Disk Storage and File Systems

CS 256/456

Dept. of Computer Science, University of Rochester

Characteristics of I/O Devices

- Data transfer mode – block vs. character
- Access method – sequential vs. random
- Transfer schedule – synchronous vs. asynchronous
- Sharing mode – dedicated vs. sharable
- Device speed – latency, seek time, transfer rate, occupancy/delay between operations
- I/O direction – R, W, R/W

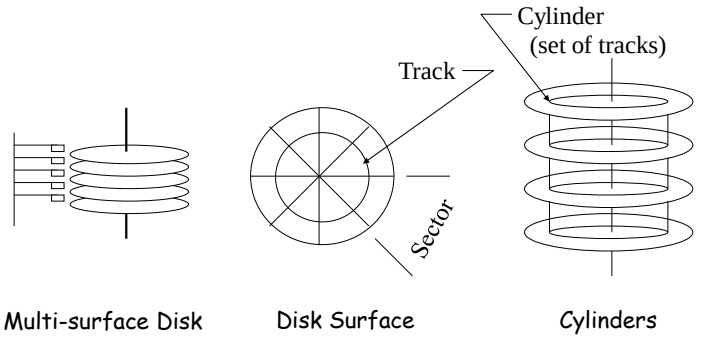
Recap: Disk Storage

- Disk drive
  - mechanical parts (cylinders, tracks, sectors) and how they move to access disk data
  - electronic part (disk controller main) exposes a one-dimensionally addressable set of blocks
  - large seek/rotation time

Disk Management

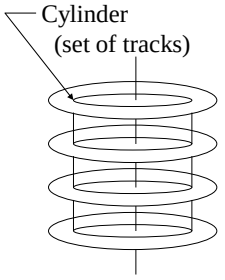
- Formatting
  - Header: sector number etc.
  - Footer/tail: ECC codes
  - Gap
  - Initialize mapping from logical block number to defect-free sectors
- Logical disk partitioning
  - One or more groups of cylinders
  - Sector 0: master boot record loaded by BIOS firmware, which contains partition information
  - Boot record points to boot partition

Disk Drive – Mechanical Parts



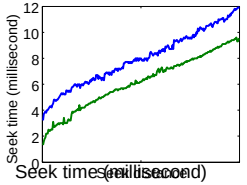
Disk Structure

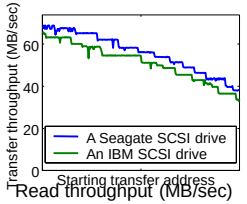
- Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
  - Sector 0 is the first sector of the first track on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.



Disk Performance Characteristics

- A disk operation has three major components
  - *Seek* – moving the heads to the cylinder containing the desired sector
  - *Rotation* – rotating the desired sector to the disk head
  - *Transfer* – sequentially moving data to or from disk





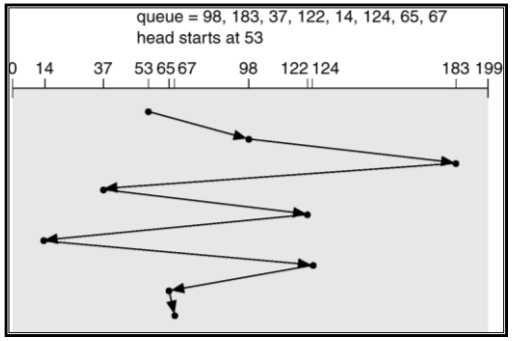
https://en.wikipedia.org/wiki/Hard_disk_drive

Improvement of HDD characteristics over time			
Parameter	Started with (1957)	Developed to (2017)	Improvement
Capacity (formatted)	3.75 megabytes[13]	14 terabytes[14]	3.73-million-to-one[15]
Physical volume	68 cubic feet(1.9 m^3)[16]	2.1 cubic inches(34 cm^3)[16][d]	56,000-to-one[17]
Weight	2,000 pounds(910 kg)[6]	2.2 ounces (62 g)[16]	15,000-to-one[18]
Average access time	approx. 600 milliseconds[6]	2.5 ms to 10 ms; RW RAM dependent	about 200-to-one[19]
Price	US$9,200 per megabyte (1961)[20]	US$0.032 per gigabyte by 2015[21]	300-million-to-one[22]
Data density	2,000 bits per square inch[23]	1.3 terabits per square inch in 2015[24]	650-million-to-one[25]
Average lifespan	c. 2000 hrs MTBF[*citation needed*]	c. 2,500,000 hrs (~285 years) MTBF[26]	1250-to-one[27]

