

Final Review 04



Operating Systems
Wenbo Shen

Summary

- Computer architecture
- OS introduction
- OS structures
- Processes
- IPC
- Thread
- Scheduling
- Synchronization
- Deadlock

Summary

- Memory - segmentation
- Memory - paging
- Virtual memory
- Virtual memory - Linux
- Mass storage
- IO
- FS interface
- FS implementation
- FS in practice

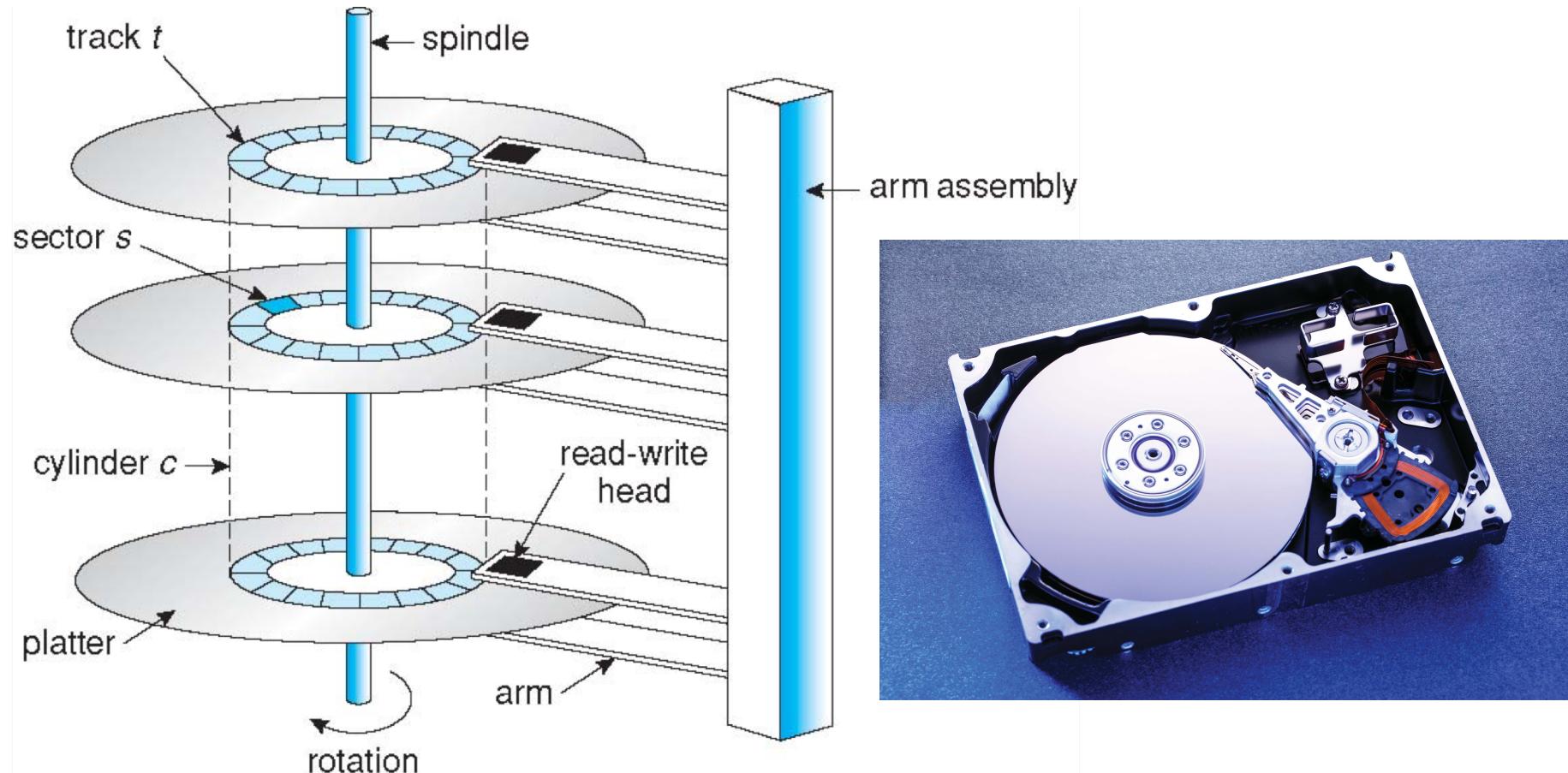
11: Mass storage



Disk Structure

- Disk drives are addressed as a 1-dimensional arrays of logical blocks (LBA)
 - logical block is the smallest unit of transfer
- Logical blocks are mapped into **sectors** of the disk sequentially
 - sector 0 is the first sector of the first track on the outermost **cylinder**
 - mapping proceeds in order
 - first through that **track**
 - then the rest of the tracks in that **cylinder**
 - then through the rest of the cylinders from outermost to innermost
 - logical to physical address should be easy
 - except for bad sectors

Moving-head Magnetic Disk



Nonvolatile Memory Devices

- If disk-drive like, then called solid-state disks (SSDs)
- Other forms include **USB drives** (thumb drive, flash drive), DRAM disk replacements, surface-mounted on motherboards, and main
- Can be **more reliable** than HDDs
- More expensive per MB
- Maybe have shorter life span - need careful management
- Less capacity, but much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency
 - First come first serve is good

Disk Scheduling

- OS is responsible for using hardware efficiently
 - for the disk drives: a fast access time and high disk bandwidth
 - **access time**: seek time (roughly linear to seek distance) + rotational latency
 - **disk bandwidth** is the speed of data transfer, data /time
 - data: total number of bytes transferred
 - time: between the first request and completion of the last transfer

Disk Scheduling

- Disk scheduling chooses which pending disk request to service next
 - concurrent sources of disk I/O requests include OS, system/user processes
 - idle disk can immediately work on a request, otherwise os queues requests
 - each request provide I/O mode, disk & memory address, and # of sectors
 - OS maintains a queue of requests, per disk or device
 - optimization algorithms only make sense when a queue exists
- In the past, operating system responsible for queue management, disk drive head scheduling
 - Now, built into the storage devices, controllers - firmware
 - Just provide LBAs, handle sorting of requests
 - Some of the algorithms they use described next

Disk Scheduling

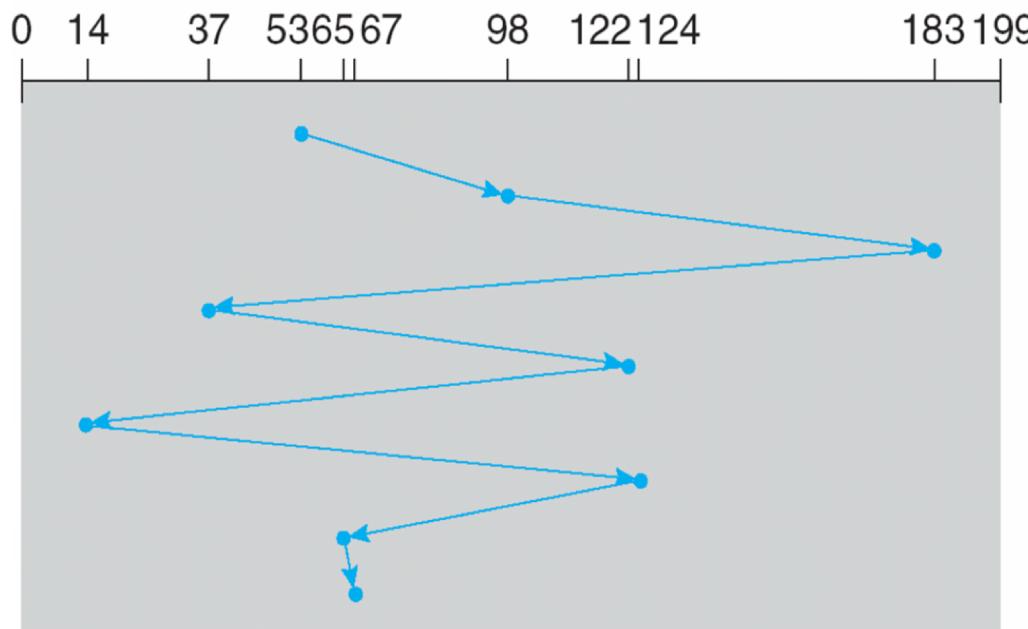
- Disk scheduling usually tries to minimize **seek time**
 - rotational latency is difficult for OS to calculate
- There are many disk scheduling algorithms
 - FCFS
 - SSTF
 - SCAN
 - C-SCAN
 - C-LOOK
- We use a request queue of “98, 183, 37, 122, 14, 124, 65, 67” ([0, 199]), and initial head position 53 as the example

