

# File Storage and Disk Performance

CIT 595  
Spring 2007

## Disk as Secondary Storage

Convenient for storing:

- *Rewritable*: data can be read from a sector, modified and written back into same place
- *Random/Direct Access*: each sector can be accessed independently of other sectors
  - data need not be stored in contiguous order

CIT 595

18 - 2

## Data Storage

- To access data in low-level format is cumbersome
  - Need to know about tracks and sectors
- For convenience, OS provides a uniform logical view of information storage via the *File-System*
  - Logical storage unit a.k.a *file*
  - *Directory structure* for organization and information of all files in the system
- A file is nothing more than a sequence of bytes
- A file is stored in one or more *blocks* on disk

CIT 595

18 - 3

## Block

- A block is unit of transfer between Memory and Disk
- A block comprises of one or more sectors of the disk
  - Depends on file-system used by an OS
  - If sector is 512 bytes, block sizes ranges from 2,048 to 32,768 bytes
- Operating System (OS) maintains which blocks belong to a file by assigning blocks with logical *block numbers*
  - Starting at 0 and running through consecutive integers up to some maximum
- The logical block number is converted into *physical disk address* before data request is made to the disk controller
  - Physical Disk Address -> track/cylinder, surface, sector

CIT 595

18 - 4

## High-Level or Logical Formatting

- Low-level formatting is to format disk into tracks and sectors
  - Done during disk manufacturing
- To use a disk to hold files, the OS records its own data structures on the disk
  - Known as *high-level* or *logical* formatting
  - The formatting tells us to store and access the file-system
- But before logical formatting we need to perform one more thing – partition the disk!

CIT 595

18 - 5

## Disk Partitioning

- Is the creation of *logical* divisions (isolated sections) of a hard disk
  - Partitioned into one or more groups of cylinders
  - Gives appearance of more than one hard drive e.g. C drive, D drive
  - So disk can be prepared for use, or even dedicated to different uses
- Partitions also improve disk efficiency
  - Some file systems used by OS, assign block sizes based on disk size
    - Larger disk size tend to have larger block size
  - Large blocks result in wasted disk space
    - Remember that a file will atleast occupy one block
- Can apply OS-specific logical formatting per partition
  - Allows user can have more than one OS on his/her computer
  - E.g. Windows and Unix

CIT 595

18 - 6

## Information kept on Disk for File-System

Following information is maintained by OS on Disk:

1. Information about how to *boot* the system
2. Information on *partition details* such as partition size, size of blocks, count of free blocks, type of file-system
3. Information on *accessing* files
  - Called File Allocation Table in Windows File System
  - Called Inode in Unix File System
4. Directory structure
5. Files

Note: Logical/High-level Format = All the above information per partition

CIT 595

18 - 7

## Boot Block

- Occasionally, a computer will need to update the BIOS
  - E.g. If new devices and standards arise
  - If the entire BIOS is stored in *ROM*, changing it is hard
- Hence, boot program stored in ROM only does part of the work
  - Instructs the disk controller to read the boot blocks into memory
- The boot block contains the main BIOS program
  - Load the OS
  - Load device drivers (software) that identifies and interacts I/O hardware
- Always contained at fixed location on disk

CIT 595

18 - 8

## Directory

- Directory provides information needed to find the disk data blocks of a file
- Each entry in the directory contains
  - Symbolic file name
  - Disk address of the first block of file
  - File attributes: size, access-information, time and date
- A directory is nothing more than a *file itself*, except it is specially structured
- Root directory contains information about where each sub-directory is located
  - Always contained at fixed location on disk

CIT 595

18 - 9

## Block Allocation for files

- How do we allocate files on disk so that disk space is
  - Utilized effectively
  - Accessed Quickly
- Two kinds of Allocation
  - Contiguous
  - Linked

