I/O Management

CPU vs I/O speed

- CPU speeds continue to increase, and new CPUs have multiple processing elements on the same chip
- A large amount of data can be processed very quickly
- Problem in the transfer of data to CPU or even memory in a reasonable amount of time so that CPU has some work to do at all time

Classification of devices

- Human readable – monitor, keyboard, mouse
- Machine readable – disk/tape drives, sensors, controllers, actuators
- Communication – modems, routers
- Devices may be classified further even within classes by
  - Data rate
    - Some devices may not need to be very fast, for example keyboard to enter data
    - Graphics displays can always use a faster interface
  - Application
  - Complexity of control
  - Unit of transfer
  - Data representation
  - Error conditions
- OS controls all I/O devices
- Preferable to have the same interface for all I/O devices (*device independence*)

Organization of I/O function

- Programmed I/O
  - Under direct control of CPU
  - CPU issues a command on behalf of a process to an I/O module
  - CPU then waits for the operation to be completed before proceeding
- Interrupt driven I/O
  - CPU issues a command on behalf of a process to an I/O module
  - If the I/O instruction is nonblocking, CPU continues to execute next instruction[s] from the same process
  - If the I/O instruction is blocking, OS takes over, suspends the current process, and assigns CPU to another process
- Direct memory access (DMA)
  - DMA module controls exchange of data between memory and an I/O module
  - CPU sends a request to transfer a block of data to the DMA module and is interrupted only after the entire block has been transferred
  - DMA module takes over control of system bus to perform the transfer
  - CPU initiates the I/O by sending the following information to DMA module

    * Request type – read or write; using the read or write control line between the CPU and DMA module
    * Address of the I/O device, on the data line
    * Starting location in memory for read/write; communicated on data lines and stored by DMA module in its address register
    * Number of words to read/write; communicated on data lines and stored in the data count register
  - DMA may ignore the paging circuitry and access the address bus directly to transfer several disk sectors in a single I/O operation
    * This requires the requested buffers to be located in contiguous page frames

I/O device

- Any I/O device is hosted by one and only one bus
- Bus type affects the internal design of I/O device as well as its handling by the kernel
- I/O bus
  - Data path connecting a CPU to an I/O device
  - Intel x86 uses 16 of the address pins to address I/O devices and 8/16/32 of data pins to transfer data
  - I/O bus connected to an I/O device by a hierarchy of hardware components, including I/O ports, interfaces, and device controllers
- I/O ports
  - Every device connected to the I/O bus has its own set of I/O addresses
  - In the IBM PC architecture, I/O address space has up to 2^16 8-bit I/O ports
    * Two consecutive 8-bit ports may be regarded as a single 16-bit port, starting at an even address
    * Two consecutive 16-bit ports may be regarded as a single 32-bit port, starting at an address divisible by 4

Device drivers

- Device drivers interact with the rest of the kernel through a specific interface
- Abstract interfaces
  - An interface is a collection of routines and data structures that operate in a well-defined way
  - Network device drivers have to provide certain routines to operate on particular data structures
  - Generic layers of code using the services (interface) of lower layers of specific code
  - Network layer is generic and supported by device specific code that conforms to a standard interface
- Advantages of extra layer between hardware and applications
  1. Makes programming easier; users do not have to study low-level programming characteristics of hardware devices
     - Device-specific code can be encapsulated in a specific module
  2. Increases system security; kernel can check accuracy of request at the interface level before attempting to satisfy it
  3. Unifrom interface makes programs more portable; programs compiled and executed correctly on every kernel that offers the same set of interfaces
     - Vendors can add new devices without access to kernel source
     - Kernel deals with all devices in a uniform way
     - Possible to load/unload a driver as per needs; plug-and-play

Communicating with device files

- Device files
  - Represent physical devices
  - Possibly used for both input and output
  - Need a mechanism for device drivers in the kernel to receive output from process to send to device
  - Typically achieved by a function like `device_write`
- What if you need to communicate with the device itself?
  - Serial port connected to a modem
  - Above scheme will allow you to write data to modem and receive data from modem
    * Write either modem commands or data to be sent through phone line
    * Read response to modem command or data received through phone line
  - What to do when you need to talk to the serial port itelf?
    * Send the rate at which data is sent/received
- `ioctl` – I/O Control
  - Every device can have its own `ioctl` commands
  - `ioctl` function called with three parameters
    1. File descriptor of appropriate device
    2. `ioctl` number
    3. A parameter of type `long` so you can use a cast to use it to pass anything
  - `ioctl` number
    * Encodes the major device number, type of `ioctl`, command, and type of parameter
    * Usually created by a macro call (`_IO`, `_IOR`, `_IOW`, or `_IOWR` in a header file