Disk Scheduling

- Disk scheduling – choose from outstanding disk requests when the disk is ready for a new request
  - can be done in both disk controller and the operating system
  - Disk scheduling non-preemptible
- Goals of disk scheduling
  - overall efficiency – small resource consumption for completing disk I/O workload
  - fairness – prevent starvation

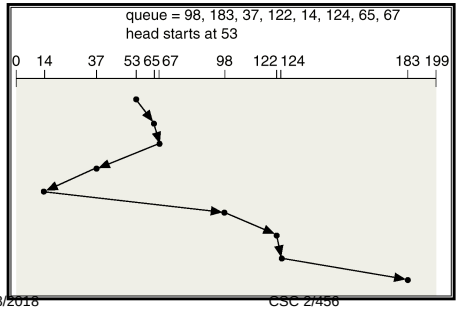
FCFS (First-Come-First-Serve)



- Illustration shows the total head movement is 640.
- Starvation?

SSTF (Shortest-Seek-Time-First)

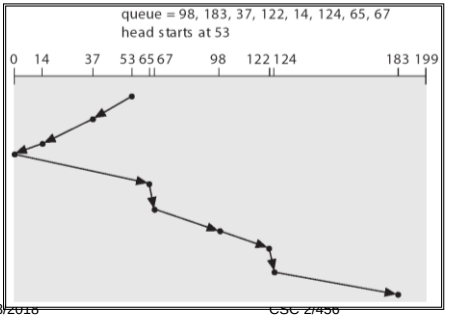
- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling.
- Illustration shows the total head movement is 236.



Starvation?

SCAN

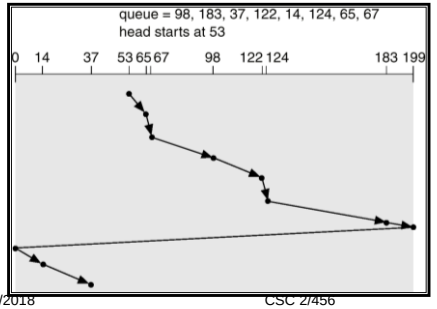
- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- Illustration shows the total head movement is 208.



Starvation?

C-SCAN (Circular-SCAN)

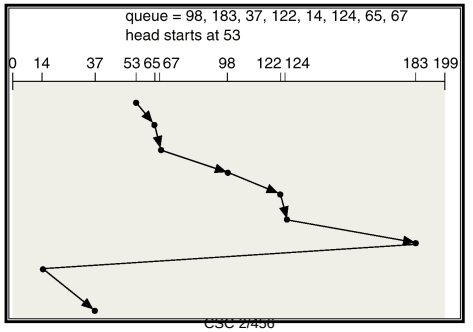
- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.



Starvation?

C-LOOK

- Variation of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

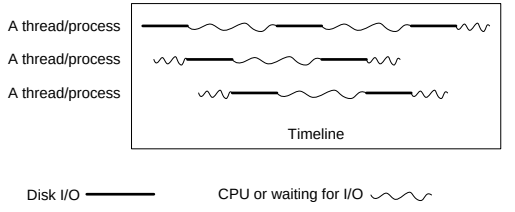


Deadline Scheduling in Linux

- A regular elevator-style scheduler similar to C-LOOK
- Additionally, all I/O requests are put into a FIFO queue with an expiration time (e.g., 500ms)
- When the head request in the FIFO queue expires, it will be executed next (even if it is not next in line according to C-LOOK).
- A mix of performance and fairness.

Concurrent I/O

- Consider two request handlers in a Web server
  - each accesses a different stream of sequential data (a file) on disk;
  - each reads a chunk (the buffer size) at a time; does a little CPU processing; and reads the next chunk
- What happens?

