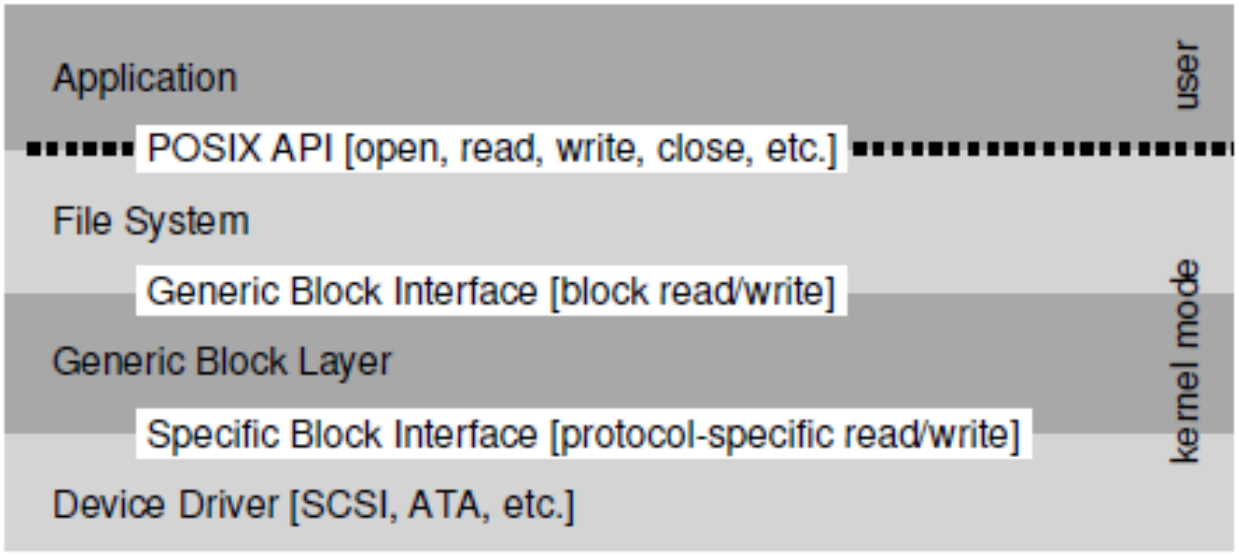
Lecture 12

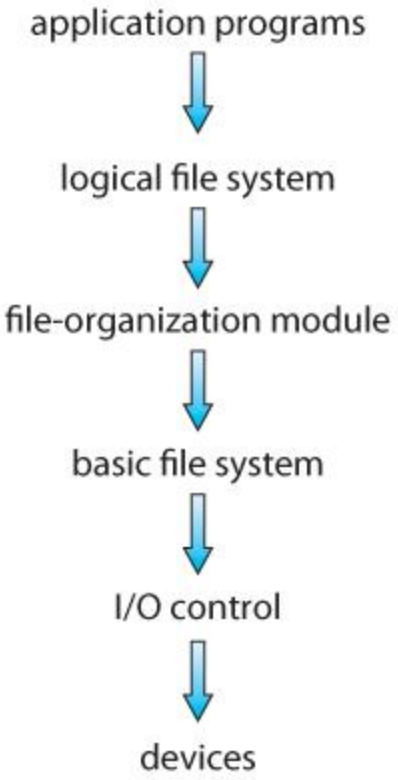
File System and Its Implementation

Previously …

- We have looked at general I/O management in terms of Devices, Ports and Buses
- We have studied a typical I/O Device and compared alternative I/O Protocols
  - Polling, Interrupt and Direct Memory Access
- We have looked at Mass Data Storage, in particular Hard Disk Drives and different disk scheduling algorithms
- We now go on to look at the File System

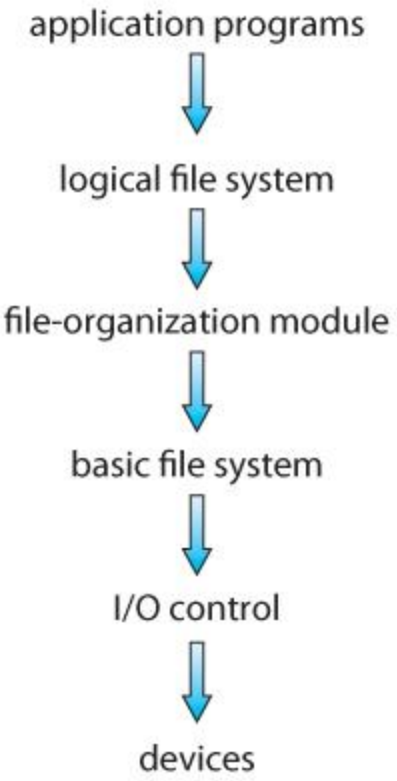
Recall: File System Stack





Introduction

- Each level in the diagram uses the features of a lower level
  - Last time we discussed the I/O control (Device Drivers and ISR) and how data is transferred between Main Memory and Hard Disk Drive
- We now focus on the higher levels
  - First, how to abstract the file system and operate it at the logical file system level
  - Second, how to implement and manage the file system



Abstraction of the File System

- We use the concept of Files and Directories to abstract the File System
- The OS abstracts from the physical properties of data storage to define a logical storage unit, called a File
  - It is a linear array of bytes that the user can read and write
- A Directory is like a file, but contains a list of name pairs
  - Each pair is (user-readable name, low-level name) of a file or sub-directory – the low-level name is often called an inode number
  - By placing directories within other directories, users are able to build an arbitrary directory tree (or directory hierarchy)