Buffer cache

- System gathers requests to write to and read from the disk in chunks
- Helps with reduction of overhead associated with the operation
- In some applications (databases), system may need to access the data that was just written; buffer cache helps improve performance in such cases
- Caching
  - Use of available physical memory as cache for hard disk and other block devices
  - Disk access much slower than memory, making disk a bottleneck in system performance
  - Unix defers writing to disk as long as possible
    * Data read previously from disk and no longer used by any process stays in RAM
    * There is a good chance that new process will require data read from or written to disk by terminated processes
    * When process asks to access a disk, kernel first checks that the data is already in cache; if it is (cache hit), kernel can service the request without accessing the disk
  - `sync()` forces disk synchronization by writing all dirty buffers into disk
  - All OSs periodically write dirty buffers to disk to avoid data loss

Secondary Storage Management

- Secondary storage – An extension of primary storage
  - Must hold vast amount of data permanently
  - Capacity given by the following terminology

Disk Capacity in bytes			
10^3	Thousand	Kilobyte	KB
10^6	Million	Megabyte	MB
10^9	Billion	Gigabyte	GB
10^{12}	Trillion	Terabyte	TB
10^{15}	Quadrillion	Petabyte	PB
10^{18}	Quintillion	Exabyte	EB
10^{21}	Sextillion	Zettabyte	ZB
10^{24}	Septillion	Yottabyte	YB
10^{27}	Octillion	Ronnabyte	RB
10^{30}		Queccabyte	QB

  - Main memory
    * Too small to store all needed programs and data permanently
    * Volatile storage device
  - Magnetic tape
    * Quite slow in comparison to main memory
    * Limited to sequential access
    * Unsuitable to provide random access needed for virtual memory
  - Magnetic disks/HDDs, CDROMs, optical disks
    * The storage capacity is much larger
    * The price per bit is much lower; currently about $0.03 per GB
    * Information is not lost when power is turned off
  - Solid state drives SSDs, USB flash drives
    * Use solid-state memory to store persistent data
    * Cost much higher than HDD; currently about $0.20 per GB
    * Typically emulate the HDD interface, making it easily replacable in most machines
    * No moving parts, making them less fragile and silent than HDDs
    * No mechanical delays imply low access time and latency
    * Capacity consistently on the rise; currently about 1 TB for notebooks, up to 4 TB for desktops
    * Current SSDs are good for single task (single r/w) but do not work very well when two or more simultaneous r/w operations
      · Good for single task operations
      · Limitation more due to the fact that current OSs are not optimized for SSDs but are optimized for HDDs; Solaris ZFS and Microsoft exFAT are file systems optimized for SSD
      · There are Enterprise Flash Drives (EFDs) for applications that require high I/O performance, reliability, and energy efficiency

- Disk hardware
  - Physical structure
    * Disk surface divided into tracks
    * A read/write head positioned just above the disk surface
    * Information stored by magnetic recording on the track under read/write head
    * Fixed head disk
    * Moving head disk

        - Designed for large amount of storage
        - Primary design consideration cost, size, and speed
        - Head crash
    - Hardware for disk system
        - Disk drive, Device motor, Read/write head, Associated logic
        - Disk controller
            - Determines the logical interaction with the computer
            - Can service more than one drive (*overlapped seeks*)
        - Cylinder
            - The same numbered tracks on all the disk surfaces
            - Each track contains between 8 to 32 sectors
        - Sector
            - Smallest unit of information that can be read from/written into disk
            - Range from 32 bytes to 4096 bytes
        - Data accessed by specifying surface, track, and sector
        - View the disk as three dimensional array of sectors
        - OS treats the disk as one dimensional array of disk blocks
          s – Number of sectors per track
          t – Number of tracks per surface
          Disk address b of surface i, cylinder/track j, sector k

$$b = k + s \times (j + i \times t)$$

        - Seek time
            - Time required by read/write head to move to requested track
            - Includes the initial startup time and the time required to traverse the tracks to be crossed once the access arm is up to speed
            - Not necessarily a linear function of the number of tracks
        - Latency or rotational delay
            - Time required for the requested sector to come under the read/write head