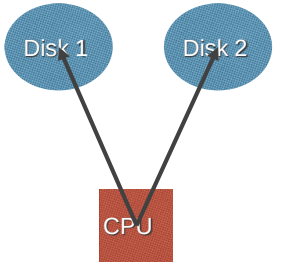


How to Deal with It?

- Aggressive prefetching
- Anticipatory scheduling [Iyer & Druschel, SOSP 2001]
  - at the completion of an I/O request, the disk scheduler will wait a bit (despite the fact that there is other work to do), in anticipation that a new request with strong locality will be issued; schedule another request if no such new request appears before timeout
  - included in Linux 2.6

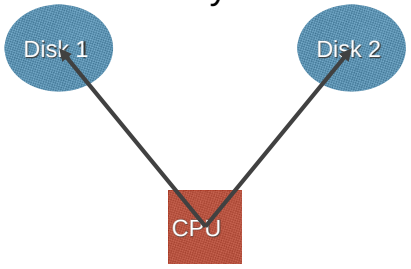
Two Disks: Disk Striping

- Blocks divided into subblocks
- Subblocks stored on different disks



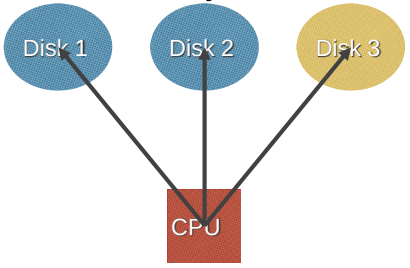
Two Disks: Mirroring

- Make a copy of each block on each disk
- Provides redundancy



Multiple Disks: Parity Block

- Have one disk contain parity bits of blocks on other devices
- Provides redundancy without full copy



Exploiting Concurrency

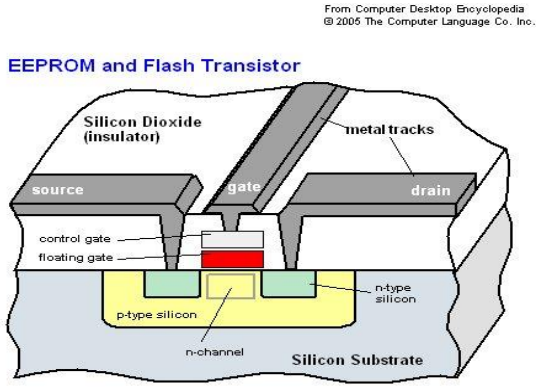
- RAID: Redundant Arrays of Independent Disks
  - RAID 0: data striping at block level, no redundancy
  - RAID 1: mirrored disks (100% overhead)
  - RAID 2: bit-level striping with parity bits, synchronized writes
  - RAID 3: data striping at the bit level with parity disk, synchronized writes
  - RAID 4: data striping at block level with parity disk
  - RAID 5: scattered parity
  - RAID 6: handles multiple disk failures

Solid State Drives

- No mechanical component (moving parts)
- Lower energy requirements
- Speed
  - Reads and writes in the order of 10s of microseconds (reading faster than writing)
  - Erase on the order of a millisecond
- Finite number of erase and write cycles, requiring what is called "wear leveling"

Flash Memory (Based on Charge)

- Based on floating-gate transistor



File Systems

- A File system is the OS abstraction for storage resources
  - File is a logical storage unit in the OS abstract interface for storage resources
    - Extension of address space (temporary files)
    - Non-volatile storage that survives the execution of an individual program (persistent files)
  - Directory is a logical "container" for a group of files

Operations Supported

- Create – associate a name with a file
- Delete – remove the file
- Rename – associate a new name with a file
- Open – create cached context that is associated implicitly with future reads and writes
- Write – store data in a file
- Read – access the data associated with a file
- Close – discard cached context
- Seek – random access to any record or byte
- Map – place in address space for convenience (memory-based loads and stores), speed; disadvantages: lengths that are not multiples of the page size, consistency with open/read/write interface

File System Issues

- File naming and other attributes:
  - name, size, access time, sharing/protection, location
- Intra-file structure
  - None - sequence of words, bytes
  - Complex Structures
    - records/formatted document/executable
- File system organization: efficiency of disk access
- Concurrent access: allow multiple processes to read/write
- Reliability: integrity in the presence of failures
- Protection: sharing/protection attributes and access control lists (ACLs)