FCFS

- First-come first-served, simplest scheduling algorithm
- Total head movements of 640 cylinders

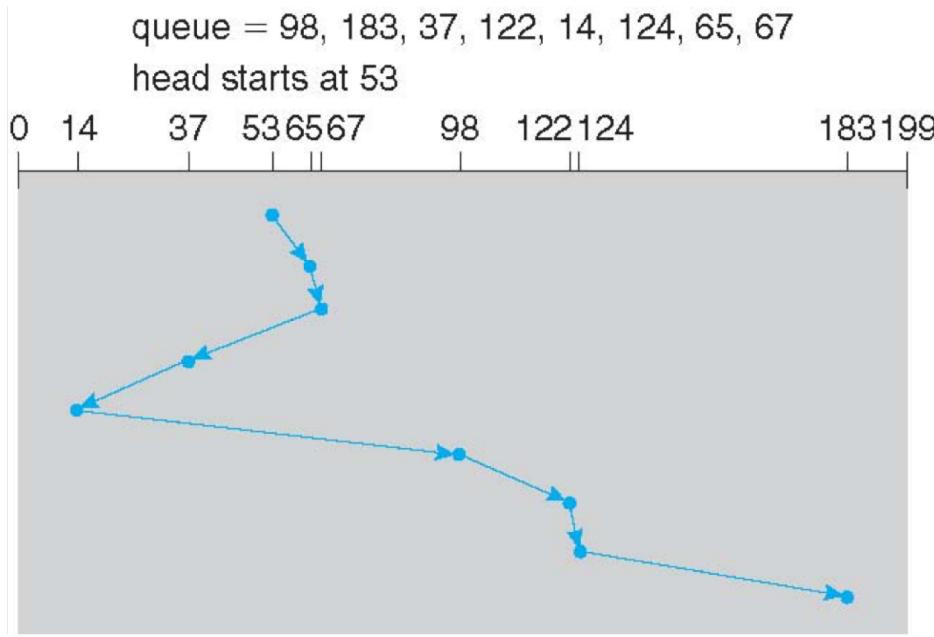
queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



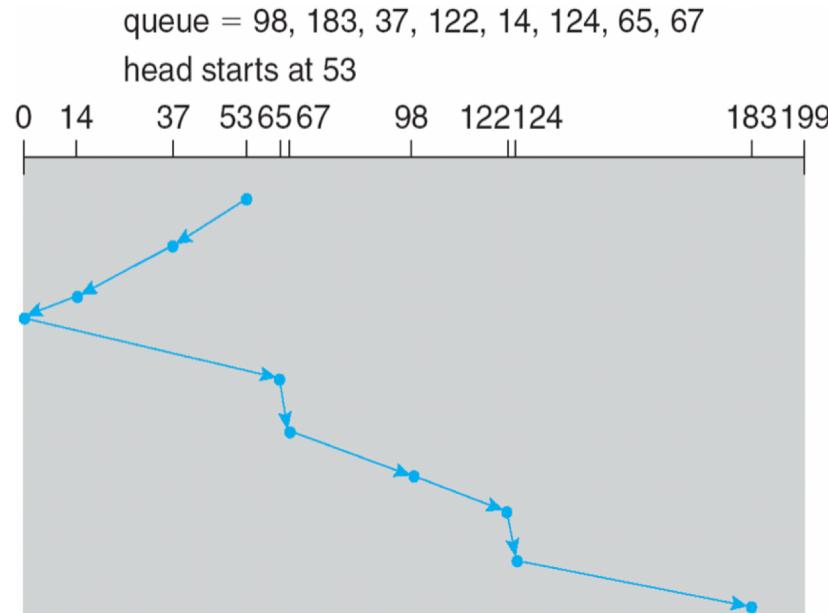
SSTF

- SSTF: shortest seek time first
 - selects the request with minimum seek time from the **current head position**
 - SSTF scheduling is a form of SJF scheduling, **starvation may exist**
 - unlike SJF, SSTF **may not be optimal**
- Total head movement of 236 cylinders



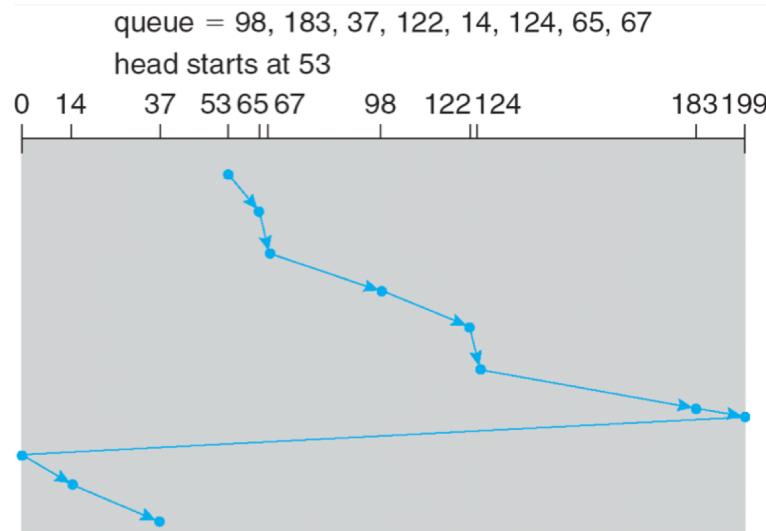
SCAN

- SCAN algorithm sometimes is called the **elevator** algorithm
 - disk arm starts at one **end** of the disk, and moves toward the **other end**
 - service requests during the movement until it gets to the other end
 - then, the head movement is reversed and servicing continues.



