

I/O Management

- OS controls all I/O devices
- Preferable to have the same interface for all I/O devices (*device independence*)

Secondary Storage Management

- Secondary storage – An extension of primary storage
 - Must hold vast amount of data permanently
 - Main memory is too small to store all needed programs and data permanently
 - Main memory is 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, CDROMs, Optical disks
 - * The storage capacity is much larger
 - * The price per bit is much lower
 - * Information is not lost when power is turned off
- 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 cylinder

Disk address b of cylinder i , surface j , sector k

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

- * Seek time

Time required by read/write head to move to requested track

Farther apart the tracks, more the seek time

- * Latency time

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

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

- n th 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
- i th entry in index block \equiv i th 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 \Rightarrow 48K of data accessed directly
 - Next three pointers point to *indirect blocks*
 - First indirect block pointer \equiv address of single indirect block
 - Index block containing addresses of blocks that contain data
 - Second indirect block pointer \equiv *double indirect block pointer*
 - Contains address of a block that contains addresses of blocks that contain data
 - Third indirect block pointer \equiv *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 service depends on
 - Seek time, most dominating in most disks
 - Latency time, or rotational delay
 - Data transfer time
- 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)
- 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 263 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