CS 537
Lecture 16
Secondary Storage

Michael Swift

Secondary storage

- Secondary storage typically:
  - is anything that is outside of “primary memory”
  - does not permit direct execution of instructions or data retrieval via machine load/store instructions
- Characteristics:
  - it’s large: 80GB-1TB
  - it’s cheap: 0.30¢/GB
  - it’s persistent: data survives power loss
  - it’s slow: 100us-10 ms to access (compared to 100ns for ram)
    - why is this slow??

Motivation: I/O is Important

Applications have two essential components:
- Processing
- Input/Output (I/O)
  - What applications have no input? no output?

I/O performance predicts application performance
- Amdahl’s Law: If continually improve only part of application (e.g., processing), then achieve diminishing returns in speedup
- `f`: portion of application that is improved (e.g., processing)
- $speedup_f$: speedup of portion of application
- $Speedup_{Application} = 1/ ((1-f) + (f/speedup_f))$
  - Example:
    - f = 1/2, $speedup_f$ = 2, $speedup_{app}$ = 1.33
    - f = 1/3, $speedup_f$ = 2, $speedup_{app}$ = 1.20

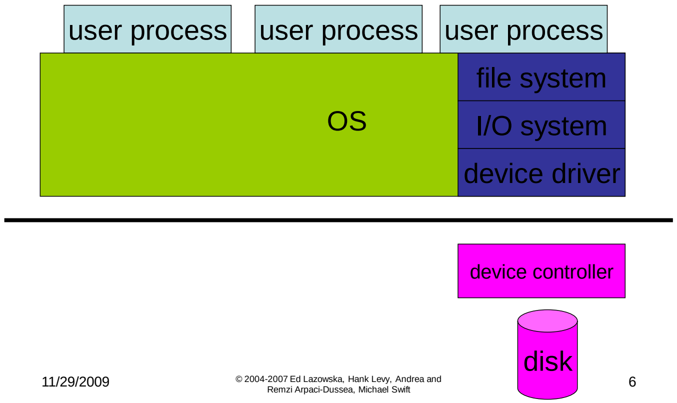
Disk trends

- Disk capacity, 1975-1989
  - doubled every 3+ years
  - 25% improvement each year
  - factor of 10 every decade
  - exponential, but far less rapid than processor performance
- Disk capacity since 1990
  - doubling every 12 months
  - 100% improvement each year
  - factor of 1000 every decade
  - 10x as fast as processor performance!

Disks and the OS

- Disks are messy, messy devices
  - errors, bad blocks, missed seeks, etc.
- Job of OS is to hide this mess from higher-level software
  - low-level device drivers (initiate a disk read, etc.)
  - higher-level abstractions (files, databases, etc.)
- OS may provide different levels of disk access to different clients
  - physical disk block (surface, cylinder, sector)
  - disk logical block (disk block #)
  - file logical (filename, block or record or byte #)

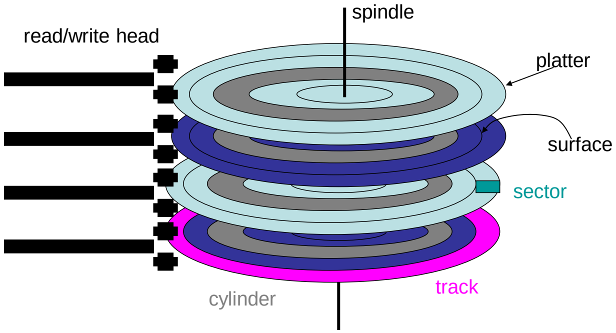
I/O System



Device Drivers: When is I/O complete?

- Polling
  - Handshake by setting and clearing flags
    - Controller sets flag when done
    - CPU repeatedly checks flag
  - Disadvantage: Busy-waiting
    - CPU wastes cycles when I/O device is slow
    - Must be attentive to device, or could lose data
- Interrupts: Handle asynchronous events
  - Controller asserts interrupt request line when done
  - CPU jumps to appropriate interrupt service routine (ISR)
    - Interrupt vector: Table of ISR addresses
    - Index by interrupt number
  - Low priority interrupts postponed until higher priority finished
  - Combine with DMA: Do not interrupt CPU for every byte

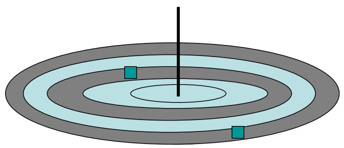
Disk Terminology



ZBR (Zoned bit recording): More sectors on outer tracks

Disk Performance

- How long to read or write n sectors?
  - Positioning time + Transfer time (n)
  - Positioning time: Seek time + Rotational Delay
  - Transfer time: n / (RPM * bytes/track)