- Device directory
    - Contains identification of files on the disk
        - Name of file
        - Address on the disk
        - Length, type, owner
        - Time of creation
        - Time of last use
        - Protections
    - Often stored in a fixed address

Master boot record/Master boot block

- CHS 0,0,0 on PCs
- Contains code to read the partition table that is located just past the MBR
- Finds the active partition from the partition table, and starts to read and execute instructions in the first block of active partition

- Viruses that affect MBR force the system to go to a different sector in the disk (typically the last sector) and start executing the code from there (boot viruses)
  - Friday the 13th virus: The code in the pointed sector checks CMOS for the date; if the date is 13th and day is Friday, it forces the system to erase the first few block of disk
- Safeguard: Disable the modification of MBR in CMOS
- Newer viruses may modify CMOS to enable the modification of MBR

Free-Space Management

- Free-space list – All disk blocks that are free
- Bit vector
  - Each block represented by a bit
  - Relatively simple approach
  - Efficient to find n consecutive free blocks on the disk
  - Uses bit manipulation instructions (Intel 80386, Motorola 68020/30)
  - Used by Apple Macintosh
  - Inefficient unless the entire vector kept in main memory for most accesses and occasionally written to disk for recovery
  - May not be feasible to keep the bitmap in memory for large disks
- Linked list
  - Link all free disk blocks together
  - Not efficient – to traverse the list, must read each block requiring substantial I/O time
- Grouping
  - Store the addresses of n free blocks in first free block
  - nth block contains the address of another n free blocks
- Counting
  - Several contiguous blocks may be allocated or freed en masse
  - Keep the address of first free block and the number n of free contiguous blocks that follow

Allocation Methods

- Problem – Allocate space to files so that
  - disk space is utilized effectively
  - files can be accessed quickly
- Assume a file to be a sequence of blocks
- Contiguous allocation
  - Each file occupies a set of contiguous addresses on disk
  - Number of disk seeks required to access contiguously allocated files is minimal
  - Seek time, if needed, is minimal
  - Defined by the disk address and number of blocks

- Straightforward file access
  - Sequential access – Remember the last block referenced and when necessary, read the next block
  - Direct access – To access block i of a file starting at b, access block $b + i$
- Problem in finding space for a new file
  - Equivalent to general dynamic storage allocation problem
  - Solution by first-fit, best-fit, and worst-fit strategies
  - External fragmentation
  - Must repack or compact files occasionally
  - Determining the size of file being created
  - A file growing slowly (over a period of a few months) must be allocated enough space for its final size

- Linked allocation
  - Each file a linked list of disk blocks
  - Disk blocks may be scattered anywhere on the disk
  - Directory contains a pointer to first and last block of file
  - Easy to fix the problems in contiguous allocation
  - No external fragmentation
  - No need to declare the size of a file
  - No need to compact disk space
  - Problems
    - Effective only for sequentially accessed files
    - Wasted space to keep pointers (2 words out of 512 $\Rightarrow$ 0.39% wastage)
    - Reliability – A bug might overwrite or lose a pointer
      Might be solved by doubly linked lists (more waste of space)
  - File Allocation Table (FAT)
    - Create a table on disk, indexed by block number
    - One entry for each disk block
    - Used as a linked list
    - Unused blocks indicated by a zero-valued entry
    - Used by MS-DOS and OS/2

- Indexed allocation
  - Bring all pointers into one block called *index block*
  - Index block for each file – disk-block addresses
  - ith entry in index block $\equiv$ ith block of file
  - Supports direct access without suffering from external fragmentation
  - Pointer overhead generally higher than that for linked allocation
  - More space wasted for small files
  - Size of index block
    - Preferred to be small
    - Linked scheme
      Normally taken as one disk block
      Larger files can be accommodated by linking together several index blocks

