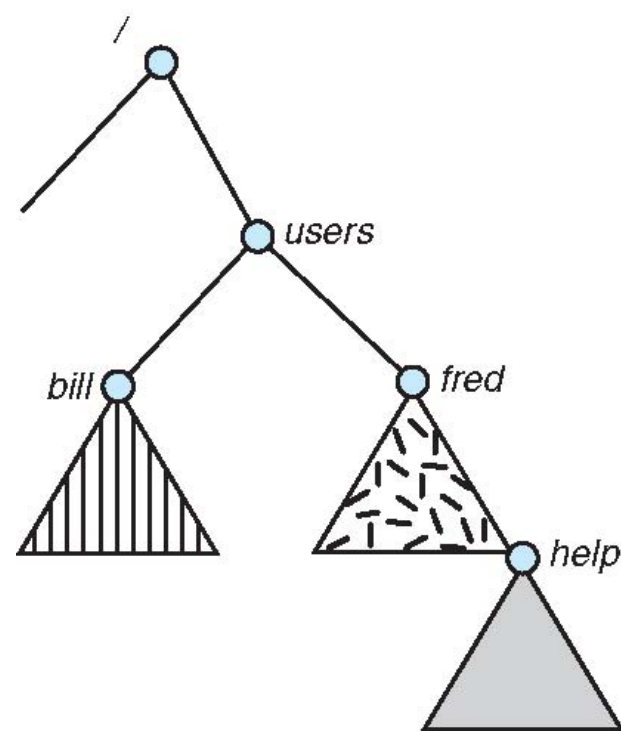


File System Mounting

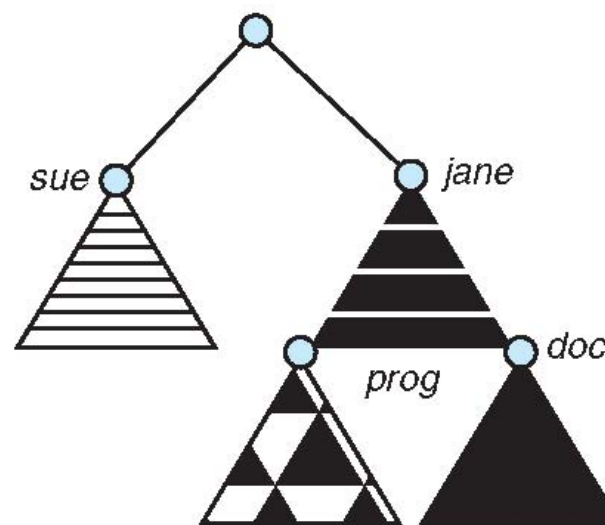
- A file system must be **mounted** before it can be accessed
 - mounting link a file system to the system, usually forms a **single name space**
 - the location of the file system being mounted is call the **mount point**
 - a mounted file system makes the old directory at the mount point **invisible**

File System Mounting

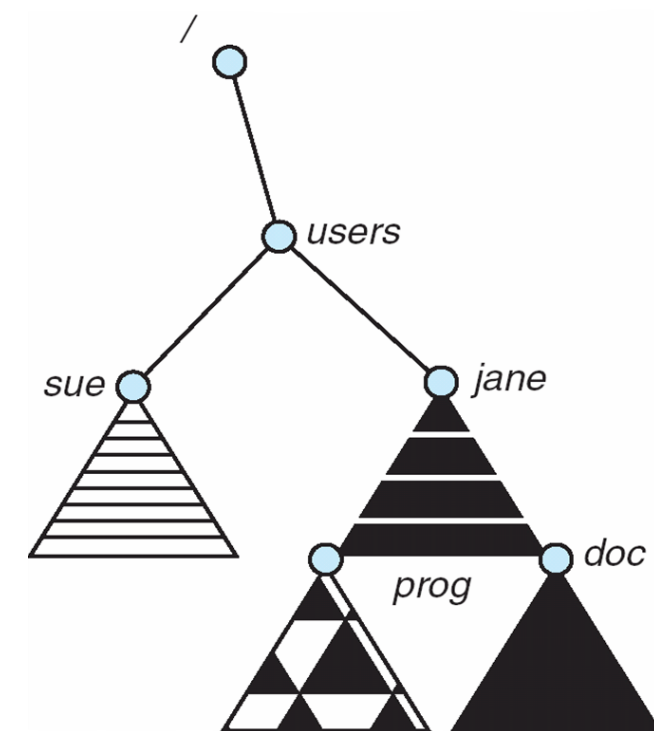
- **a**: existing file system
- **b**: an unmounted partition
- **c**: the partition mounted at **/users**



(a)



(b)



(c)



File Sharing

- Sharing of files on multi-user systems is desirable
 - sharing must be done through a protection scheme
 - **User IDs** identify users, allowing protections to be per-user
 - **Group IDs** allow users to be in groups, permitting group access rights
- On distributed systems, files may be shared across a network
 - Network File System (NFS) is a common distributed file-sharing method



Remote File Sharing

- **Use 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 FS from servers
 - a server can serve multiple clients
 - client and user-on-client identification is complicated
 - server cannot assume the client is trusted
 - standard OS file calls are translated into remote calls
- **NFS** is standard UNIX file sharing protocol, **CIFS** is standard for Windows



Protection

- File owner/creator should be able to control
 - what can be done
 - by whom
- Types of access
 - read, write, append
 - execute
 - delete
 - list



ACL

- Assign each file and directory with an access control list (ACL)
- Advantages: fine-grained control
- Disadvantages
 - How to construct the list
 - How to store the list in directory