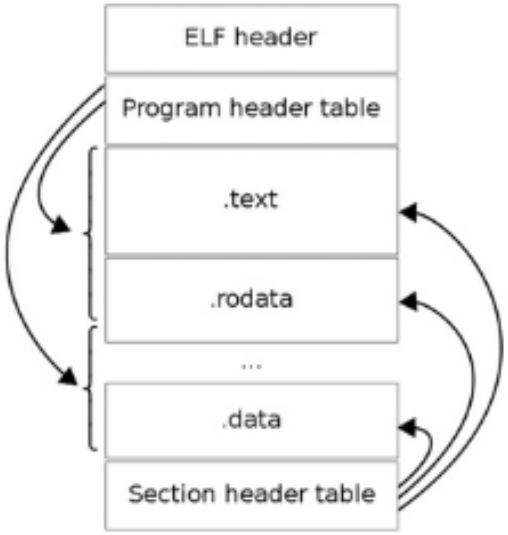
File Structure (1)

- Overview
  - All files must conform to file structures that the OS understands
    - Example: executable file structure includes the location of first instruction
  - Some OSs impose and support a minimal file structure
    - Linux treats all files as a sequence of bytes
    - Application must includes code to understand file structure
  - Some OSs have many different file structures which are indicated by file types
    - This needs more complicated file support

Example of Executable File Structure

ELF





Mach-O



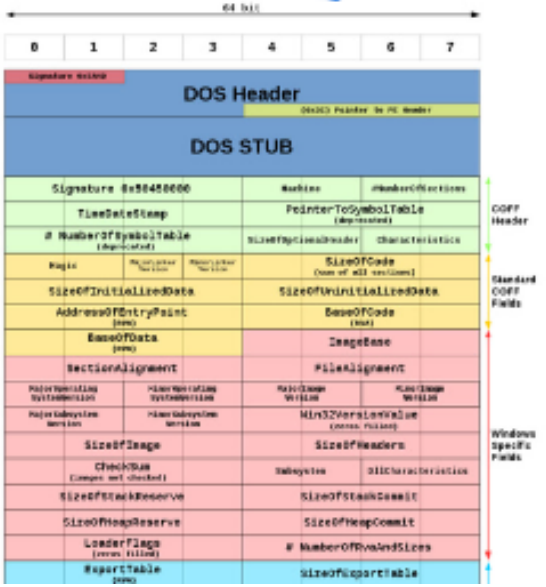
Mach-O header

Mach-O File header[4]

Offset	Bytes	Description
0	4	Magic number
4	4	CPU type
8	4	CPU subtype
12	4	File type
16	4	Number of load commands
20	4	Size of load commands
24	4	Flags
28	4	Reserved (64-bit only)

PE





File Structure (2)

- File Attributes
  - Name: Some OSs give special significance to names and particularly extensions (.exe or .txt etc.) in the user readable name
  - Identifier: the unique tag or low-level name, usually a number (inode number) that is not user readable
  - Type: text, executable, other binary, etc.
  - Location: a pointer to the device where the file is stored
  - Size, Protection or Permission, Time and Date, and Owner User ID
- File Types
  - Windows uses a file extension to indicate the type of file
  - Mac OS stores type in attribute
  - Linux uses symbols to describe the type

```
crw-rw-rw-   1 root root      1,   8 Dec  6 11:17 random
drwxr-xr-x   2 root root          60 Dec  6 11:17 raw
lrwxrwxrwx   1 root root           4 Dec  6 11:17 root -> sda1
lrwxrwxrwx   1 root root           4 Dec  6 11:17 rtc -> rtc0
crw-rw----   1 root root    254,   0 Dec  6 11:17 rtc0
lrwxrwxrwx   1 root root           3 Dec  6 11:17 scd0 -> sr0
brw-rw----   1 root disk      8,   0 Dec  6 11:17 sda
brw-rw----   1 root disk      8,   1 Dec  6 11:17 sda1
```

file permissions,
number of links,
owner name,
owner group,
file size,
time of last modification
file/directory name

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

File Structure (3)

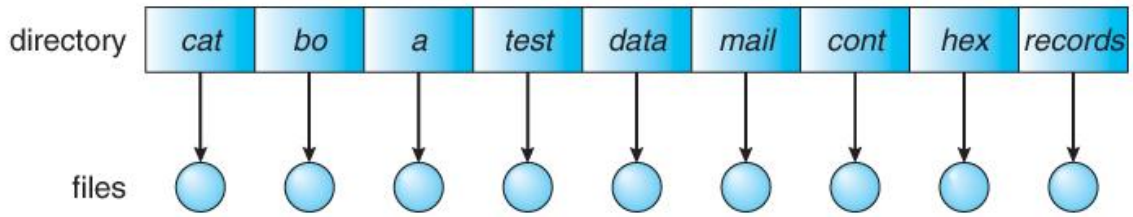
- File Access Methods
  - Sequential Access
    - Supports operations: read next, write next, rewind, and skip
  - Random Access (Direct Access)
    - Can jump to any record n (relative to start) and read or write that record
    - Supports operations: read n, write n, and jump to n
  - Indexed Access
    - Requires an index file, pointing to the ordinary relative file
    - Very large files may require a multi-tiered indexing scheme

File Operations

- Create
  - First, a location must be found on the device for the file
  - Second, the detail must be record in the directory where the file is created
- Read or Write
  - Need a read pointer or write pointer to the record in the file where the next read or write is to take place
- Delete
  - Search the directory entries for the file name then release the file space and erase the directory entry

Directory Structure (1)