File Naming

- Fixed vs. variable length
  - Fixed: 8-255 characters
  - Variable: length:value encoding
- File extensions – system supported vs. convention

Naming Files Using Directory Structures

- Directory: maps names to files; directories may themselves be files
  - Single level (flat): no two files may have the same name
  - Two level: per-user single-level directory
  - Hierarchical: generalization of two level; each file system is assigned the root of a tree
  - Acyclic (or cyclic) graph: allow sharing of files across directories; hard versus soft (symbolic) links

Shared Files: Links

- File appears simultaneously in different directories
- File system is now a directed acyclic graph (DAG)
- Hard link – directory points to file inode, which maintains a count of pointers
- Soft link – new file type, containing the path of the file to which it is linked, along with permissions (symbolic linking) – no pointer to inode

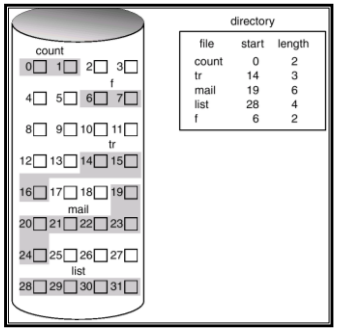
File Types

- Control operations allowed on files
- Use file name extensions to indicate type (in Unix, this is just a convention)
- Structured vs. unstructured data
  - None - sequence of words, bytes
  - Complex Structures
    - records/formatted document/executable
- Sequential, random, or key-based (indexed) access

File Space Organization

- Disk basic allocation unit is a sector (e.g., 512 bytes)
- File system may choose to use a larger block size (e.g., 4KB)
- File allocation methods
  - How disk blocks are allocated for files
    - Contiguous allocation
    - Linked allocation
    - Indexed allocation
  - Metrics:
    - Access speed (sequential & random)
    - Space utilization

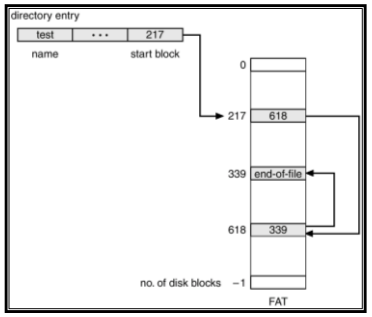
Contiguous File Allocation



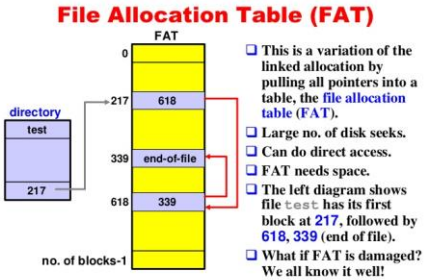
- Each file occupies a set of contiguous blocks on the disk
- Advantage:
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Fast sequential; also quite fast random access
- Disadvantage:
  - External fragmentation
  - Inflexible when appending to a file

Linked File Allocation

- Each file is a linked list of disk blocks
  - each block contains a next pointer
  - directory only needs to store the pointer to the first block
  - blocks may be scattered anywhere on the disk
- Advantage
  - Space efficient
  - Flexible in appending
- Disadvantage:
  - Poor access speed (sequential & random)