- ∗ Multilevel index
  Separate index block to point to index blocks which point to file blocks
  Assume 256 pointers to one index block
  65,536 pointers to two levels of index
  1K per block
  64M file
- ∗ Combined scheme
  BSD `Unix`
  First 15 pointers of the index block into device directory
  First 12 pointers point to *direct blocks*
  Data for small files do not need separate index block
  Block size of 4K ⇒ 48K of data accessed directly
  Next three pointers point to *indirect blocks*
  First indirect block pointer ≡ address of single indirect block
  Index block containing addresses of blocks that contain data
  Second indirect block pointer ≡ *double indirect block pointer*
  Contains address of a block that contains addresses of blocks that contain data
  Third indirect block pointer ≡ *triple indirect block pointer*

Disk Scheduling

- Disk service for any request must be as fast as possible
- Scheduling meant to improve the average disk service time
- Speed of CPU and memory seems to increase at about twice the rate compared to that of disks
  - Disks are about four times slower than main memory
- Disk performance parameters
  - Speed of service depends on
    - ∗ Seek time, most dominating in most disks
    - ∗ Latency time, or rotational delay
    - ∗ Data transfer time
  - Seek time and latency together denote the access time for the data block
- Each disk drive has a queue of pending requests
  - Each request made up of
    - ∗ Whether input or output
    - ∗ Disk address (disk, cylinder, surface, sector)
    - ∗ Memory address
    - ∗ Amount of information to be transferred – (byte count)
  - After making the request, the process has to wait for the device to be allocated to it
    - ∗ If there are multiple drives controlled by an I/O channel, there may be additional delay for the channel to be available
    - ∗ After device is allocated, the seek operation starts
  - Rotational pre-sensing (RPS)
    - ∗ Technique to improve disk I/O performance
    - ∗ After issuing seek, the channel is released to handle other I/O operations
    - ∗ After completion of seek, device determines when the sector containing data will rotate under the head
    - ∗ As the sector approaches, device attempts to reestablish communication with the host

    - If control unit or channel is busy with another I/O, the reconnection attempt fails and the device must rotate one complete revolution before attempting to reconnect
      - Known as RPS *miss*

- FCFS Scheduling
  - First Come First Serve scheduling
  - Simplest form of disk scheduling
  - May not provide the best possible service
  - Ordered disk queue with requests on tracks

$$98, 183, 37, 122, 14, 124, 65, 67$$

  - Read/write head initially at track 53
  - Total head movement = 640 tracks
  - Wild swing from 122 to 14 and back to 124
  - Wild swings occur because the requests do not always come from the same process; they are interleaved with requests from other processes
- SSTF Scheduling
  - Shortest Seek Time First scheduling
  - Service all requests close to the current head position before moving the head far away
  - Move the head to the closest track in the service queue
  - Example service queue can be serviced as

$$53, 65, 67, 37, 14, 98, 122, 124, 183$$

  - Total head movement of 236 tracks
  - May cause starvation of some requests
  - Not optimal
    - Consider the service schedule as

$$53, 37, 14, 65, 67, 98, 122, 124, 183$$

    - Total head movement of 208 tracks
- SCAN Scheduling
  - Also called *elevator algorithm* because of similarity with building elevators
  - Head continuously scans the disk from end to end
  - Read/write head starts at one end of the disk
  - It moves towards the other end, servicing all requests as it reaches each track
  - At other end, direction of head movement is reversed and servicing continues
  - Assume head moving towards 0 on the example queue

$$53, 37, 14, 0, 65, 67, 98, 122, 124, 183$$

  - Total head movement of 236 tracks
  - Upper time bound of twice the number of cylinders on any request
  - Few requests as the head reverses direction

  - Heaviest density of requests at the other end

- C-SCAN Scheduling
  - Circular SCAN
  - Variation of SCAN scheduling
  - Move the head from one end to the other
  - Upon reaching the other end, immediately come back to the first end without servicing any requests on the way

- LOOK Scheduling
  - Move the head only as far as the last request in that direction
  - No more requests in the current direction, reverse the head movement
  - *Look* for a request before moving in that direction
  - LOOK and C-LOOK scheduling

Selecting a Disk-Scheduling Algorithm

- Natural appeal in SSTF scheduling
- SCAN and C-SCAN more appropriate for systems that place heavy load on disk
- Performance heavily dependent on number and types of requests
- Requests greatly influenced by file allocation method
  - Contiguously allocated file generates several requests close together on the disk
  - Linked allocation might include blocks that are scattered far apart
- Location of directories and index blocks
  - Directory accessed upon the first reference to each file
  - Placing directories halfway between the inner and outer track of disk reduced head movement