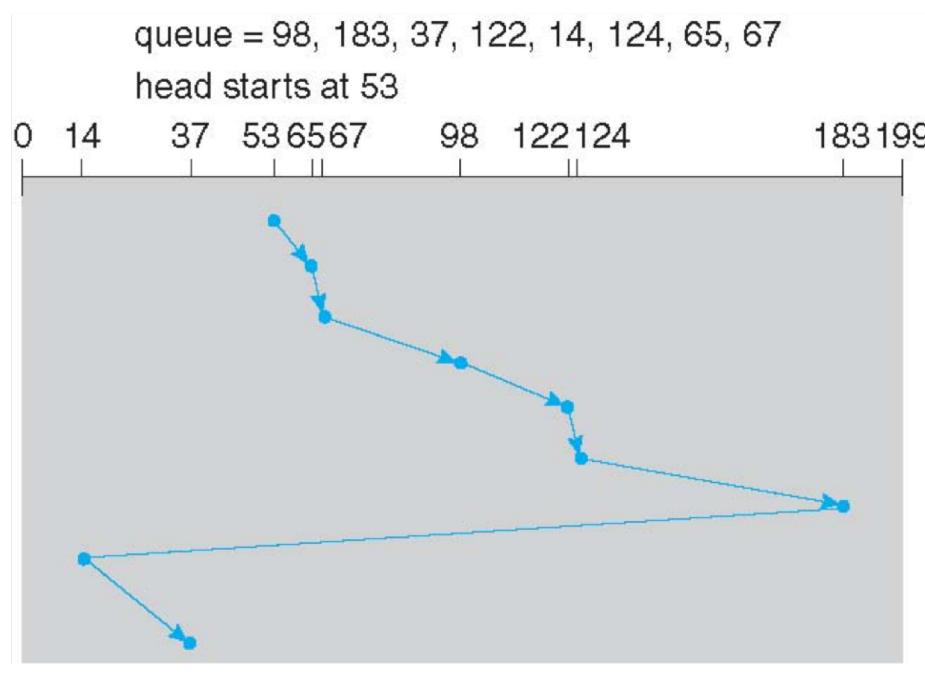
C-SCAN

- Circular-SCAN is designed to provides a more uniform wait time
 - head moves from **one end to the other**, servicing requests while going
 - when the head reaches the end, it immediately returns to the beginning
 - **without servicing any requests on the return trip**
 - it essentially treats the cylinders as a circular list



LOOK/C-LOOK

- SCAN and C-SCAN moves head end to end, even no I/O in between
 - in implementation, head only goes as far as **last request** in each direction
 - **LOOK** is a version of **SCAN**, **C-LOOK** is a version of **C-SCAN**



Disk Management

- **Physical formatting:** divide disk into sectors for controller to read/write
 - Each sector can hold header information, plus data, plus error correction code (ECC)
 - Usually 512 bytes of data but can be selectable
- OS records its own data structures on the disk
 - **partition disk** into groups of cylinders, each treated as a logical disk
 - **logical formatting** partitions to make a file system on it
 - some FS has spare sectors reserved to handle bad blocks
 - FS can further group blocks into clusters to improve performance
 - Disk I/O done in blocks
 - File I/O done in **clusters**
 - initialize the boot sector if the partition contains OS image

Disk Management-Device names

- By convention, IDE drives will be given device names /dev/hda to /dev/hdd.
 - Hard Drive A (/dev/hda) is the first drive (not partition)
 - Hard Drive C (/dev/hdc) is the third drive
- SCSI drives follow a similar pattern; They are represented by 'sd' instead of 'hd'.
 - Partition starts from 1, such as /dev/sda1 or /dev/hda1
 - **The first partition of the second SCSI drive would therefore be /dev/sdb1.**

Disk Management-Partition

- There are at most 4 primary partitions on a disk

drive name	drive controller	drive number	partition type	partition number
/dev/hda1	1	1	primary	1
/dev/hda2	1	1	primary	2
/dev/hda3	1	1	primary	3
/dev/hda4	1	1	swap	NA
/dev/hdb1	1	2	primary	1
/dev/hdb2	1	2	primary	2
/dev/hdb3	1	2	primary	3
/dev/hdb4	1	2	primary	4

- If we need to have more than 4 partitions, we need to use the logical partition - number starts from 5

drive name	drive controller	drive number	partition type	partition number
/dev/hdb1	1	2	primary	1
.....				
/dev/hda5	1	2	logical	2
/dev/hdb6	1	2	logical	3

Disk Management

- **Root partition** contains the OS, other partitions can hold other OSes, other file systems, or be raw
 - **Mounted at boot time**
 - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
 - Is all metadata correct?
 - If not, fix it, try again
 - If yes, add to mount table, allow access
- **Boot block** can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
 - Or a boot management program for multi-os booting

Swap Space Management

- Used for moving entire processes (swapping), or pages (paging), from DRAM to secondary storage when DRAM not large enough for all processes
- Operating system provides swap space management
 - Secondary storage slower than DRAM, so important to optimize performance
 - Can be in **separate partition** or a **file system** (for convenience of adding)

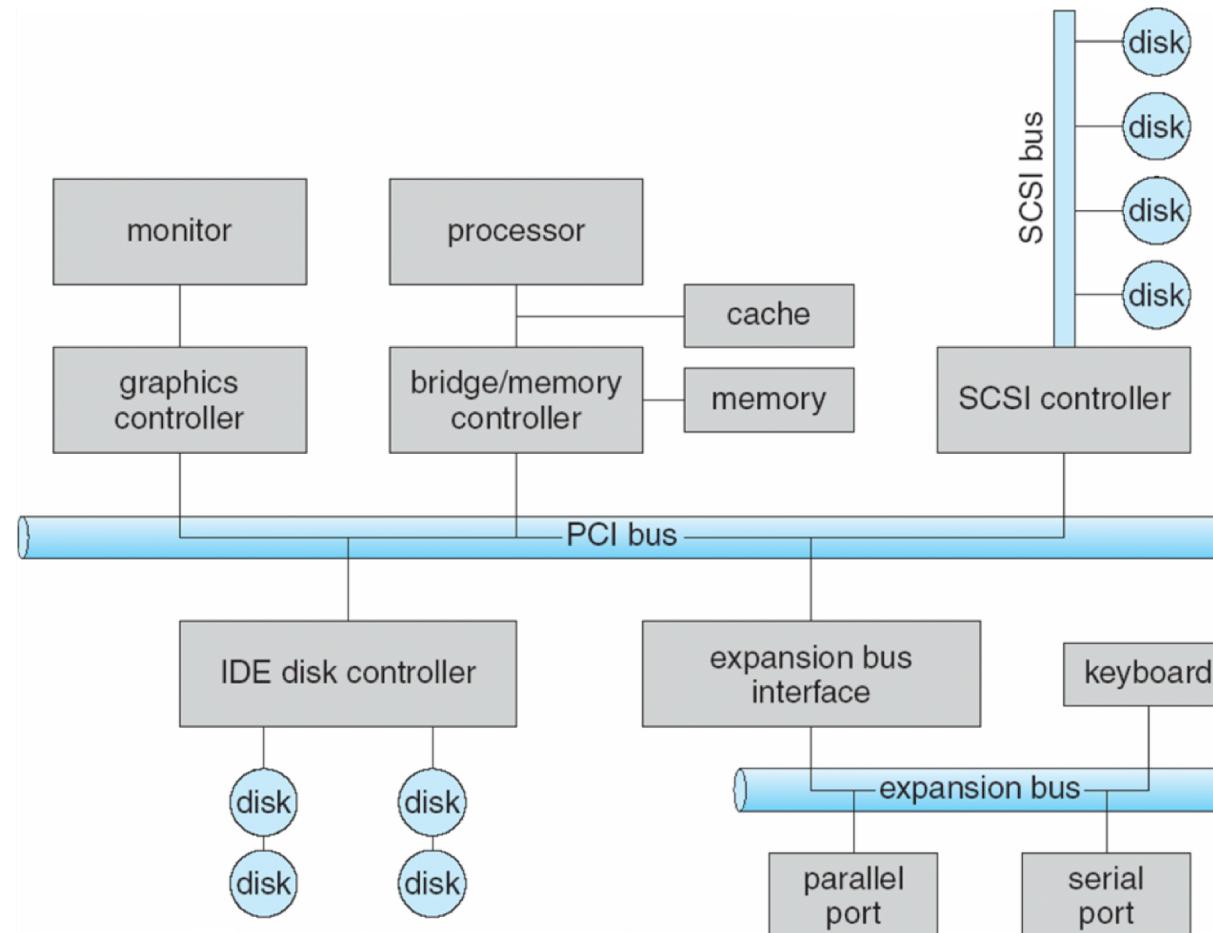
RAID

- RAID – redundant array of inexpensive disks
 - Or Redundant Array of Independent Disks
 - multiple disk drives provides reliability via redundancy
- Increases the mean time to failure