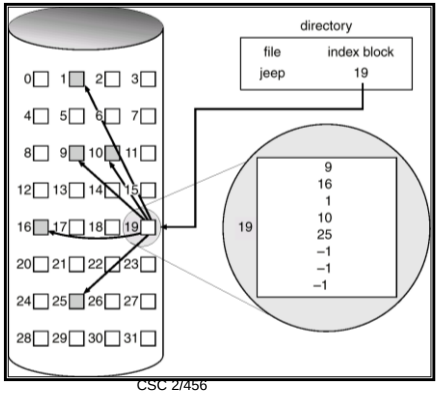
File Allocation Table



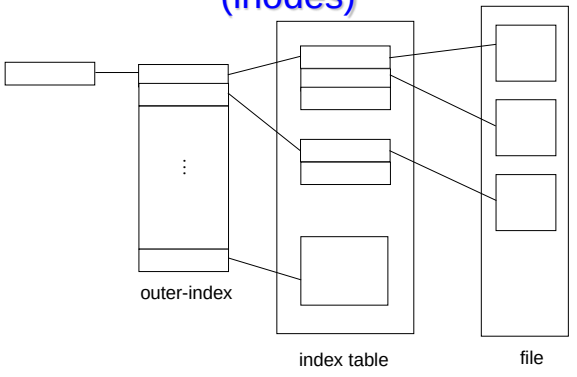
- This is a variation of the linked allocation by pulling all pointers into a table, the file allocation table (FAT).
- Large no. of disk seeks.
- Can do direct access.
- FAT needs space.
- The left diagram shows file `test` has its first block at 217, followed by 618, 339 (end of file).
- What if FAT is damaged? We all know it well!

Indexed File Allocation

- Brings all pointers together into the *index block*.



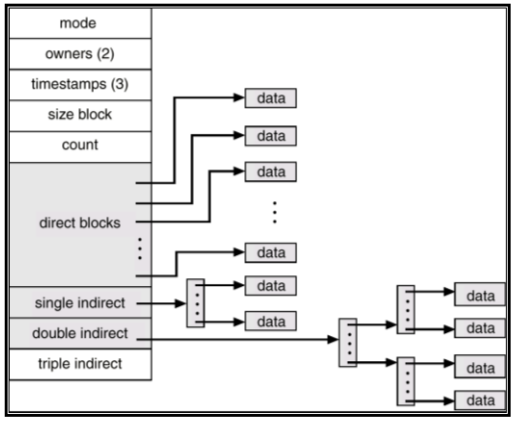
Multi-level Indexed File Allocation (inodes)



Indexed Allocation (pros and cons)

- Space efficiency
  - no external fragmentation
  - overhead of index blocks
- Access speed
  - random access
  - sequential access

UNIX (4K bytes per block)



Directory on the Disk

- Directory is a container of files
- For space management, similar to files
- But for directory, file system does care about its content
  - Linear list of file names and attributes (including pointers to the data blocks)
    - time-consuming to search an item
  - Hash Table – using a link list to chain all files hashed to the same value
    - Pro: decreases directory search time
    - Con: increased complexity, a little waste of space
    - how much benefit does it really provide?

Where to put file attributes?

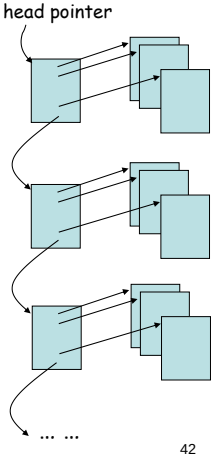
- File control block – data structure including all attributes for a file
- Where to put the file control block?
  - In the directory data structure
    - Hard to share files through links
  - In the system-level dedicated data structure
    - inode

Device Space Management

- Block size: internal fragmentation/wasted space vs. allocation efficiency and access latency
- Free space management
- Reducing disk arm motion

Free-Space Management

- Free-space management for memory
- Bit map and linked free block list
- Space overhead: bit vs. word
- Efficiency
  - getting the address of one free block
  - getting the addresses of a number of free blocks
- Alternative: Grouping/clustering



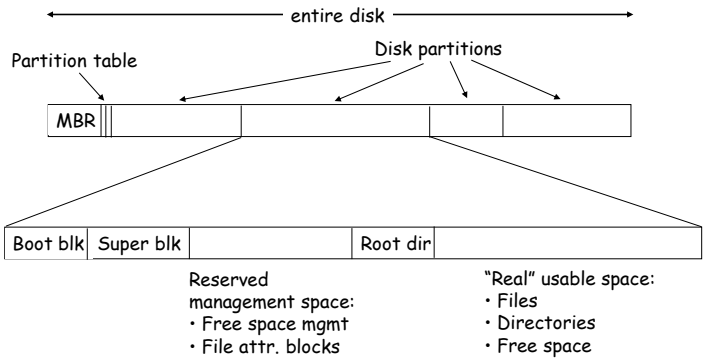
File System Issues

- File naming and other attributes:
  - name, size, access time, sharing/protection, location
- Intra-file structure
  - None - sequence of words, bytes
  - Complex Structures
    - records/formatted document/executable
- File system organization: efficiency of disk access
- Concurrent access: allow multiple processes to read/write
- Reliability: integrity in the presence of failures
- Protection: sharing/protection attributes and access control lists (ACLs)