CIT 595

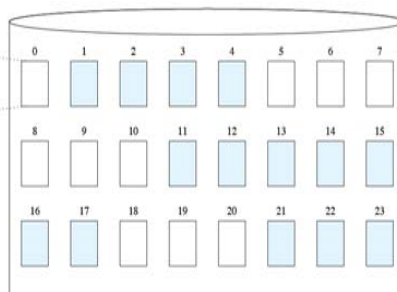
18 - 10

## Contiguous Allocation

- Requires files to occupy a set of contiguous blocks

Directory

File	Start	Size
CIT593doc	5	6
CIT595doc	18	3



- Assume each row is track with 8 sectors
- Assume 1 block = 1 sector

CIT 595

18 - 11

## Contiguous Allocation (contd..)

### Advantage:

- Random access is possible
- If all blocks of one file fit physically on track then
  - Disk seek is minimized after the starting block

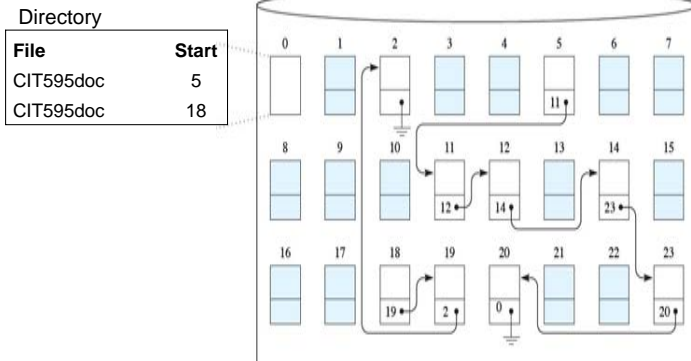
### Disadvantage:

- Finding contiguous space for file
- Suffers from *external fragmentation* i.e. disk is broken into pieces
  - A new request may not be able to fit into any of the pieces
  - Need to compact all free space into one continuous space - disk defragment

CIT 595

18 - 12

## Linked Allocation



CIT 595

18 - 13

## Linked Allocation (contd..)

### Advantage:

- No external fragmentation

### Disadvantage:

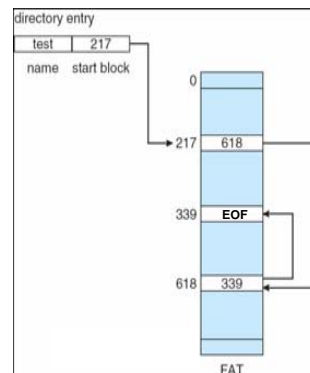
- Random Access is not possible
- Disk seek time is increased for a file as the blocks are now scattered all over
  - Pointer storage overhead
    - Some of the storage in each block devoted to the pointer

CIT 595

18 - 14

## File Allocation Table (FAT)

- Implementation of linked allocation
  - Pointers to data blocks are stored in FAT
- FAT is an array – one entry for each disk block
  - Indexed by the block number
  - Each array entry is a pointer to the next block
  - Stores info about all files
- FAT is also kept on disk
  - One per partition
  - Maintains info on disk blocks that are currently used by files, and which are free for use
  - Spread over couple blocks on disks



CIT 595

18 - 15

## FAT (contd..)

- Random Access problem is possible
- Does not solve the seek problem
  - Considerable arm movement to read the FAT and actual file blocks
- Seek time can be minimized by caching the FAT
  - Store table in main memory
  - No disk I/O needed for FAT lookup – only block requests are sent to disk
- Caching FAT also has another overhead
  - E.g. partition of 5 GB, 1KB block size
  - 5 Million block entries, each entry size will be ~3 bytes, hence FAT occupies 15 MB of Memory

CIT 595

18 - 16

## File Index

- Only a few files are open at any one time
  - So keep only necessary information about open files in memory to get good performance
- The *index block* groups together index information about each file individually
  - Called Inode in Unix File System
- The index block is an array of block numbers but it can only have a subset of block numbers
  - Less memory overhead when cached
- Updating the index block is like writing to any block on disk