12: I/O Systems



A Typical PC Bus Structure



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
 - in Linux, devices can be accessed as files; low-level access with `ioctl`
- **Device-driver layer** hides differences among I/O controllers from kernel
 - each OS has its own I/O subsystem and device driver frameworks
 - new devices talking already-implemented protocols need no extra work

13&14: File System



File Concept

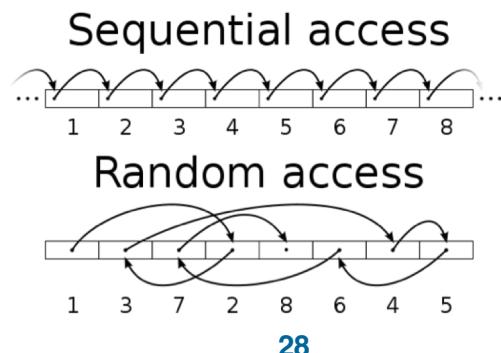
- **File** is a contiguous logical address space for storing information
 - database, audio, video, web pages...
- There are different types of file:
 - data: numeric, character, binary
 - program
 - special one: proc file system - use file-system interface to retrieve system information

File Attributes

- **File 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

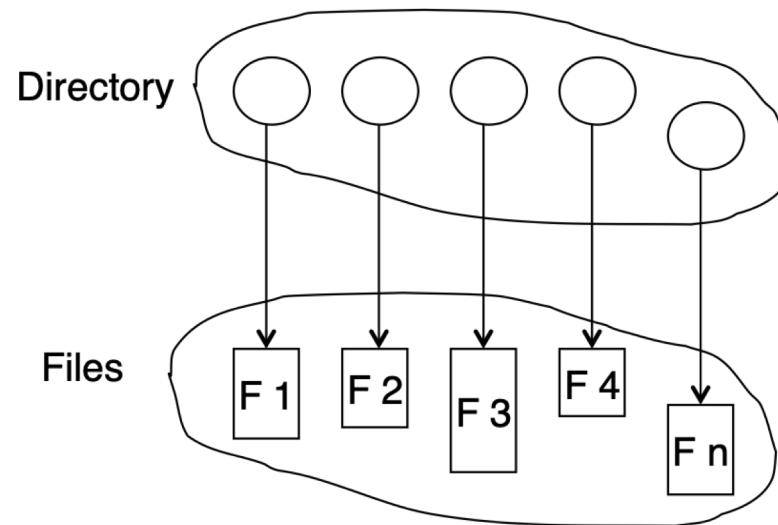
Access Methods

- Sequential access
 - a group of elements is accessed 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
 - sometimes called random access



Directory Structure

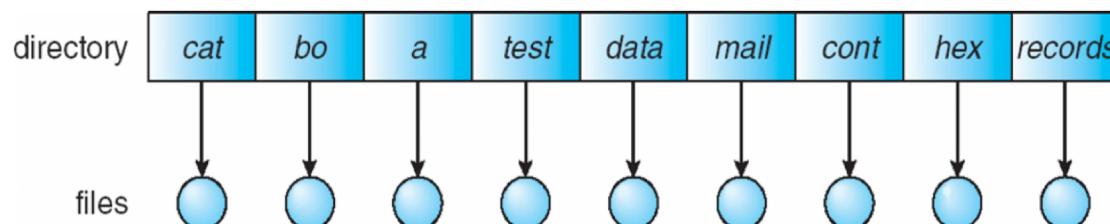
- Directory is a collection of nodes containing information about all files



both the directory structure and the files reside on disk

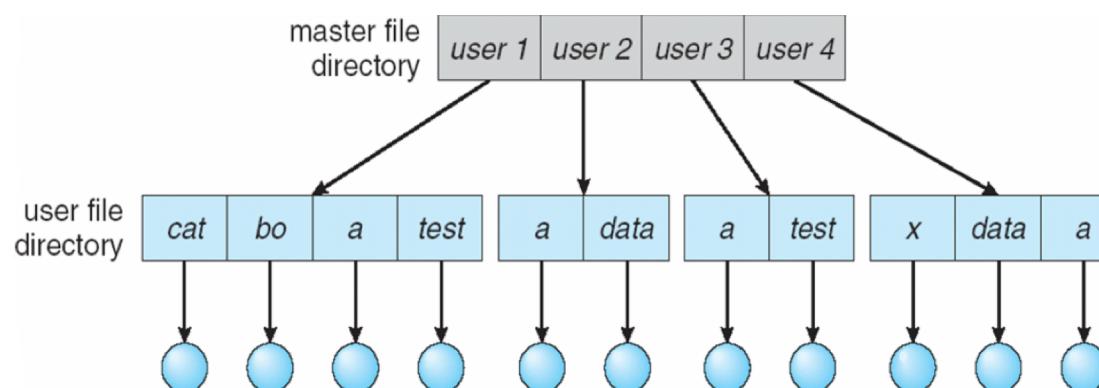
Single-Level Directory

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



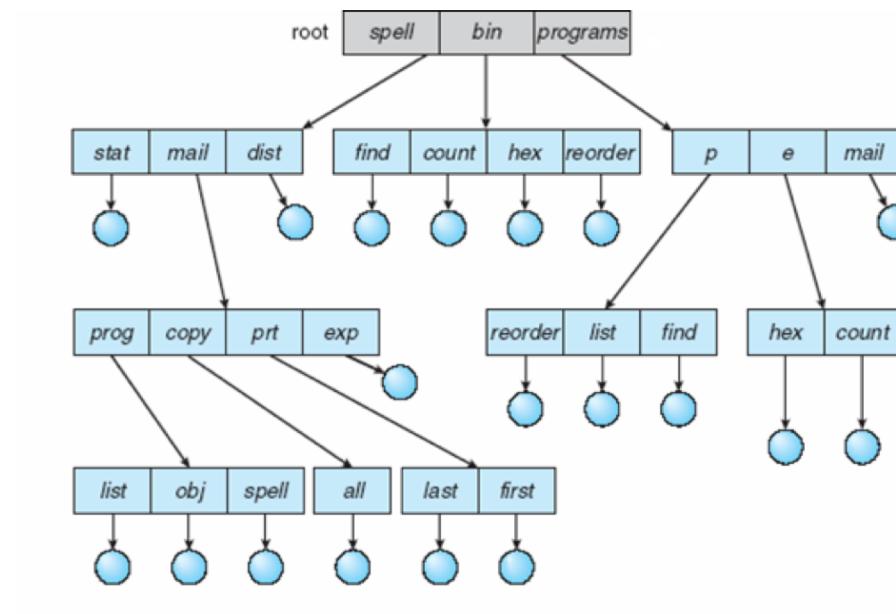
Two-Level Directory

- 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 - solving file name conflicts for different users!