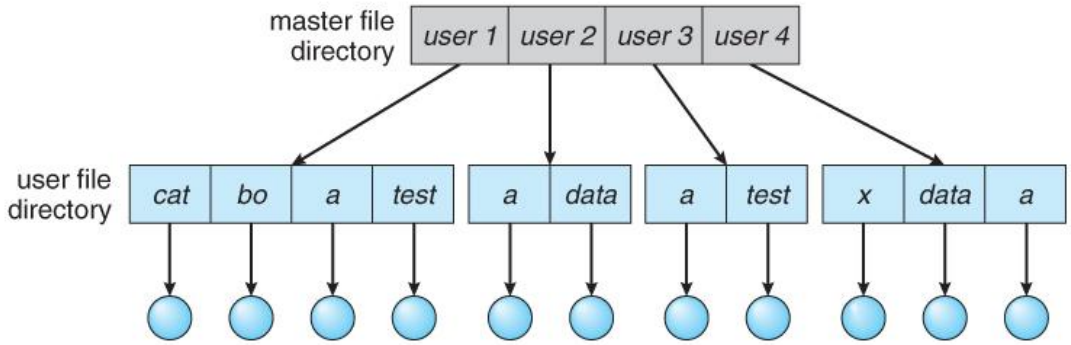
- One Level Directory



A single directory for all users

Cannot have the same file name for different users

- Two Level Directory

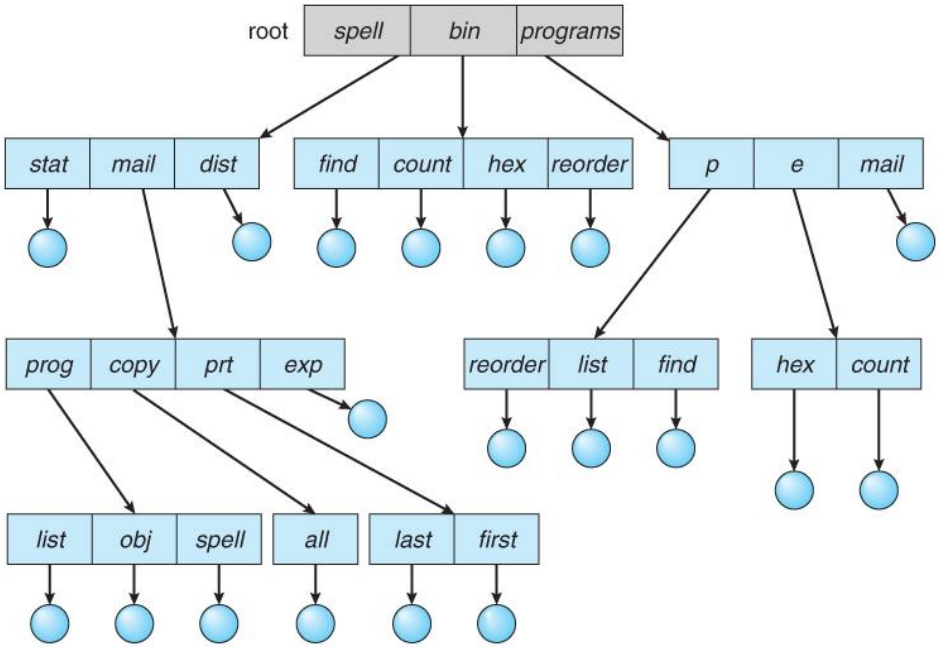


Separate directory for each user

No files in the master file directory

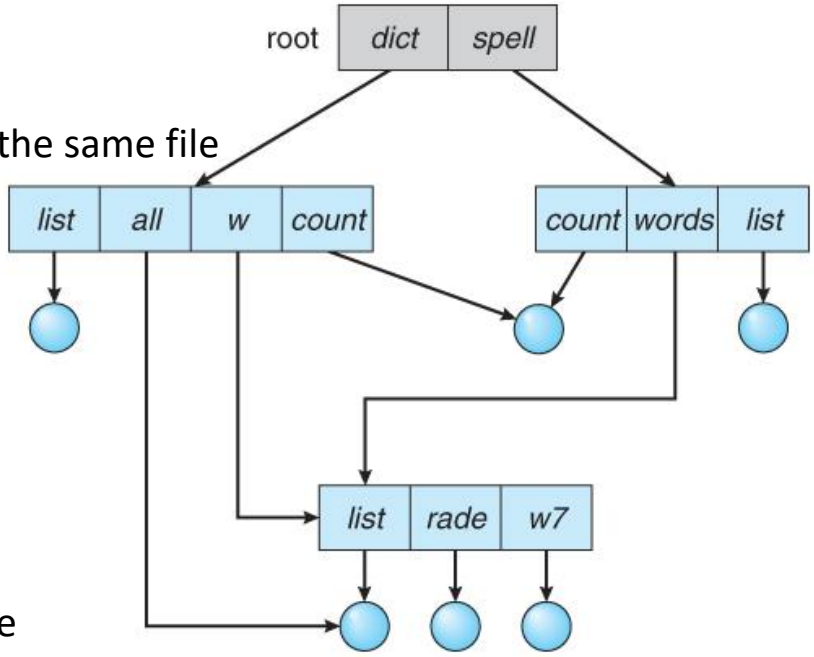
Directory Structure (2)

- Tree Structured Directory
  - Access files with absolute pathnames or relative pathnames
  - Leads to concept of current directory, e.g. cd /spell/mail/prog
  - Efficient searching
  - Grouping capability
    - Users can organize their directories according to their use