Disk performance

- Performance depends on a number of steps
  - seek: moving the disk arm to the correct cylinder
    - depends on how fast disk arm can move
      - seek times aren't diminishing very quickly (why?)
  - rotation (latency): waiting for the sector to rotate under head
    - depends on rotation rate of disk
      - rates are increasing, but slowly (why?)
  - transfer: transferring data from surface into disk controller, and from there sending it back to host
    - depends on density of bytes on disk
      - increasing, and very quickly
- When the OS uses the disk, it tries to minimize the cost of all of these steps
  - particularly seeks and rotation

Disk Calculations

- Example disk:
  - #surfaces: 4
  - #tracks/surface: 64K
  - #sectors/track: 1K (assumption??)
  - #bytes/sector: 512
  - RPM: 7200 = 120 tracks/sec
  - Seek cost: 1.3ms - 16ms
- Questions
  - How many disk heads? How many cylinders?
  - How many sectors/cylinder? Capacity?
  - What is the maximum transfer rate (bandwidth)?
  - Average positioning time for random request?
  - Time and bandwidth for random request of size:
    - 4KB?
    - 128 KB?
    - 1 MB?

Interacting with disks

- In the old days…
  - OS would have to specify cylinder #, sector #, surface #, transfer size
    - i.e., OS needs to know all of the disk parameters
- Modern disks are even more complicated
  - not all sectors are the same size, sectors are remapped, …
  - disk provides a higher-level interface, e.g., SCSI
    - exports data as a logical array of blocks [0 … N]
    - maps logical blocks to cylinder/surface/sector
    - OS only needs to name logical block #, disk maps this to cylinder/surface/sector
    - on-board cache
    - as a result, physical parameters are hidden from OS
      - both good and bad

Disk Controller

- Responsible for interface between OS and disk drive
  - Common interfaces: ATA/IDE vs. SCSI
    - ATA/IDE used for personal storage: slow rotation, seek, high capacity
    - SCSI for enterprise-class storage: faster rotation and seek
    - QUESTION: which will be larger diameter? Which will have more platters?
- Basic operations
  - Read block
  - Write block
- OS does not know of internal complexity of disk
  - Disk exports array of Logical Block Numbers (LBNs)
  - Disks map internal sectors to LBNs
- Implicit contract:
  - Large sequential accesses to contiguous LBNs achieve much better performance than small transfers or random accesses

Disk Abstraction

- How should disk map internal sectors to LBNs?
- Goal: Sequential accesses (or contiguous LBNs) should achieve best performance
- Approaches:
  - Traditional ordering
  - Serpentine ordering

Reliability

- Disks fail more often....
  - When continuously powered-on
  - With heavy workloads
  - Under high temperatures
- How do disks fail?
  - Whole disk can stop working (e.g., motor dies)
  - Transient problem (cable disconnected)
  - Individual sectors can fail (e.g., head crash or scratch)
    - Data can be corrupted or block not readable/writable
- Disks can internally fix some sector problems
  - ECC (error correction code): Detect/correct bit flips
  - Retry sector reads and writes: Try 20-30 different offset and timing combinations for heads
  - Remap sectors: Do not use bad sectors in future
    - How does this impact performance contract??

Buffering

- Disks contain internal memory (2MB-16MB) used as cache
- Read-ahead: “Track buffer”
  - Read contents of entire track into memory during rotational delay
- Write caching with volatile memory
  - Immediate reporting: Claim written to disk when not
  - Data could be lost on power failure
    - Use only for user data, not file system meta-data
- Command queueing
  - Have multiple outstanding requests to the disk
  - Disk can reorder (schedule) requests for better performance

Role of OS for I/O

- Standard library
  - Provide abstractions, consistent interface
  - Simplify access to hardware devices
- Resource coordination
  - Provide protection across users/processes
  - Provide fair and efficient performance
    - Requires understanding of underlying device characteristics
- User processes do not have direct access to devices
  - Could crash entire system
  - Could read/write data without appropriate permissions
  - Could hog device unfairly
- OS exports higher-level functions
  - File system: Provides file and directory abstractions
  - File system operations: mkdir, create, read, write

File systems

- The concept of a file system is simple
  - the implementation of the abstraction for secondary storage
    - abstraction = files
  - logical organization of files into directories
    - the directory hierarchy
  - sharing of data between processes, people and machines
    - access control, consistency, …

Abstraction: File

- User view
  - Named collection of bytes
    - Untyped or typed
    - Examples: text, source, object, executables, application-specific
  - Permanently and conveniently available

Files

- A file is a collection of data with some properties
  - contents, size, owner, last read/write time, protection …
- Files may also have types
  - understood by file system
    - device, directory, symbolic link
  - understood by other parts of OS or by runtime libraries
    - executable, dll, source code, object code, text file, …
- Type can be encoded in the file's name or contents
  - file extension: .com, .exe, .bat, .dll, .jpg, .mov, .mp3, …
  - content: #! for scripts
- Operating system view
  - Map bytes as collection of blocks on physical non-volatile storage device
    - Magnetic disks, tapes, NVRAM, battery-backed RAM
    - Persistent across reboots and power failures