Tree-Structured Directories

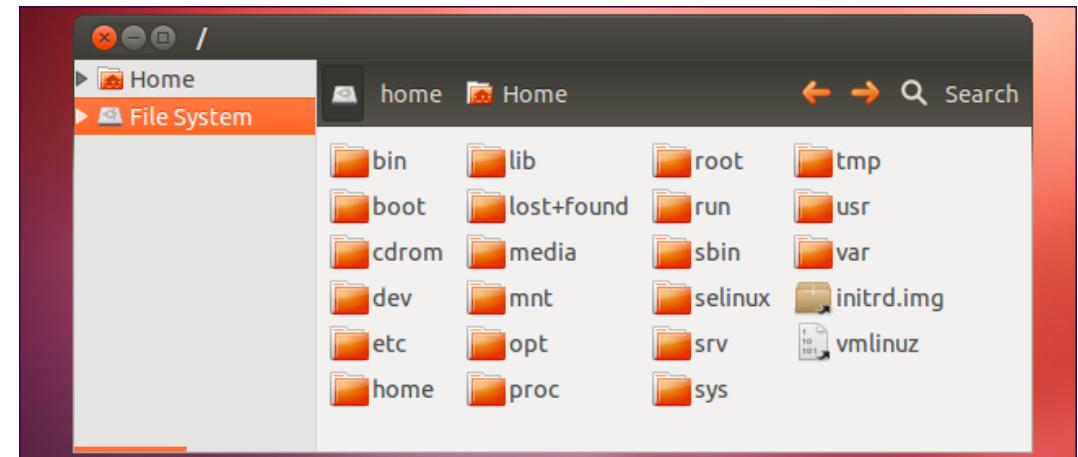
- File can be accessed using **absolute** or **relative** path name
 - absolute path name: /home/alice/..
 - relative path is relative to the **current directory (pwd)**
 - creating a new file, delete a file, or create a sub-directory
 - e.g., if current directory is /mail, a **mkdir count** will create /mail/count

```
echo $(pwd)
```

Quick quiz: what 's the functionality of the set shell command?
->to set or get environment variables

Linux Common Directories

- / - The Root Directory
 - /bin - Essential User Binaries
 - /boot - Static Boot Files
 - /dev - Device Files
 - /etc - Configuration Files
 - /home - Home Folders
 - /lib - Essential Shared Libraries
 - /mnt - Temporary Mount Points
 - /proc - Kernel & Process Files
-
- Demo



<https://www.howtogeek.com/117435/htg-explains-the-linux-directory-structure-explained/#:~:text=%2F%20%E2%80%93%20The%20Root%20Directory,doesn't%20have%20drive%20letters>

Protection

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

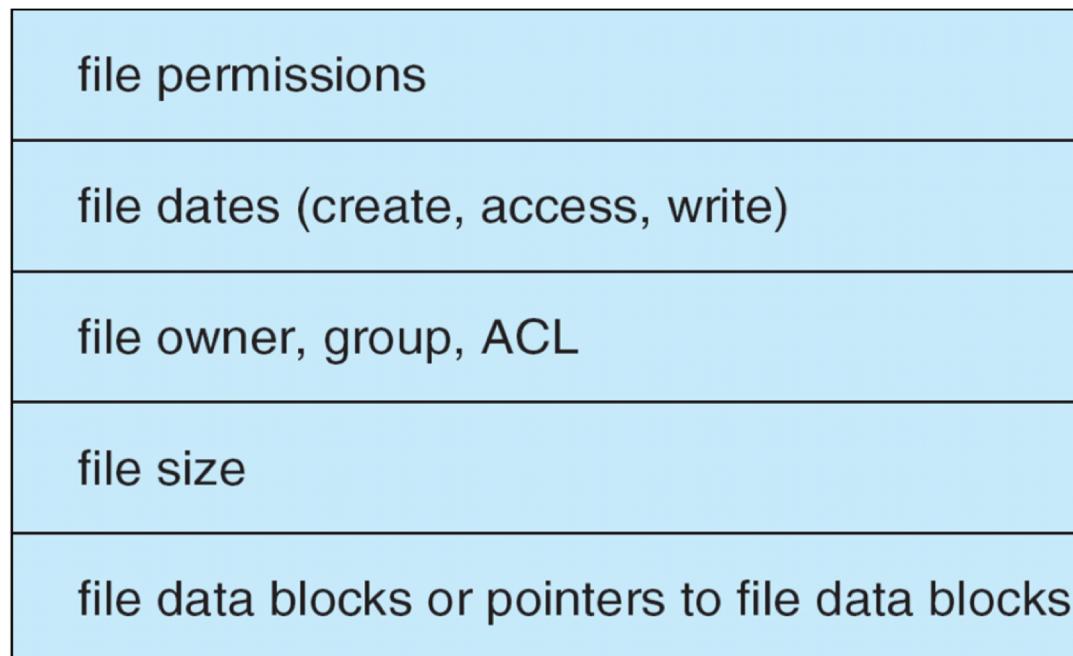
File-System Structure

- File is a logical storage unit for a collection of related information
- There are many file systems; OS may support several **simultaneously**
 - Linux has Ext2/3/4, Reiser FS/4, Btrfs...
 - Windows has FAT, FAT32, NTFS...
 - new ones still arriving - ZFS, GoogleFS, Oracle ASM, FUSE
- File system resides on **secondary storage** (disks)
 - disk driver provides interfaces to read/write disk blocks
 - **fs** provides user/program interface to storage, mapping **logical to physical**
 - file control block - storage structure consisting of information about a file
- File system is usually implemented and organized into **layers**

File System Data Structures

- The file system comprises data structures
- On-disk structures:
 - An optional **boot control block**
 - First block of a volume that stores an OS
 - boot block in UFS, partition boot sector in NTFS
 - A **volume control block**
 - Contains the number of blocks in the volume, block size, free-block count, free-block pointers, free-FCB count, FCB-pointers
 - superblock in UFS, master file table in NTFS
 - A **directory**
 - File names associated with an ID, FCB pointers
 - A **per-file File Control Block (FCB)**
 - In NTFS, the FCB is a row in a relational database
- In-memory structures:
 - A **mount table** with one entry per mounted volume
 - A **directory cache** for fast path translation (performance)
 - A **global open-file table**
 - A **per-process open-file table**
 - Various **buffers** holding disk blocks "in transit" (performance)