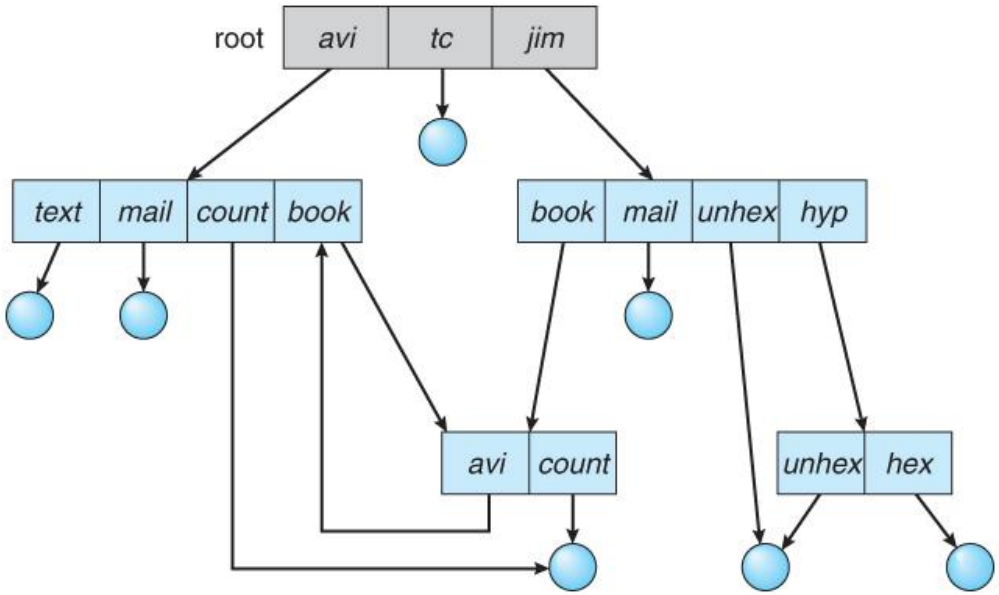
Directory Structure (3)

- Acyclic Graph Directory
  - The same file may need to be accessed in more than one place in the directory structure
  - Use a special file type, called a link
    - Hard link
      - Multiple directory entries refer to the same file
      - Valid only for ordinary files
      - Need reference count in the file control block of how many directories refer to this file
    - Symbolic link or Soft link
      - Contains information about where to find the linked file (pathname)
      - What happen when the original file is deleted or moved?



Directory Structure (4)

- General Graph Directory
  - Soft link can also be used to link to a directory
  - Cycles can lead to infinite loop problems
    - How do we guarantee no cycles?
    - Every time a new link is added can use a cycle detection algorithm



Directory Operations

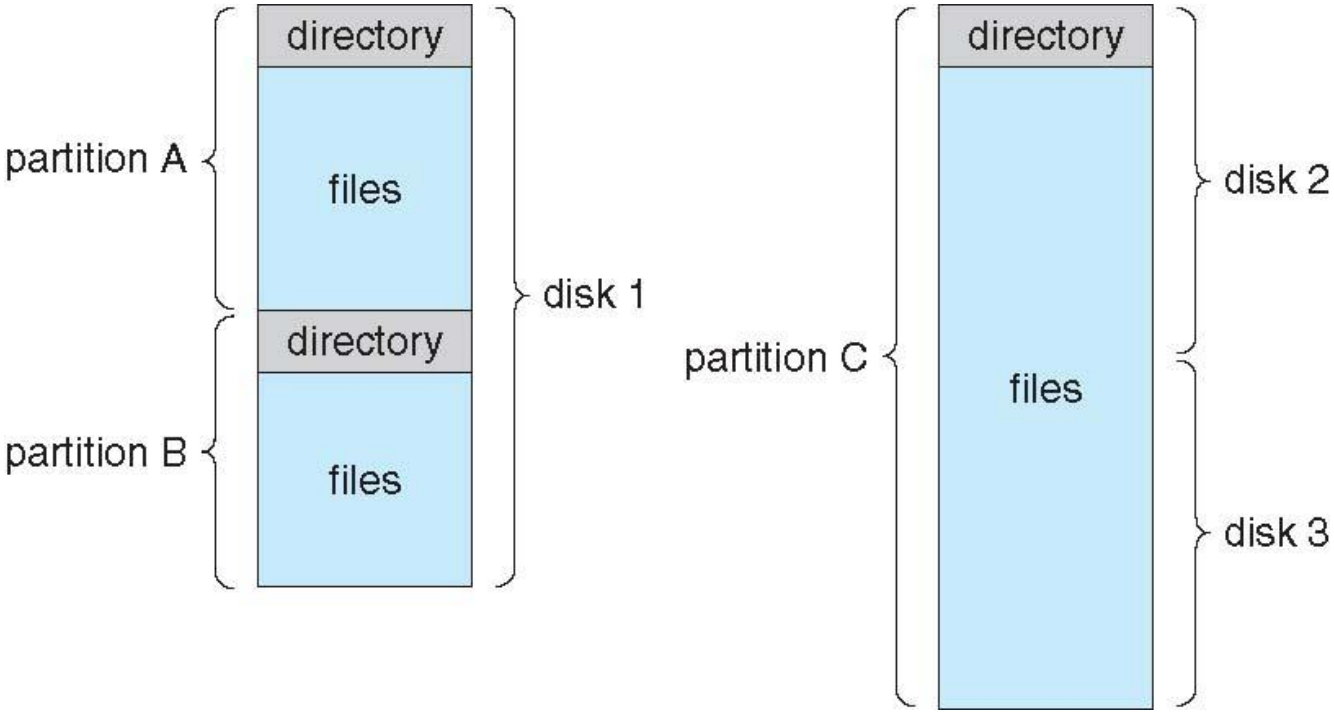
- Make
  - When the directory is created, it is empty, except for two entries
    - A reference to its parent directory, (“..” in Linux)
    - A reference to itself (“.” in Linux)
- Write
  - The OS does not usually allow writing to the directory entry directly
  - However, operations on files or subdirectories (e.g. create, rename, delete, etc.) involve updating the parent directory
- Read
  - Allows the directory entries to be read, where each entry shows the mapping of the user-readable name to the inode number, along with other details
- Delete
  - Is the directory empty? If not, return error or delete all files and subdirectories recursively (dangerous)

