Folk Chapter 3

Secondary Storage and System Software

Organization of Disks

- Disks *platters* rotate under a read/write head
- Disk partitions:
  - **Track** – *radial* partition
  - **Cylinder** – *radial* partition {a particular track from all platter surfaces}
  - **Sector** – *angular* partition
- **Read/Write Head**
  - At end of cantilever (actuator arm)
  - Moves radially

Organization of Disks

- To read a particular byte:
  - Appropriate <u>surface</u>, <u>track</u>, and <u>sector</u> identified by OS
  - Then *entire* sector read into <u>buffer</u>
  - Finally, desired byte located in buffer

Disk Capacities

- T – track, S – sector, b – byte, C – cylinder, D – drive
- T = b/S * S/T = b/T
- C = b/T * T/C = b/C
  - { = b/S * S/T * T/C}
- D = b/C * C/D = b/D
  - {= b/S * S/T * T/C * C/D}

Sector Organization of Tracks

- Logical vs. physical storage schema
  - Disk controller delay while processing sector can lead to not being ready for successive contiguous sector
  - Solved by interleaving
  - Obviated by sufficiently improved controller speeds in contemporary systems

Sector Organization of Tracks

- **Clusters**
  - A fixed number of contiguous sectors
  - Sector/Cluster correspondence maintained by the File Allocation Table (FAT)
  - Sectors/cluster ratio adjustable by sysadmin
- **Extents**
  - Clusters organized into a set of one or more extents.
  - Each contiguous set of clusters for a file is an extent.
  - Extents for a file are non-contiguous.
  - Increasing the number of extents/file tends to increase the number of seeks/file

Sector Organization of Tracks

- **Fragmentation**
  - Internal fragmentation w.r.t. sectors: wasted space in a sector when sector size is not an integral multiple of record size, and records not *spanning* two sectors.
  - W.r.t clusters: wasted space in a cluster when the number of bytes/file isn't an integral multiple of the cluster size.

Block Organization of Tracks

- Disk blocks
  – Sizes can vary (some user control)
  – Not Unix system blocks
  – Alternative to sector organization
  – Obviates internal fragmentation problem
  – <u>Blocking factor:</u> records/block
  – Subblocks contain additional count and key info. regarding the block
    - Count: # of bytes in respective data block
    - Key: key for last record in data block – can allow more efficient searching by drive of track for block w. given key

Nondata Overhead

- Preformatting overhead
  - Sectors – at front of each sector: sector address, track address, condition (defective?); gaps and synchronization marks
  - Blocks – sub-block & inter-block gaps
    - Overhead can vary w. block sizes
    - More overhead w. blocks vs. sectors
      - Some overhead visible to programmer
    - Greater block sizes can leads to greater potential amount of internal track fragmentation

Costs of Disk Access

1. Seek Time (delay): time for move of r/w head to destination cylinder
   - Proportional to relative distance from starting cylinder to destination
   - Average seek distance: 1/3 total # of tracks
2. Rotational Delay: time until destination sector/block is under r/w head
   - Can be all but eliminated for sequentially written files, both within a track and between tracks if the *beginning* of each successive track is staggered
3. Transfer Time (delay): time until all data (sectors/blocks) pass under r/w head

Issues Affecting Disk Performance

- Sequential access provides for much better r/w throughput than random access
- Block size affects performance
  - Larger blocks can more than linearly improve throughput, but at expense of fragmentation space. (table 3.2)
  - Dividing large blocks into smaller sub-blocks (e.g. 8 ½K blocks in one 4K block) maintains throughput w. less fragmentation by using sub-blocks for smaller files

Mitigating Disk as Bottleneck

- Disk performance lags well behind LAN performance. The following techniques help to mitigate the disk bottleneck:
  - ***Multi*programming/*multi*processing**: CPU attends to other programs/processes while waiting on disk I/O
  - **Disk *striping***: partition file onto several drives enabling *simultaneous* access, i.e. *parallelism*
    - RAID 0: Large blocks split into full tracks gather/scattered w. large disk controller cache (*buffer*)
  - **Disk *cache***: large block or RAM *mirroring* <u>pages</u> of data from a disk (*buffer*ing)