A Typical File Control Block

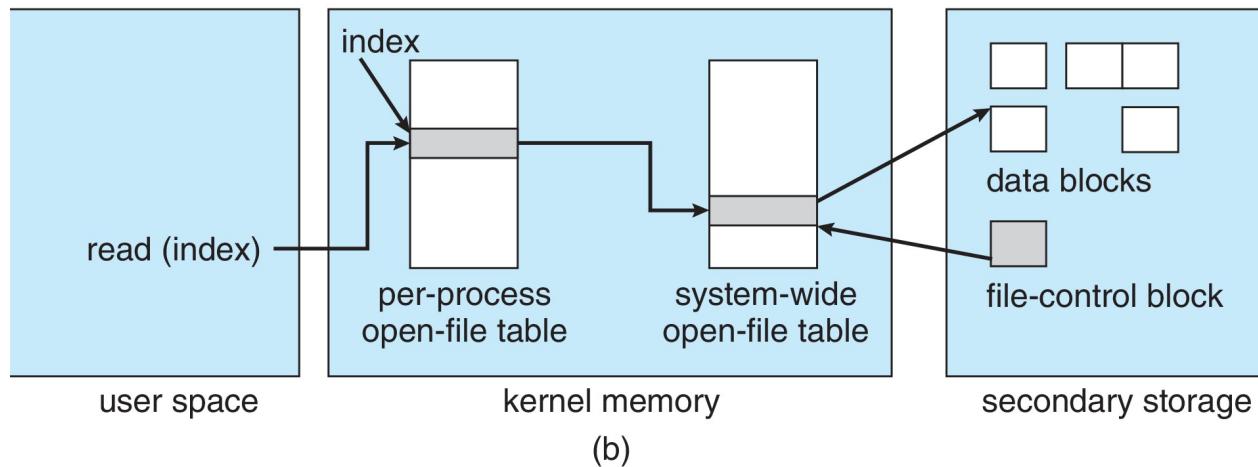
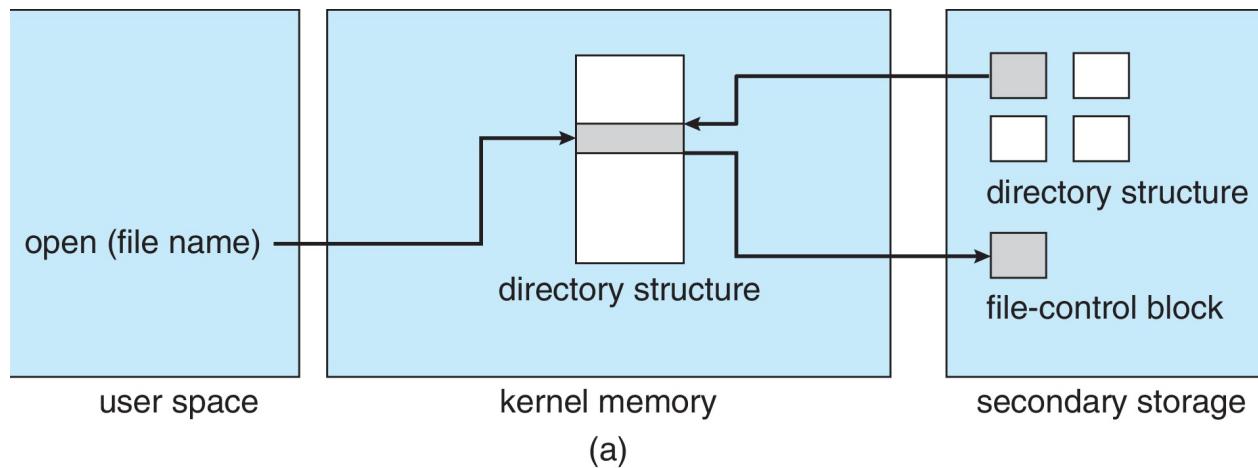


File Creation

- application process requests the creation of a new file
- logical file system allocates a new FCB, i.e., inode structure
- appropriate directory is updated with the new file name and FCB, i.e., inode

Directory

- Unix - directories are treated as files containing special data
- Windows - directories differently from files;
 - they require a separate set of systems calls to create, manipulate, etc



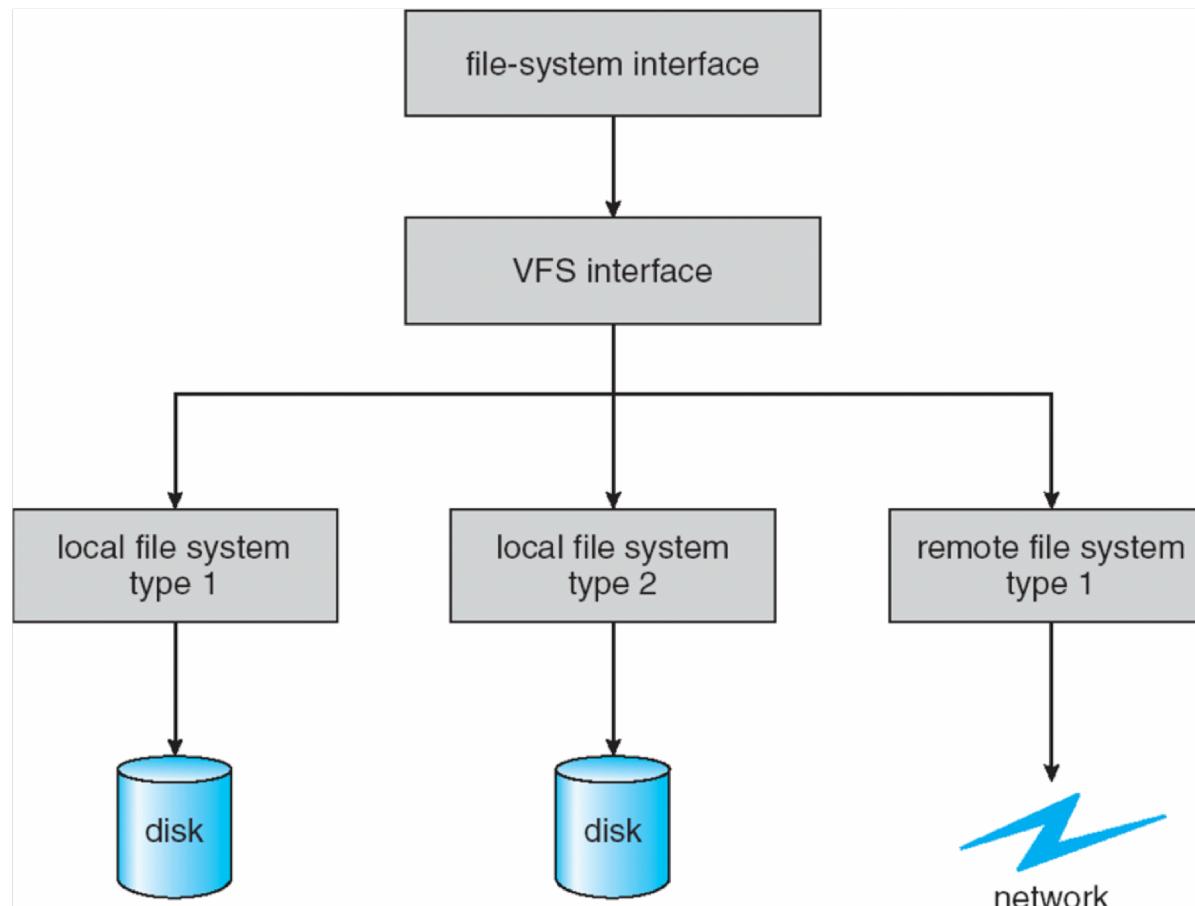
Operations - open()

- search **System-Wide Open-File Table** to see if file is currently in use
 - if it is, create a Per-Process Open-File table entry pointing to the existing System-Wide Open-File Table
 - if it is not, search the directory for the file name; once found, load the **FCB** from disk to memory and place it in the System-Wide Open-File Table
- make an entry in the **Per-Process Open-File Table**, with pointers to the entry in the **System-Wide Open-File Table** and other fields which include a pointer to the current location in the file and the access mode in which the file is open

Virtual File Systems

- **VFS** provides an **object-oriented** way of implementing file systems
 - OS defines a **common interface** for **FS**, all FSes implement them
 - *system call is implemented based on this common interface*
 - it allows the same syscall API to be used for different types of FS
- VFS separates FS generic operations from implementation details
 - implementation can be one of many FS types, or network file system
 - OS can dispatches syscalls to appropriate FS implementation routines

Virtual File System



Virtual File System Example

- Linux defines four **VFS object types**:
 - **superblock**: defines the file system type, size, status, and other metadata
 - **inode**: contains metadata about a file (location, access mode, owners...). -FCB!!
 - **dentry**: associates names to inodes, and the directory layout
 - **file**: actual data of the file
- VFS defines set of operations on the objects that must be implemented
 - the set of operations is saved in a function table

```
struct file_operations {  
    int (*lseek) (struct inode *, struct file *, off_t, int);  
    int (*read) (struct inode *, struct file *, char *, int);  
    int (*write) (struct inode *, struct file *, const char *, int);  
    int (*readdir) (struct inode *, struct file *, void *, filldir_t);  
    int (*select) (struct inode *, struct file *, int, select_table *);  
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);  
    int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);  
    int (*open) (struct inode *, struct file *);  
    void (*release) (struct inode *, struct file *);  
    int (*fsync) (struct inode *, struct file *);  
    int (*fasync) (struct inode *, struct file *, int);  
    int (*check_media_change) (kdev_t dev);  
    int (*revalidate) (kdev_t dev);  
};
```