File Sharing and Protection

- Sharing of files on multi-user systems is desirable
- Sharing must be accompanied by a *protection* scheme
  - In general, a protection scheme specifies whether any specific user can access any specific file
    - Access control lists (ACL)
    - User, group, other permissions

File System Layout



In-Memory Structures

- Used for file system management and performance improvement via caching
  - Mount table (info on each mounted volume)
  - Directory-structure cache
  - System-wide open file table
    - Copy of FCB (file control block) of each open file
  - Per-process open file table
    - Pointer to entry in system-wide table along with process-specific information
- Open system call returns a pointer to the appropriate entry in per-process file table (file descriptor or file handle)

Swap Space Management

- Part of file system?
  - Requires navigating directory structure
  - Disk allocation data structures
- Separate disk partition
  - No file system or directory structure
  - Optimize for speed rather than storage efficiency
  - When is swap space created?

Delayed Writes and Data Loss at Machine Crash

- Writes are commonly delayed for better performance
  - data to be written is cached
- A sudden machine crash may result in a loss of data
  - a completed write does not mean the data is safely stored on storage
- fsync() – flush all delayed writes to disk
  - fsync() may not even be totally safe with delayed writes on disk controller buffer cache

Consistency: Weaker Form of Reliability

- File system operations are not atomic; a sudden machine crash may leave the file system in an inconsistent state
- (In-)Consistency
  - Missing blocks
  - Duplicate free blocks
  - Duplicate data blocks
- Consistency checking and fix (fsck, scandisk)
  - use redundant data on disk to recover consistency
  - E.g., free block cannot be on the free list and in a file

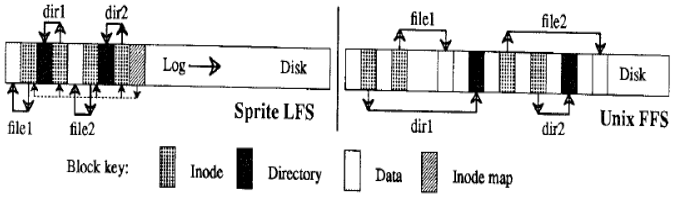
Journaling

- Journaling file system:
  - maintain a dedicated journal that logs all operations
  - the logging happens before the real operation
  - each logging is made to be atomic
  - after the completion of an operation, its entry is removed from the journal
  - at the recovery time, only journal entries need to be examined ⇒ fast recovery
  - similar to transactions in database systems

Log-Structured File Systems

- With CPUs faster, memory larger
  - buffer caches can also be larger
  - most of read requests can come from the memory cache
  - thus, most disk accesses will be writes
  - poor disk performance when most writes are small
- LFS Strategy [Rosenblum&Ousterhout SOSP1991]
  - structures entire disk as a log
  - always write to the end of the disk log
  - when updates are needed, simply add new copies with updated content; old copies of the blocks are still in the earlier portion of the log
  - periodically purge out useless blocks

Log-Structured vs. Unix



Journaling

- LFS is a dynamic journal
- Physical journal (ext3)
- Logical journal (NTFS)
- Snapshotting (ZFS)

Linux and the Extended File Systems

ext	Volume Size	File Size	Filename length
ext2	2-32 TiB	16 GiB-2TiB	255 bytes
ext3	4-32 TiB	16 GiB-2TiB	255 bytes
ext4	16TiB-1EiB	16 TiB	255 bytes

Ext 3 and 4: Journaling

"New" Motivations

- Fast recovery
  - Compared to fsck/scandisk
- Persistency
  - Availability

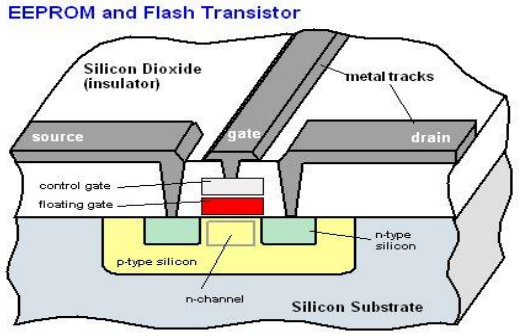
Solid State Drives

- No mechanical component (moving parts)
- Lower energy requirements
- Speed
  - Reads and writes in the order of 10s of microseconds (reading faster than writing)
  - Erase on the order of a millisecond
- Finite number of erase and write cycles, requiring what is called "wear leveling"

Flash Memory (Based on Charge)

- Based on floating-gate transistor

From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.