File System Implementation

- We have looked at the abstraction of a file system and its interface
- We now go on to look at the file system implementation
  - This involves implementing the files allocation methods and the data structure of directory implementation
- Before that …
  - The file system needs to store important information on the disk of various types:
    - Boot Control Block – information needed by system to boot OS
    - Volume Control Block – details of volume (partition)
    - File Control Block (inode in Linux) – details about the file
  - The OS needs to create partitions and mount a file system before use

Partitions

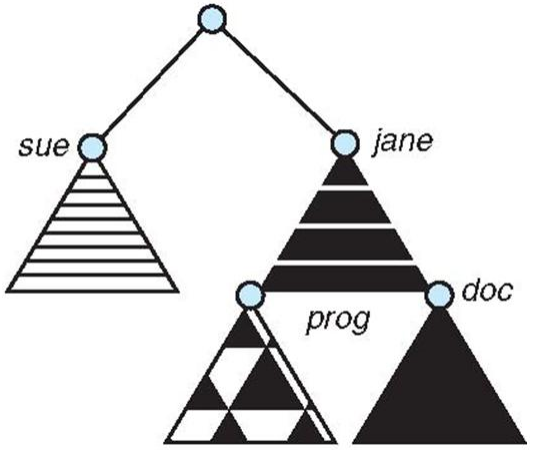
- A file system usually contains at least one partition
- A disk can be separated into a number of partitions
- Some OSs support partitions across more than one disk



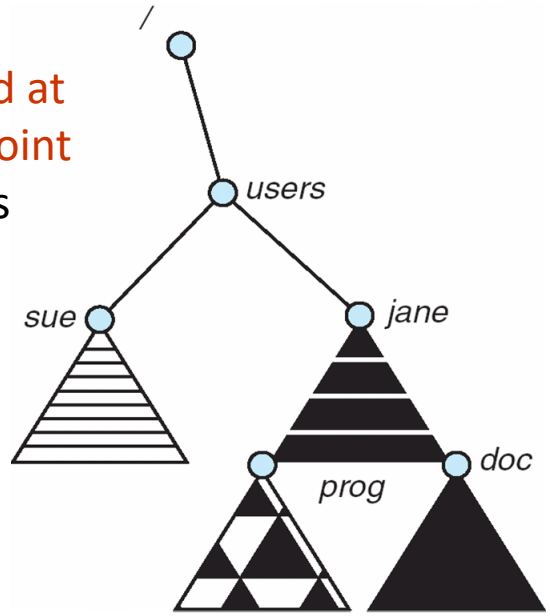
Mounting a File System

- A file system must be mounted before use – the basic idea is to combine multiple file systems into one large file system tree
- This requires the name of the storage device and partition containing the file system to mount and the location within the directory structure to attach the file system, called the mount point

Unmounted file system



Mounted at mount point /users

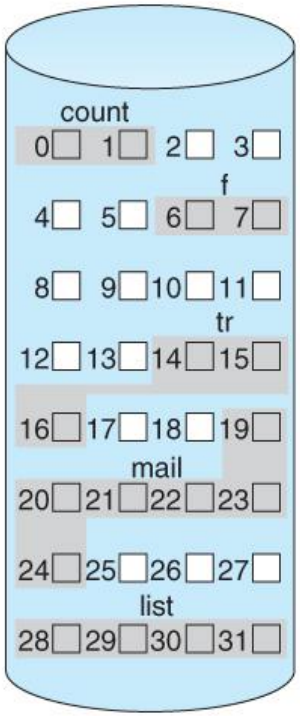


File Allocation Methods

- The file allocation method refers to how disk blocks are allocated for files
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation
    - Linked Scheme
    - Multilevel Index Scheme
    - Combined Scheme

Contiguous Allocation

- Each file occupies a set of contiguous blocks
  - Advantages
    - Simple – only need the start block number and number of blocks
    - Best performance in most cases
  - Disadvantages
    - Difficult to know file size at create time
    - Over-estimation of the file's final size wastes disk space and increases external fragmentation
    - Under-estimation may require that a file be moved if it grows beyond its originally allocated space