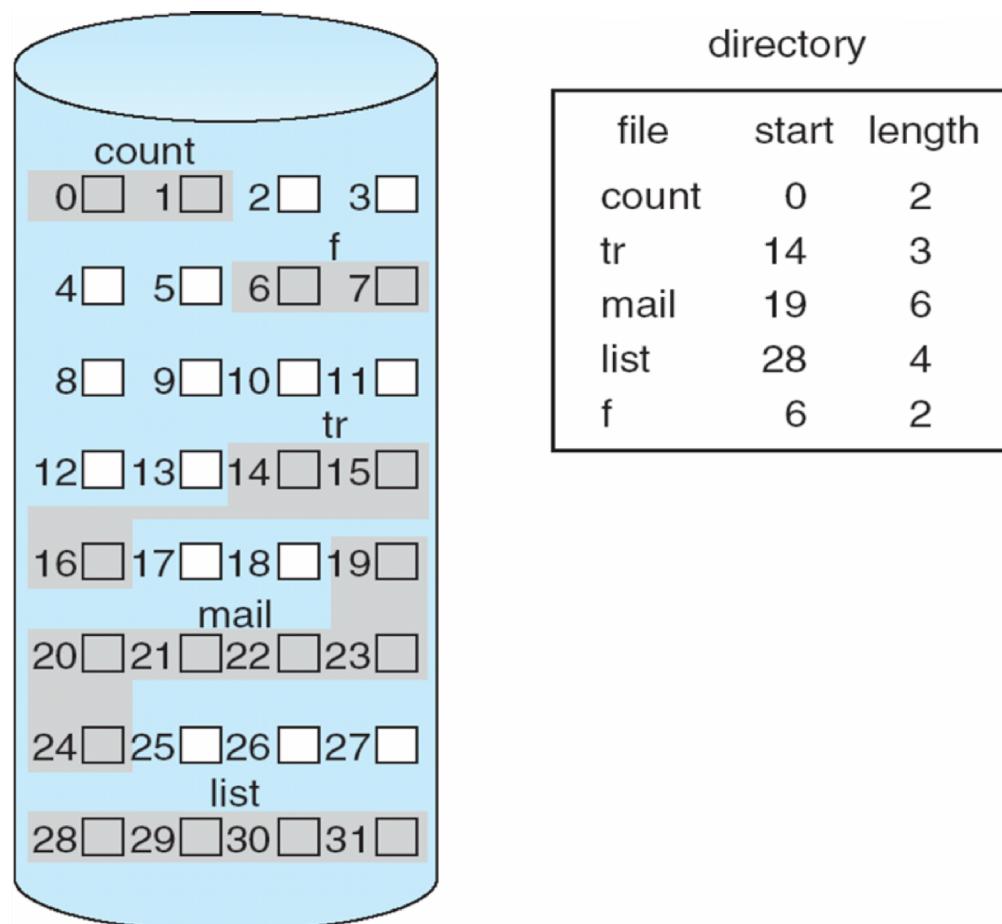
Disk Block Allocation

- Files need to be allocated with disk blocks to store data
 - different allocation strategies have different complexity and performance
- Many allocation strategies:
 - contiguous
 - linked
 - indexed
 - ...

Contiguous Allocation

- Contiguous allocation: each file occupies set of **contiguous blocks**
 - best performance in most cases
 - simple to implement: only starting location and length are required
- Contiguous allocation is not flexible
 - how to *increase/decrease* file size?
 - need to know file size at the file creation?
 - **external fragmentation**
 - how to compact files offline or online to reduce external fragmentation
 - need for **compaction** off-line (downtime) or on-line
 - appropriate for sequential disks like **tape**
- Some file systems use **extent-based contiguous allocation**
 - extent is a set of contiguous blocks
 - a file consists of extents, extents are not necessarily adjacent to each other

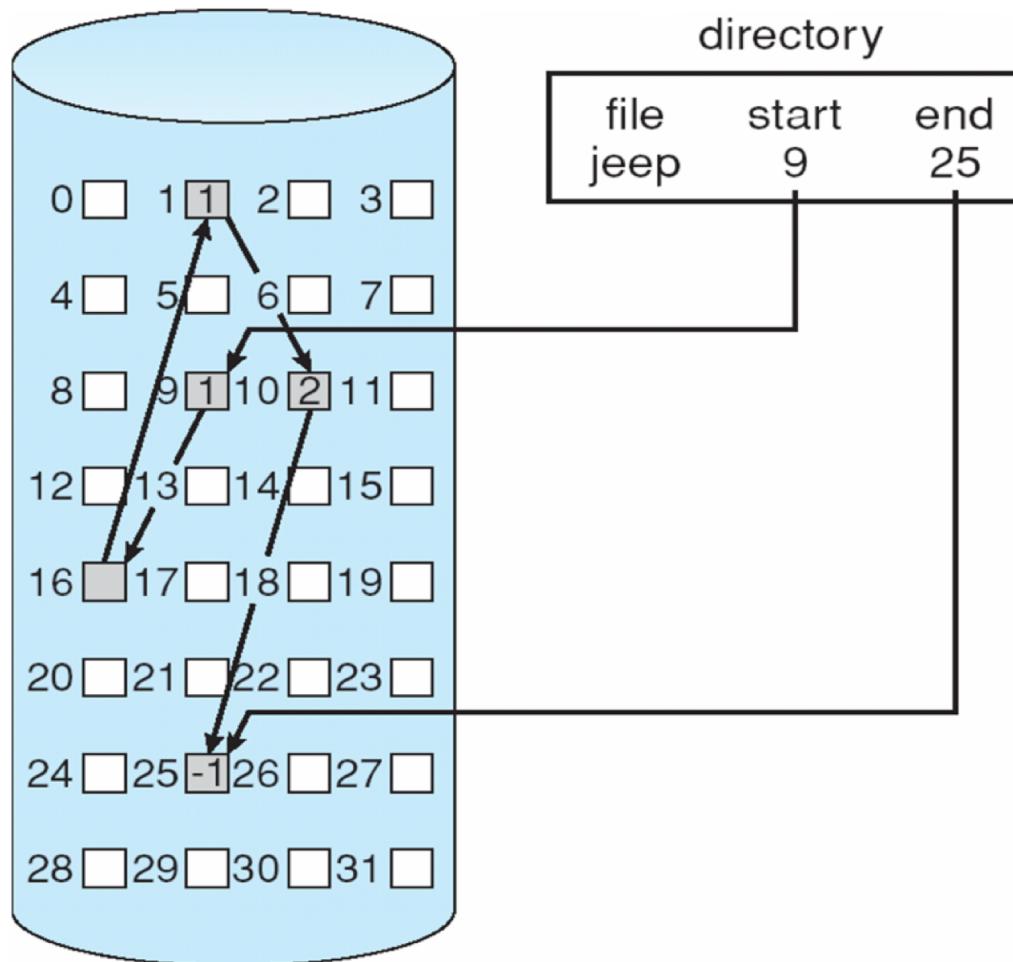
Contiguous Allocation



Linked Allocation

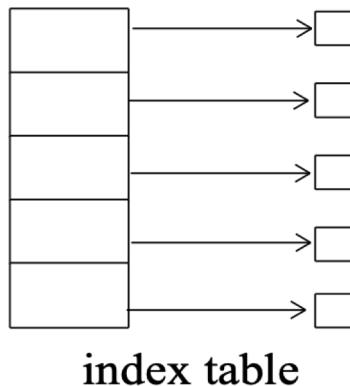
- Linked allocation: each file is a **linked list of disk blocks**
 - each block contains pointer to **next block**, file ends at nil pointer
 - blocks may be scattered anywhere on the disk (no external fragmentation, no compaction)
 - *Disadvantages*
 - *locating a file block can take many I/Os and disk seeks*
 - *Pointer size: 4 of 512 bytes are used for pointer - 0.78% space is wasted*
 - *Reliability: what about the pointer has corrupted!*