File Systems

- Data elements in file grouped together for the purpose of access control, retrieval, and modification
- Logical records packed into blocks
- File system in Unix
  - Significant part of the Unix kernel
  - Accesses file data using a buffering mechanism to control data flow between kernel and I/O devices
- Directory Structure
  - Files represented by entries in a *device directory*
  - Information in the device directory
    - Name of file
    - Location of file
    - Size of file
    - Type of file
  - Device directory may be sufficient for single user system with limited storage
  - With increase in number of users and amount of storage, a directory *structure* is required

  - Directory structure
    * Provides a mechanism to organize many files in a file system
    * May span device boundaries and may include several different disk units
    * May even span disks on different computers
  - User concerned only with logical file structure
  - Systems may have two separate directory structures
    * Device directory
      Describes the physical properties of each file – location, size, allocation method, etc.
    * File directory
      Describes the logical organization of files on all devices
      Logical properties of the file – name, type, owner, accounting information, protection, etc.
      May simply point to the device directory to provide information on physical properties

Hierarchical Model of the File and I/O Subsystems

- Average user needs to be concerned only with logical files and devices
- Average user should not know machine level details
- Unified view of file system and I/O
- Hierarchical organization of file system and I/O
  - File system functions closer to the user
  - I/O details closer to the hardware
- Functional levels
  - Directory retrieval
    * Map from symbolic file names to precise location of the file, its descriptor, or a table containing this information
    * Directory is searched for entry to the referenced file
  - Basic file system
    * Activate and deactivate files by opening and closing routines
    * Verifies the access rights of user, if necessary
    * Retrieves the descriptor when file is opened
  - Physical organization methods
    * Translation from original logical file address into physical secondary storage request
    * Allocation of secondary storage and main storage buffers
  - Device I/O techniques
    * Requested operations and physical records are converted into appropriate sequences of I/O instructions, channel commands, and controller orders
  - I/O scheduling and control
    * Actual queuing, scheduling, initiating, and controlling of all I/O requests
    * Direct communication with I/O hardware
    * Basic I/O servicing and status reporting

Consistency Semantics

- Important criterion for evaluation of file systems that allows file sharing
- Specifies the semantics of multiple users accessing a file simultaneously

- Specifies when modifications of data by one user are observable by others
- File session
  - Series of accesses between an open and close operation by the same user on the same file
- Unix Semantics
  - Writes to an open file by a user are visible immediately to other users that have this file open at the same time
  - There is a mode of sharing where users share the pointer of current location into the file. Advancing of pointer by one user affects all sharing users. A file has single image that interleaves all accesses, regardless of their origin

File Protection

- Save the file from
  - Physical damage – Reliability
    * Damage possible because of
      Hardware problems – error in read/write
      Power surge or failure
      Head crash
      Dirt and temperature
      Vandalism
      Accidental deletion
      Bugs in file system software
    * Duplicate copies of files
    * File backup at regular intervals
  - Improper access – Protection
    * Physical removal of floppies and locking them up
    * Problem in large system due to need to provide shared access to the files
    * Extremes
      Provide complete access by prohibiting access
      Provide free access with no protection
    * Controlled access
      Access by limiting the types of possible file accesses
      Read access
      Write access
      Execute access
      Append access
      Delete access
      Rename
      Copy
      Edit
    * Protection for directories
      Create a file in the directory
      Delete a file in the directory
      Determine the existence of a file in the directory
- Protection associated with
  - File by itself
  - Path used to access the file
  - With numerous path names, a user may have different access rights to a file dependent upon the path used
  - Protection based on names

    - If a user cannot name a file, he cannot operate on it
- Protection based on passwords
    - Associate a password with each file
    - Access to each file controlled by password
    - Distinct password with each file – too many passwords to remember
    - Same password for all files – once password broken, all files accessible
    - Associate password with subdirectories (TOPS 20)
    - Multiple level passwords
- Protection based on access lists
    - Associate access list with each file containing names of users and types of accesses allowed
    - Problems with access lists
      Constructing access lists is tedious
      List of users to be allowed certain access may not be known in advance
      Space management problem in directory entry to account for variable list size
- Protection based on access groups
    - Classify users into groups
      Owner
      Group
      Universe
    - Requires strict control of group membership
    - Unix allows groups to be created and managed only by root
    - Only three fields needed to provide protection – rwx