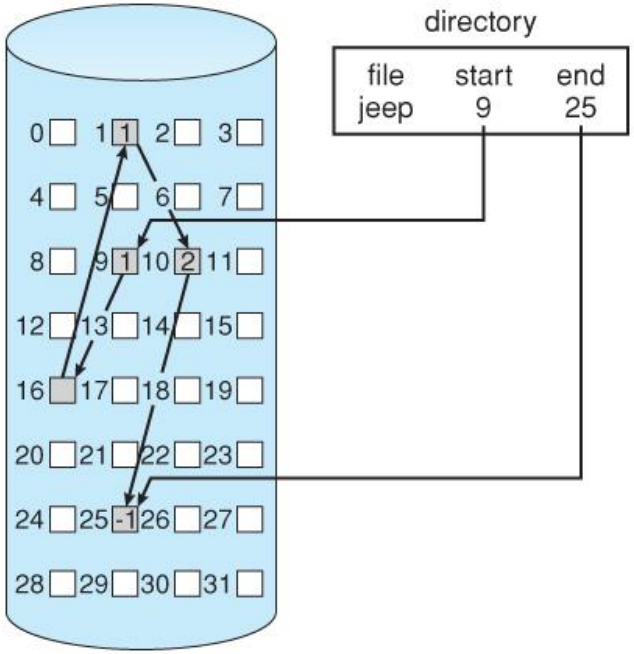


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

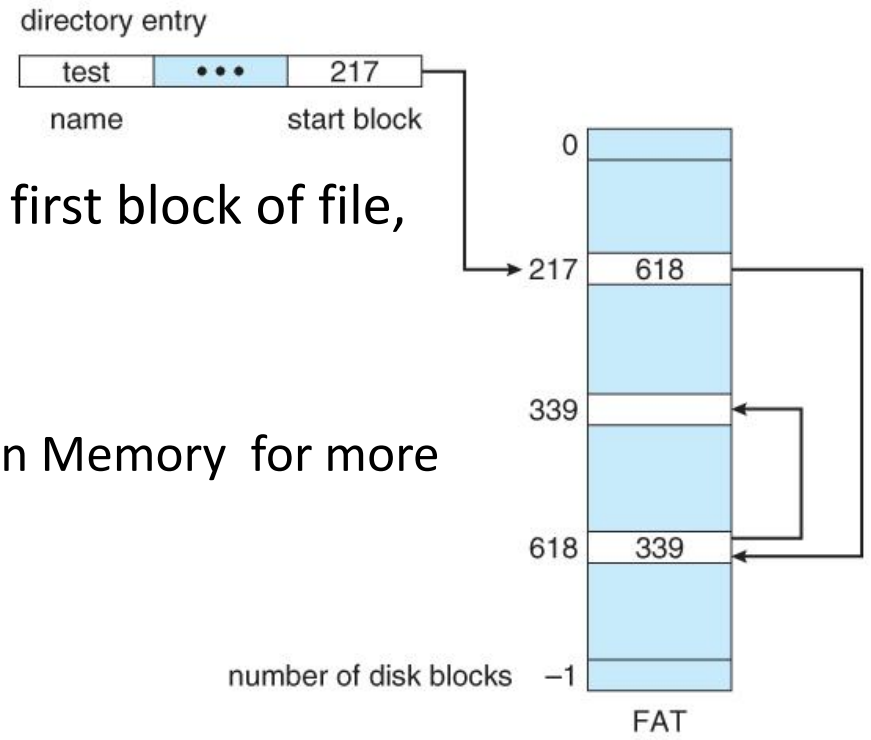
Linked Allocation (1)

- Each file is a linked list of blocks that are scattered on the disk, where each block contains a pointer to the next block
  - Advantages
    - No external fragmentation
    - Does not require pre-known file sizes
    - Allows files to grow dynamically at any time
  - Disadvantages
    - Only efficient for sequential access files
    - Space wasted by pointers
    - Allocating clusters of blocks reduces the wasted space, at the cost of internal fragmentation



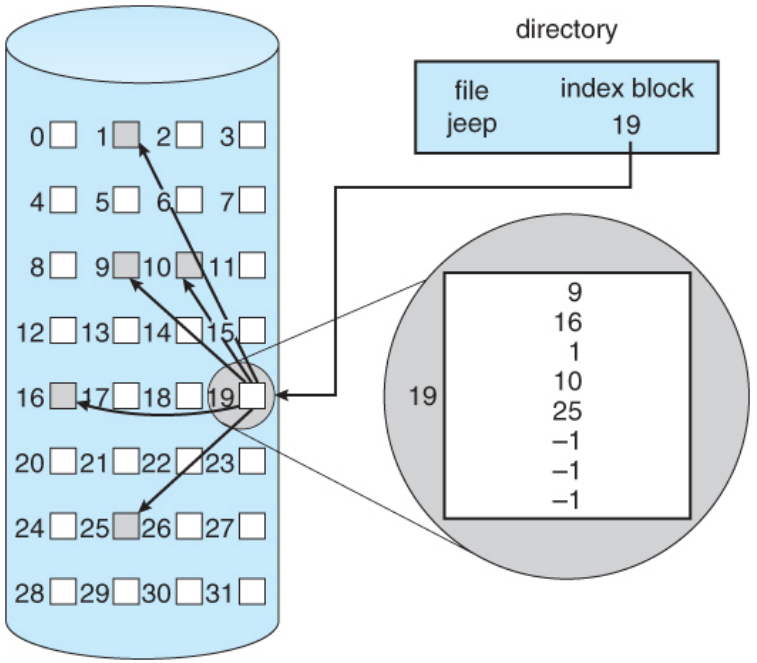
Linked Allocation (2)

- The File Allocation Table (FAT) is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the partition
  - The directory contains the first block of file, used to index the FAT
  - Advantage
    - FAT can be cached in Main Memory for more efficiency



Indexed Allocation

- This combines all of the block numbers for accessing each file into a special index block (for that file)
  - Advantages
    - Allows random (direct) access without external fragmentation
  - Disadvantages
    - Some disk space is wasted because an entire index block must be allocated for each file
  - How should the index be implemented?
    - How can we support large files that need more than one index block?