Linked Allocation



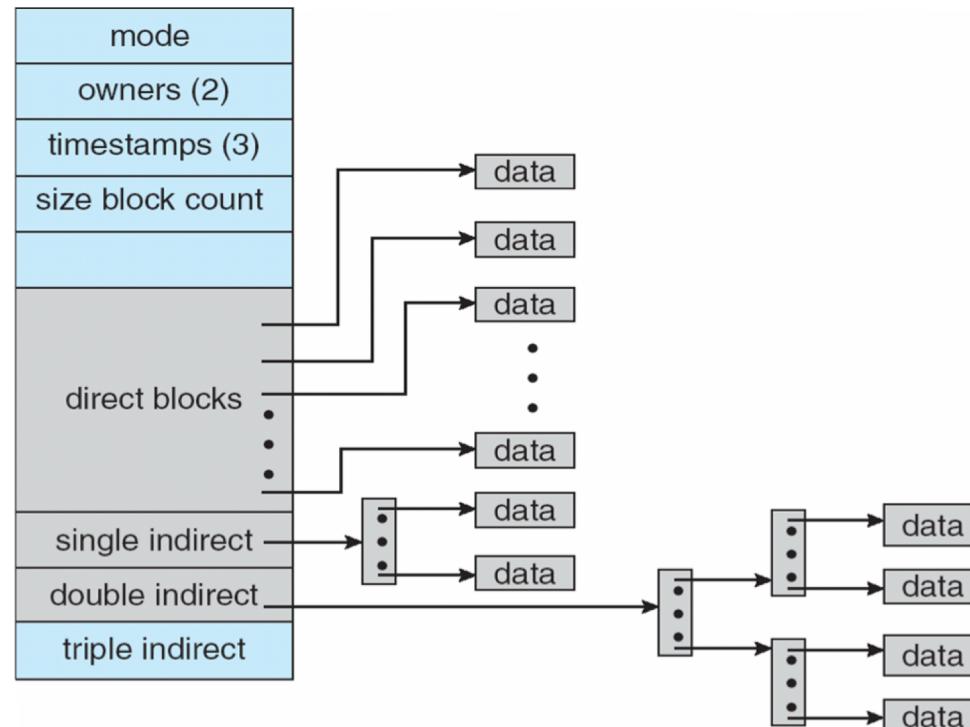
Indexed Allocation

- Indexed allocation: each file has its own **index blocks of pointers to its data blocks**
 - index table provides **random/direct access** to file data blocks
 - no **external fragmentation**, but overhead of index blocks
 - allows **holes** in the file
 - Index block needs space - waste for small files



Indexed Allocation

- Combined scheme
- The UNIX FCB: the **inode** (index node)
 - First 15 pointers are in inode
 - Direct block: first 12 pointers
 - Indirect block: next 3 pointers



Hard link vs soft link

- Link
 - A file may be known by more than one name in one or more directories. Such multiple names are known as links.
 - The two kinds of links are also known as hard links and soft links.
- Hard link
 - a hard link is a **directory entry** that associates with a file
 - The file name “.” in a directory is a hard link to the directory itself
 - The file name “..” is a hard link to the parent directory
- Soft link (a.k.a., symbolic link or symlink)
 - A symbolic link is a **file** containing the path name of another file
 - Soft links, unlike hard links, may point to directories and may cross file-system boundaries

Hard links

```
os@os:~/temp/foo$ rm file
os@os:~/temp/foo$ cat file2
hello
os@os:~/temp/foo$
```

- rm: unlink the file, and check the reference count of the inode

```
os@os:~/temp/foo$ echo hello >file
os@os:~/temp/foo$ stat file
  File: 'file'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650  Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      os)  Gid: ( 1000/      os)
Access: 2018-12-21 00:09:14.763058468 +0800
Modify: 2018-12-21 00:09:14.763058468 +0800
Change: 2018-12-21 00:09:14.763058468 +0800
 Birth: -
os@os:~/temp/foo$ ln file file2
os@os:~/temp/foo$ stat file2
  File: 'file2'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650  Links: 2
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      os)  Gid: ( 1000/      os)
Access: 2018-12-21 00:09:14.763058468 +0800
Modify: 2018-12-21 00:09:14.763058468 +0800
Change: 2018-12-21 00:09:24.247463616 +0800
 Birth: -
os@os:~/temp/foo$ ln file2 file3
os@os:~/temp/foo$ stat file
  File: 'file'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650  Links: 3
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      os)  Gid: ( 1000/      os)
Access: 2018-12-21 00:09:14.763058468 +0800
Modify: 2018-12-21 00:09:14.763058468 +0800
Change: 2018-12-21 00:09:33.975880487 +0800
 Birth: -
```

```
os@os:~/temp/foo$ rm file
os@os:~/temp/foo$ stat file2
  File: 'file2'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650  Links: 2
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      os)  Gid: ( 1000/      os)
Access: 2018-12-21 00:09:14.763058468 +0800
Modify: 2018-12-21 00:09:14.763058468 +0800
Change: 2018-12-21 00:09:40.776172622 +0800
 Birth: -
os@os:~/temp/foo$ rm file2
os@os:~/temp/foo$ stat file3
  File: 'file3'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650  Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      os)  Gid: ( 1000/      os)
Access: 2018-12-21 00:09:14.763058468 +0800
Modify: 2018-12-21 00:09:14.763058468 +0800
Change: 2018-12-21 00:09:47.092444491 +0800
 Birth: -
```

Soft links

- rm the target file, the soft link file is not accessible

```
os@os:~/temp/foo$ echo hello > file
os@os:~/temp/foo$ ln -s file file2
```

```
os@os:~/temp/foo$ ls -l
total 4
-rw-rw-r-- 1 os os 6 Dec 21 00:11 file
lrwxrwxrwx 1 os os 4 Dec 21 00:12 file2 -> file
os@os:~/temp/foo$ stat file
  File: 'file'
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 1328650      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/        os)  Gid: ( 1000/        os)
Access: 2018-12-21 00:11:54.636605978 +0800
Modify: 2018-12-21 00:11:54.636605978 +0800
Change: 2018-12-21 00:11:54.636605978 +0800
 Birth: -
os@os:~/temp/foo$ stat file2
  File: 'file2' -> 'file'
  Size: 4          Blocks: 0          IO Block: 4096   symbolic link
Device: 801h/2049d  Inode: 1328653      Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 1000/        os)  Gid: ( 1000/        os)
Access: 2018-12-21 00:12:02.300152148 +0800
Modify: 2018-12-21 00:12:00.960229971 +0800
Change: 2018-12-21 00:12:00.960229971 +0800
 Birth: -
```

```
os@os:~/temp/foo$ rm file
os@os:~/temp/foo$ cat file2
cat: file2: No such file or directory
os@os:~/temp/foo$ ls -l
total 0
lrwxrwxrwx 1 os os 4 Dec 21 00:12 file2 -> file
os@os:~/temp/foo$
```

Takeaway

- The whole slides