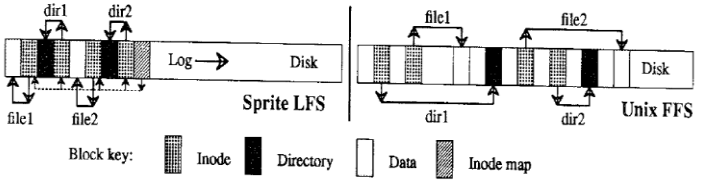


Solid State Drives: File System Implications?

- No need to "cluster" data to reduce seek time
- Need to avoid writes to the same block
- File system cache less useful due to lower speed mismatch
- Log-structured file system for SSD
  - Provides wear leveling

Flash File Systems for Solid State Drives

- E.g., JFFS, YAFFS, LogFS
- Log-structure file systems



Read and Write Bandwidths

	4MB seq. bw (Queue depth)	4KB rand bw (Queue depth)	4K bw / 4MB bw	64b rand bw
HDD	213MB/s (8)	2.3MB/s (32)	1.07%	0.8KB/s
SSD	378MB/s (4)	260MB/s (16)	70%	0.5MB/s
NVMe	2524MB/s (4)	892MB/s (32)	35%	3.5MB/s

Read bandwidth

	4MB seq. bw (Queue depth preprocess)	4KB rand bw (Queue depth)	4K bw / 4MB bw	64b rand bw
HDD	211MB/s (8)	1.5MB/s (32)	0.7%	0.5KB/s
SSD	360MB/s (4)	171MB/s (16)	0.47%	0.3MB/s
NVMe	1790MB/s (4)	1100MB/s (32)	61%	4MB/s

Write bandwidth

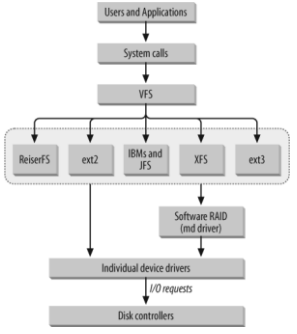
Example File Systems

- MS-DOS/Windows – file allocation table (FAT), NTFS
- Berkeley-FFS
- Linux – VFS, ext2fs, ext3, ext4
- NFS
- JFFS (Journaling Flash File System)
- …

Miscellaneous Issues

- Directory structure – acyclic versus cyclic graphs
- File system mounting and the virtual file system
- File protection
  - Types of access –
    - File - r,w,x,append, delete, list
    - Directory – search for/rename/create/delete file, list directory, traverse file system
  - Access control
- Efficiency and performance
  - Positioning of blocks
  - Buffer and page caches