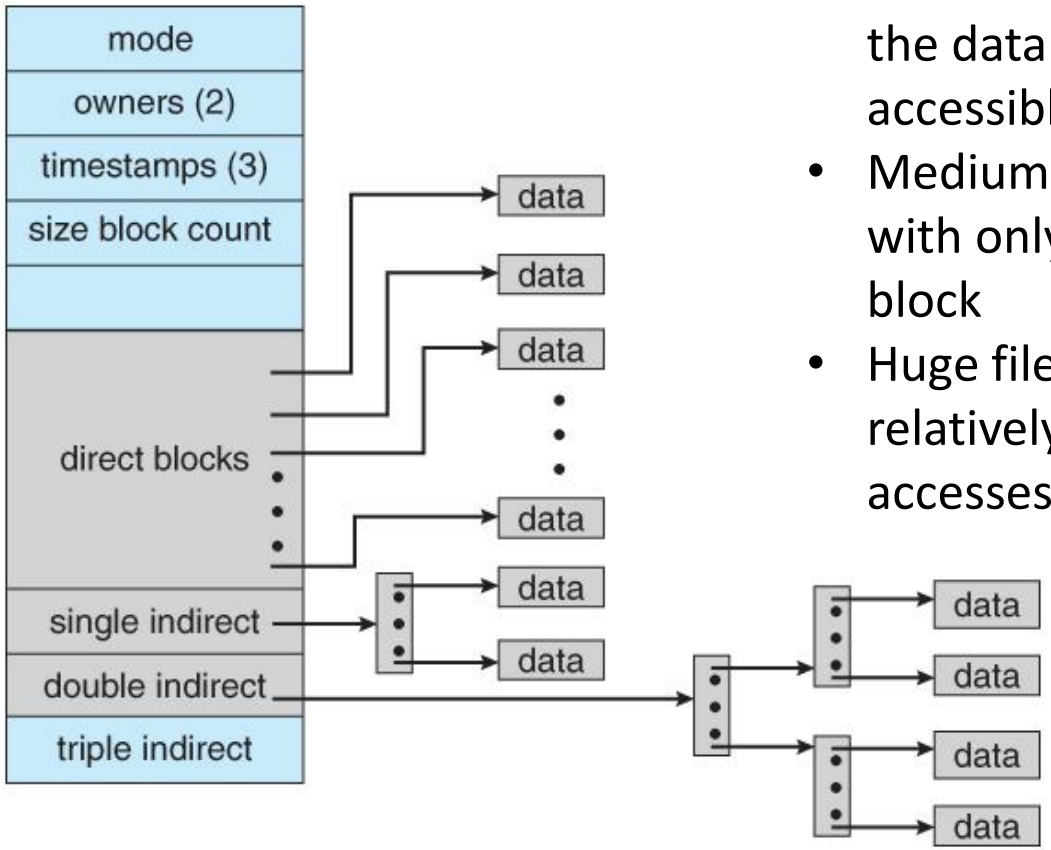


Index Block Implementation (1)

- Linked Scheme
  - The first index block contains header information, the first set of block numbers, and if necessary a pointer to additional linked index blocks
- Multilevel Index Scheme
  - The first index block contains a set of pointers to secondary index blocks, which in turn contain pointers to the actual data blocks
  - May have more than one level of indirect index blocks
- Combined Scheme
  - This is the scheme used in UNIX/Linux inodes
  - The first 12 or so data block pointers are stored directly in the inode
  - Then single, double, and triple indirect pointers provide access to more data blocks as needed

Index Block Implementation (2)

- Combined Scheme (continued)



- For small files (which many are), the data blocks are readily accessible
- Medium size files are accessible with only a single indirect index block
- Huge files are still accessible using a relatively small number of disk accesses

File Allocation Performance

- The best method is different for sequential access than for random access, and is also different for small files than for large files
  - Contiguous allocation is good for sequential and random access, but has disadvantages, especially when the size is unknown at create time
  - Linked allocation is good for sequential, but not for random access
- Some OSs require specifying how the file is to be used at create time
  - Use linked allocation for sequential access and contiguous or indexed allocation for random access
- Some OSs automatically switch their allocation method
  - Use contiguous allocation for small files, and automatically switch to indexed allocation when the file size exceeds a threshold

Data Structure of Directory Implementation

- Directories need fast search, insert, and delete operations, with a minimum of wasted disk space
  - Linear List
    - A linear list of file names with corresponding inode number
    - Simplest and easiest directory structure to set up, but finding a file requires a linear search
    - Sorting the list makes searching faster, at the expense of more complex insertions and deletions
    - A linked list makes insertions and deletions into a sorted list easier, with an overhead for the links
    - More complex data structures, such as B-trees, could also be considered
  - Hash Table
    - A hash table can be used to speed up searching
    - Collisions need to be handled, e.g. chained-overflow method

Journaling File System

- A Journaling (Log Based) file system records each update to the file system as a transaction (set of changes)
- All transactions are first written to a log (separate device or section of disk)
  - A transaction is considered committed once it is written to the log
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system structures
  - After the file system structures are updated, the transaction is removed from the log
- If the system crashes, all remaining transactions in the log must still be performed
  - Faster recovery from crash, removes chance of inconsistency

Redundant Array of Independent Disks

- Redundant Array of Independent (Inexpensive) Disks
  - The idea is to employ a group of hard disks together with some form of duplication, either to increase reliability or to speed up operations, or both
- Improvement of Reliability via Redundancy
  - Mirroring, in which a system stores identical data on two or more disks
- Improvement in Performance via Parallelism
  - Striping, where data out is spread across multiple disks that can be accessed simultaneously
    - Bit-level striping – bits of each byte are striped across multiple disks
    - Block-level striping – data is spread across multiple disks block-by-block
- There are a number of different schemes that combine mirroring and striping

RAID Level

Level 0: Striping, no mirroring

Level 1: Mirroring, no striping

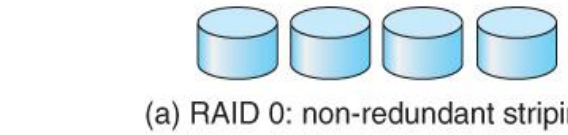
Level 2: Error-correcting code (ECC) on additional disks instead of mirroring

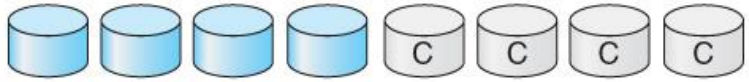
Level 3: Bit-striping and dedicated parity disk

Level 4: Block-striping and dedicated parity disk

Level 5: Block-striping with rotating parity bits (the same disk cannot hold both data and parity for the same block)

Level 6: Block-striping with dual rotating parities, can handle two disk failures


(a) RAID 0: non-redundant striping.