Tape

- Practical for sequential access only
- Used for archiving as *tertiary* storage
- Becoming increasingly obsolescent

CD-Rom

- Inexpensive, durable, high-capacity archival medium
- Write once – read many
- Very slow seeks
- Constant Linear Velocity (CLV) single track spiraling out from center
  - rotation speed proportional to radial placement of r/w head
  - Adds capacity at expense of seek time
  - Addressing by:
    - minutes (up to 70); seconds (60/min); sectors (75 2Kbytes/second)
  - Compare w. Constant Angular Velocity (CAV) of magnetic hard disks w. constant rotation speed

Storage Hierarchy

- Trade-off: capacity vs. access speed & throughput
  1. Primary
     1. Registers
     2. Level 1 cache
     3. level 2 [& 3] cache
     4. RAM
  2. Secondary
     1. Magnetic disks
     2. LAN
  3. Tertiary & Offline
     1. Removable media
     2. Broader networks (e.g. WWW)

Journey of an I/O Byte

- File manager layers of programs
  - Upper symbolic/logical file aspects
    - Opened? Type (e.g. binary)? Owner? Access permitted?
  - Lower physical layers
    - Info from FAT
- *System* I/O Buffer
  - Ensures that data organization in memory and disk respectively conform

Journey of an I/O Byte

- I/O Processor & Disk Controller
  - I/O *processor*: external processing device that *gather/scatters* byte groups to/from external devices offloading work from CPU
    - DMA (direct memory access): when the I/O processor can take data directly from RAM w.o. involving the CPU
  - Disk Controller: controls & monitors the disk.
    - Responds to queries & instructions from the I/O processor

Buffer Management

- Buffer Bottlenecks
  - Conflict between *input* and *output* function
  - Solved by separate input and output buffers
  - Defn.: **I/O bound** – CPU mostly idle *waiting* for I/O to be performed
  - Double buffering: alternating the roles of a pair of input and output buffers
    - Allows the OS to operate on one while the other's being loaded or emptied

Buffer Management

- Multiple buffering
  - Buffer pooling: buffer selected from pool of available upon demand
  - Replacement strategies
    - LRU (least recently used)
  - Best # of buffers system & problem dependent.
  - Copies between system and program buffers (*move mode*) can be eliminated if the system provides the program w. pointers to the system buffers (*locate mode*)
  - Scatter/Gather I/O: r/w with single instruction and multiple buffers – *scatter* input & *gather* output

Unix I/O

- Kernel: bottom layer of Unix OS. (Fig.3.23)
  - Views all I/O as byte sequences
  - No logical view of a file
  - Block (normal files), Character (term./printer), and Network (sockets) I/O each w. their own device & interface drivers
  - 4 tables:
    - File descriptor: points to entries in open file table
    - Open file table: file structure info. re. open files (ephemeral)
    - File allocation (*inode*) [persists as long as file exists]
    - Index nodes: hard link – file name to inode

File name linkage

- Hard link – file name in directory w. pointer to inode
  – Opening a file uses hard link to bring inode into memory & entry in open file table
  – Can be multiple hard links to inode, the number of which is maintained as an inode field – deleting a hard link decrements that count
- Soft (symbolic) link – an association between file names
  – Can leave *dangling* links post file deletion

Block I/O & Device Drivers

- Unix block: randomly addressable array of fixed blocks
- Device Driver: set of routines to perform I/O between a device and the an I/O buffer
  - Allows the kernel to view a device only abstractly

The Kernel and File Systems

- File systems reside on disk – components imported to memory by the kernel *as needed*
- File system and kernel are *separate* entities
  - File systems can be configured (*tuned*) to a specific device or usage pattern w.o. changing the kernel view of files
  - The kernel can operate with different & possibly multiple file systems