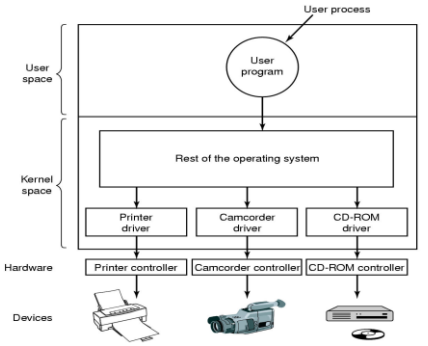
Virtual File System



Buffering, Caching, Spooling

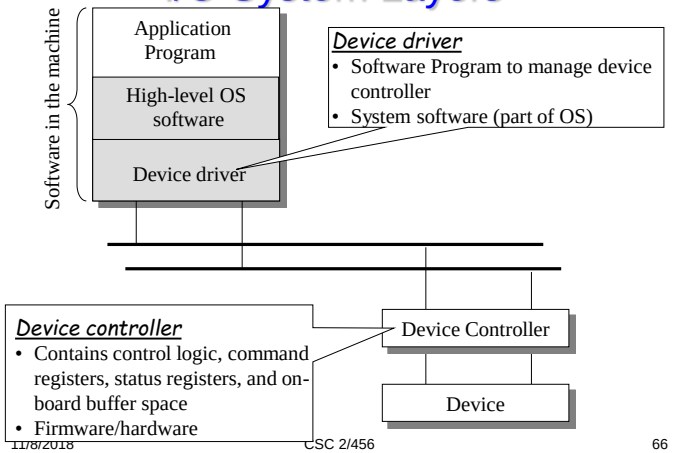
- Techniques for handling speed mismatch
  - Buffering: e.g., wait for buffer to fill from slow device before writing to disk
  - Caching: copy in memory for fast access relative to a slower device (e.g., disk)
  - Spooling: buffer to hold output for a device such as a printer that cannot accept multiplex/handle more than one request at a time

I/O Software Layers



- Device-dependent OS I/O software; directly interacts with controller hardware
- Interface to upper-layer OS code is standardized

I/O System Layers



Device Driver Reliability

- Device driver is the device-specific part of the kernel-space I/O software; It also includes interrupt handlers
- Device drivers must run in kernel mode
  ⇒ The crash of a device driver typically brings down the whole system
- Device drivers are probably the buggiest part of the OS
- How to make the system more reliable by isolating the faults of device drivers?
  - Run most of the device driver code at user level
  - Restrict and limit device driver operations in the kernel

High-level I/O Software

- Device independence
  - reuse software as much as possible across different types of devices
- Buffering
  - data coming off a device is stored in an intermediate buffer
  - purpose: access speed/granularity matching with I/O devices
- caching
- speculative I/O

File System Caching

- File content is cached in memory buffer for later reuse
  - what is the basic unit of such caching?
    - Disk blocks vs. clusters vs. pages
- Replacement policy for file system buffer cache
  - LRU replacement is one possibility; but sequential access is very likely in file system I/O
  - MRU or free-behind

File System Prefetching

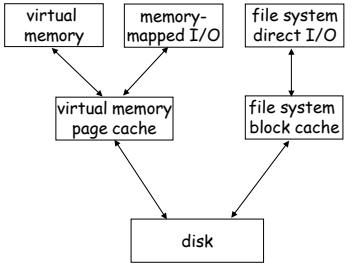
- File content is read ahead of time for anticipated use in the near future
- Often sequential (based on past access history on the file)
- What is the advantage of file prefetching?
- What is the danger of file prefetching?
- A balanced scheme that provides competitive performance to the optimal scheme [Li et al. EuroSys 2007]

Informed Prefetching

- Informed prefetching – prefetching while utilizing some information about application data access pattern
- Application I/O hints [Cao et al. 1994] [Patterson et al. 1995]
- Automatic I/O hints based on speculative execution [Chang&Gibson 2000], [Fraser&Chang 2003]

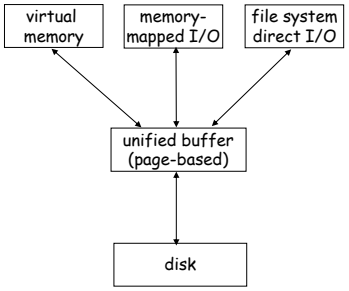
Buffer Cache in Main Memory

- Memory-mapped I/O naturally share page cache with the virtual memory system
- Problems:
  - double buffering
  - inconsistencies



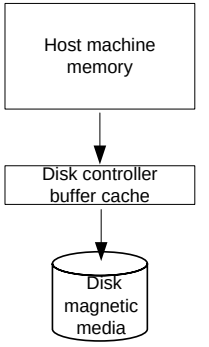
Unified Buffer Cache & Unified Virtual Memory

- A unified buffer cache uses the same page cache to store [Pai et al. 1999]
  - virtual memory pages
  - memory-mapped pages
  - file system direct I/O data



Multi-level I/O Buffer

- buffer cache in the main memory
- track cache on the disk controller



Protection in UNIX

- Protection domains: users
- Access matrix for files:
  - a simplified access control list
- Protection commands for files:
  - each user can change protection on files it owns
  - superuser can do everything

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).