File Meta-Data

- Meta-data: Additional system information associated with each file
  - Name of file
  - Type of file
  - Pointer to data blocks on disk
  - File size
  - Times: Creation, access, modification
  - Owner and group id
  - Protection bits (read or write)
  - Special file? (directory? symbolic link?)
- Meta-data is stored on disk
  - Conceptually: meta-data can be stored as array on disk

File access methods

- Some file systems provide different access methods that specify ways the application will access data
  - sequential access
    - read bytes one at a time, in order
  - direct access
    - random access given a block/byte #
  - record access
    - file is array of fixed- or variable-sized records
  - indexed access
    - FS contains an index to a particular field of each record in a file
    - apps can find a file based on value in that record (similar to DB)
- Why do we care about distinguishing sequential from direct access?
  - what might the FS do differently in these cases?

File Operations

- Create file with given pathname /a/b/file
  - Traverse pathname, allocate meta-data and directory entry
- Read from (or write to) offset in file
  - Find (or allocate) blocks of file on disk; update meta-data
- Delete
  - Remove directory entry, free disk space allocated to file
- Truncate file (set size to 0, keep other attributes)
  - Free disk space allocated to file
- Rename file
  - Change directory entry
- Copy file
  - Allocate new directory entry, find space on disk and copy
- Change access permissions
  - Change permissions in meta-data

Opening Files

Expensive to access files with full pathnames
- On every read/write operation:
  - Traverse directory structure
  - Check access permissions

Open() file before first access
- User specifies mode: read and/or write
- Search directories for filename and check permissions
- Copy relevant meta-data to open file table in memory
- Return index in open file table to process (file descriptor)
- Process uses file descriptor to read/write to file

Per-process open file table
- Current position in file (offset for reads and writes)
- Open mode

Enables redirection from `stdout` to particular file

Directories

- Directories provide:
  - a way for users to organize their files
  - a convenient file name space for both users and FS's
  - a map from file name to blocks of file data on disk
    - Actually, map file name to file meta-data (which enables one to find data on disk)
- Most file systems support multi-level directories
  - naming hierarchies (/, /usr, /usr/local, /usr/local/bin, …)
- Most file systems support the notion of current directory
  - absolute names: fully-qualified starting from root of FS

```
bash$ cd /usr/local
```

  - relative names: specified with respect to current directory

```
bash$ cd /usr/local   (absolute)
bash$ cd bin          (relative, equivalent to cd /usr/local/bin)
```


Directory internals

- A directory is typically just a file that happens to contain special metadata
  - directory = list of (name of file, file attributes)
  - attributes include such things as:
    - size, protection, location on disk, creation time, access time, …
  - the directory list is usually unordered (effectively random)
    - when you type "ls", the "ls" command sorts the results for you

Directories: Tree-Structured

- Directory listing contains <name, index>, but name can be directory
  - Directory is stored and treated like a file
  - Special bit set in meta-data for directories
    - User programs can read directories
    - Only system programs can write directories
  - Specify full pathname by separating directories and files with special characters (e.g., \ or /)
- Special directories
  - Root: Fixed index for meta-data (e.g., 2)
  - This directory: .
  - Parent directory: ..

Path name translation

- Let's say you want to open "/one/two/three"

```
fd = open("/one/two/three", O_RDWR);
```

- What goes on inside the file system?
  - open directory "/" (well known, can always find)
  - search the directory for "one", get location of "one"
  - open directory "one", search for "two", get location of "two"
  - open directory "two", search for "three", get loc. of "three"
  - open file "three"
  - (of course, permissions are checked at each step)
- FS spends lots of time walking down directory paths
  - this is why open is separate from read/write (session state)
  - OS will cache prefix lookups to enhance performance
    - /a/b, /a/bb, /a/bbb all share the "/a" prefix

Acyclic-Graph Directories

- More general than tree structure
  - Add connections across the tree (no cycles)
  - Create links from one file (or directory) to another
- Hard link: “ln a b” (“a” must exist already)
  - Idea: Can use name “a” or “b” to get to same file data
  - Implementation: Multiple directory entries point to same meta-data
  - What happens when you remove a? Does b still exist?
    - How is this feature implemented???
  - Unix: Does not create hard links to directories. Why?

Acyclic-Graph Directories

- Symbolic (soft) link: “ln -s a b”
  - Can use name “a” or “b” to get to same file data, if “a” exists
  - When reference “b”, lookup soft link pathname
  - b: Special file (designated by bit in meta-data)
    - Contents of b contain name of “a”
    - Optimization: In directory entry for “b”, put soft link filename “a”