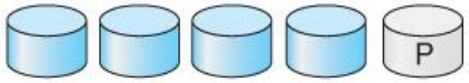
(b) RAID 1: mirrored disks.



(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



(f) RAID 5: block-interleaved distributed parity.



(g) RAID 6: P + Q redundancy.

Combined RAID Levels

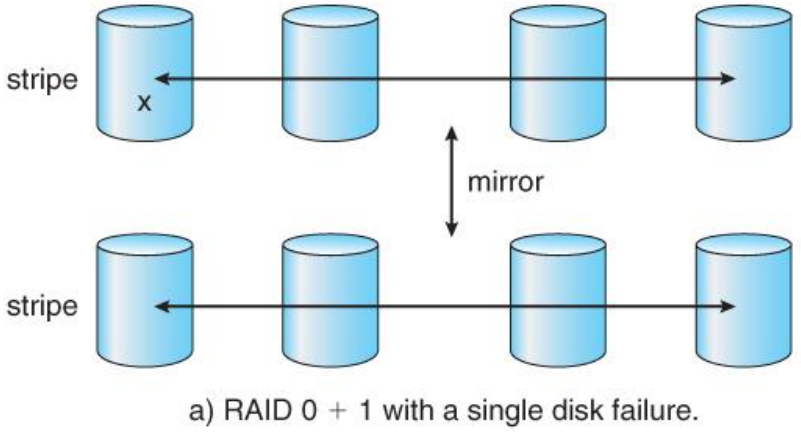
Also two RAID levels that combine levels 0 and 1 in different ways:

RAID level 0+1: disks are first striped, and then mirrored to another set

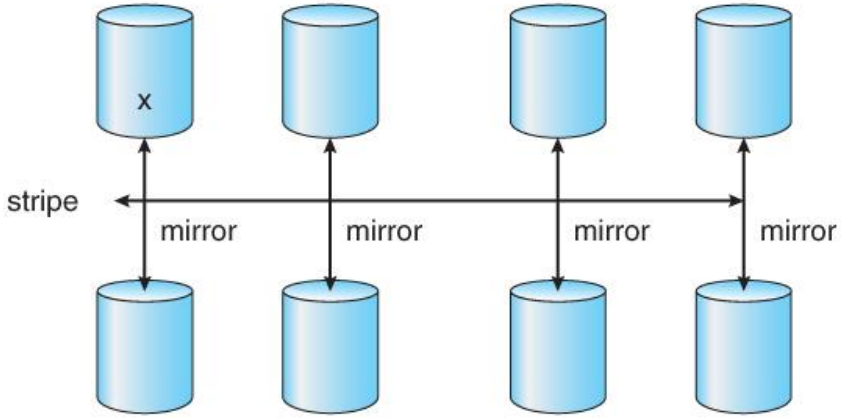
A single disk failure will wipe out one stripe set, but the other set can still be used – cannot handle two disk failures

RAID level 1+0: disks are mirrored in pairs, and then striped

Can handle as many as four disk failures, as long as no two of them are in the same mirror pair



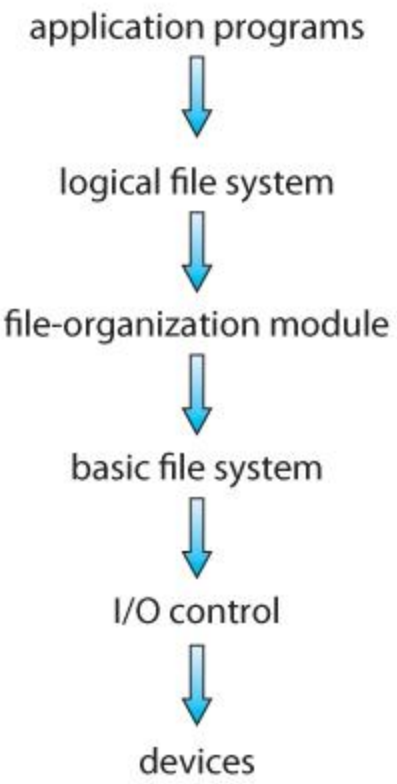
a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

Summary of Lecture

- We have looked at the abstraction of the file system and its implementation
  - The logical file system manages the directory structure
  - The file organization module knows about files and how disk blocks are allocated
  - The basic file system works with the I/O control and device drivers to store and retrieve data



Summary of Course

- In this course we have looked at three main themes
  - Virtualization
    - CPU Virtualization
    - Memory Virtualization
  - Concurrency
    - Concurrency between threads of the same process
    - Concurrency between different processes
  - Persistence
    - I/O Devices and Mass Data Storage
    - File System and its Implementation

Acknowledgements

- Many of the slides in this course include material from the following sources
- We thank all the authors for making their materials available
  - R.H. Arpaci-Dusseau and A.C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, http://www.ostep.org
  - A. Silberschatz, P.B. Galvin, and G. Gagne: Operating System Concepts, http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/
  - J. Bell: Course Notes https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/
  - S. Buettcher: Course Notes http://cnx.org/content/m27323/latest/?collection=col10785/latest