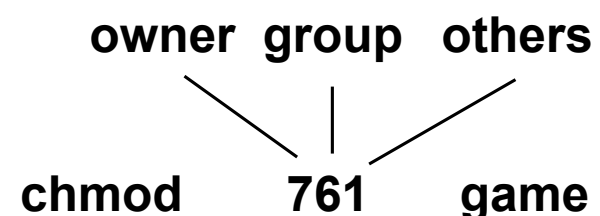


Unix Access Control

- Three modes of access: **read**, **write**, **execute** (encoded in three bits)
- Three classes of users: **owner**, **group**, and **others**

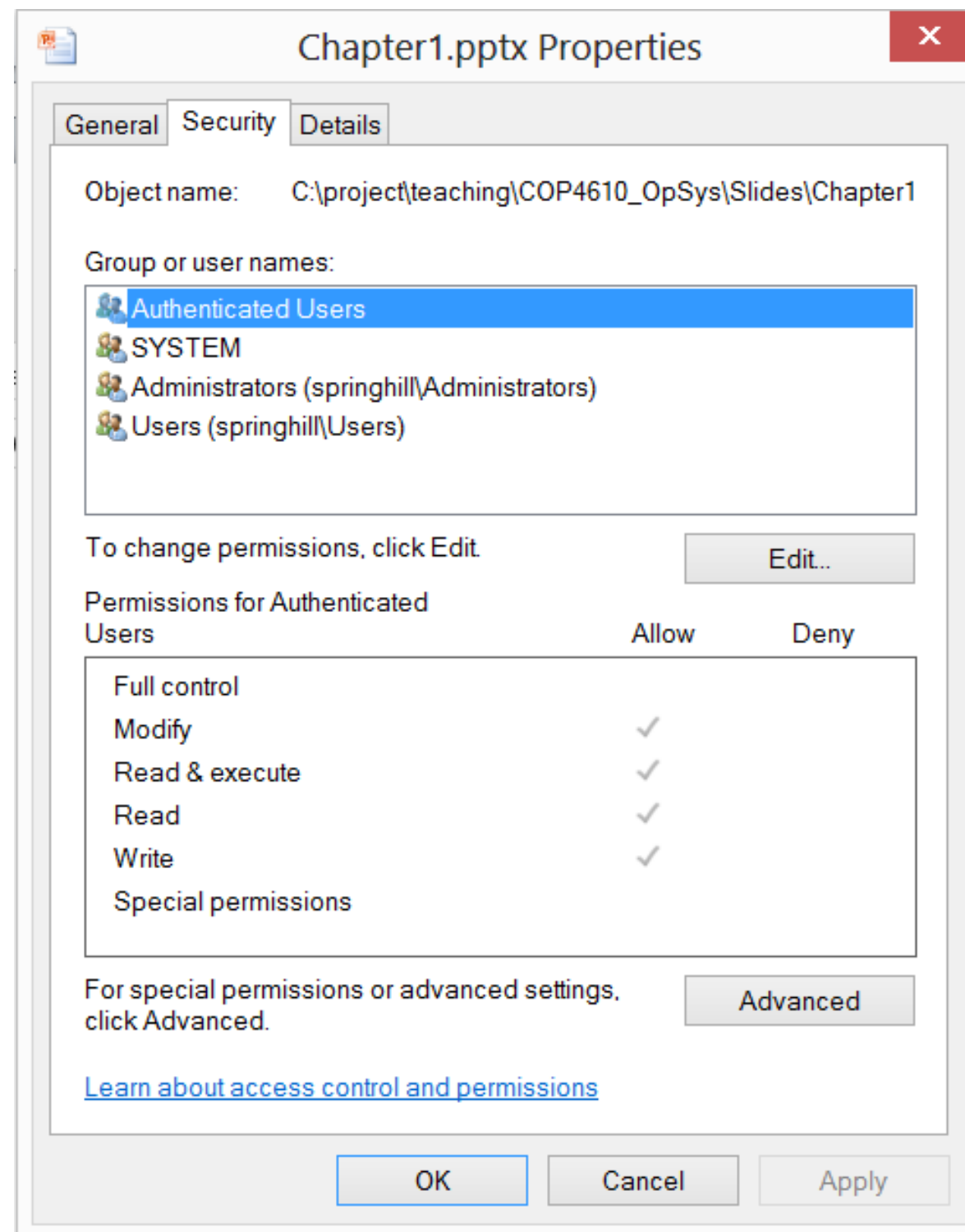
		RWX
a) owner access:	7	1 1 1
b) group access:	6	1 1 0
c) others access:	1	0 0 1

- To grant access to users, create a group and change its access mode
 - in Linux, use **chmod** and **chgrp**





Windows 8 File Access-Control





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/



ACL in practice

```
os@os:~/os2018fall/test$ ls -l
total 0
-rw-rw-r-- 1 os os 0 Dec 18 23:21 testacl
os@os:~/os2018fall/test$ getfacl testacl
# file: testacl
# owner: os
# group: os
user::rw-
group::rw-
other::r--

os@os:~/os2018fall/test$ setfacl -m u:test:rw testacl
os@os:~/os2018fall/test$ getfacl testacl
# file: testacl
# owner: os
# group: os
user::rw-
user:test:rw-
group::rw-
mask::rw-
other::r--

os@os:~/os2018fall/test$ ls -l
total 0
-rw-rw-r--+ 1 os os 0 Dec 18 23:21 testacl
os@os:~/os2018fall/test$
```