CIT 595

18 - 17

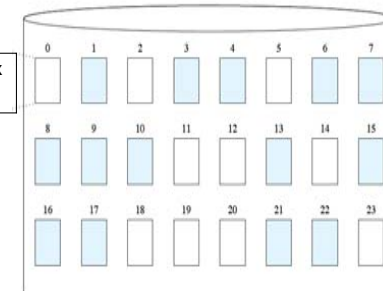
## File Index

Directory

File	Index
CIT595doc	0

Block 0

18
19
2
-1
-1
-1



CIT 595

18 - 18

## File Index (contd..)

- If the number of blocks need cannot fit in one index block, then we apply level of indirection
- Can suffer from internal fragmentation
  - A file may occupy only two blocks, but the block index can hold say 15 entries (block numbers)

CIT 595

18 - 19

## Aside: Note on Windows OS File System

- FAT technique was originally employed for MS-DOS and floppy-disk (1.44 MB)
  - Known as *FAT12* file-system
  - FAT12 ->  $2^{12}$  entries in the FAT
- Increase disk size -> Increase number of blocks -> FAT size
  - E.g. FAT16 and FAT32 (used latest till Windows 2000)
  - Also had restriction on partition sizes – FAT16 allowed 4GB partitions
- Draws backs
  - Bigger FATs
  - Lot of wasted space per block as each FAT had restriction on how many blocks the FAT can hold -> larger block sizes

CIT 595

18 - 20

### Aside: Note on Windows OS File System (contd..)

- Advent of Windows-NT and XP, new file system emerged
  - Called *NTFS* File System
- NTFS had some advanced features
  - Used for multi-user environments -> better security and protection
  - No limit on partition size
  - Uses compression techniques for storing file pointers
    - Compression employed for cluster sizes above 4KB

CIT 595

18 - 21

### Free-Space Management

- A *free-space list* is maintained that records all free blocks
- Implementation technique: *linked-list*
  - Just like linked allocation – Pointer to first free block is maintained in fixed location on disk
  - Or can be accounted in part of the allocation data structure
    - E.g. in FAT, block entries that not occupied can be distinguished with special value
- Implementation technique: *grouping*
  - Store the address of n free blocks in the first free block
  - n – 1 entries are actually free and the last block contains the address of the next n free blocks

CIT 595

18 - 22

### Disk Performance

- Low CPU utilization can actually indicates a problem in the I/O subsystem
  - CPU spends more time waiting than running
- Disk drives are the slowest memory component
- A slow disk system can drag down the performance of all programs when virtual memory paging is involved
- Optimal disk performance is critical to system throughput

CIT 595

18 - 23

### Disk Utilization

- Disk utilization is the percentage of the time the *disk is busy* servicing I/O requests
- It gives the probability that the disk will be busy when another I/O request arrives in the disk service queue
  - Esp. in a multiprocessing environment, where multiple processes are trying share the same disk resource
- $Utilization = Request\ Arrival\ Rate \div Disk\ Service\ Rate$ 
  - Arrival Rate is given in Requests per Second
  - Service Rate is given in Operations (Requests Served) per second
- If on avg. a request takes 15 ms to complete and request arrival rate is 33 requests/sec
  - Implies that Service Rate = 66.7 operations/second
  - Then Utilization is around 50 %

CIT 595

18 - 24

### Effect of Increasing Load in the System

- As the number of requests increases (load), the disk is going to be become more busy
- The wait in queue per request is also going to increase
- The amount of time that a request spends in the queue is given by queuing theory equation:

$$\text{Time in Queue} = (\text{Service time} \times \text{Utilization}) \div (1 - \text{Utilization})$$

- We basically want to *decrease* the Service Time

CIT 595

18 - 25

### Reducing Time in Queue

- Disk arm motion is the greatest consumer of service time
- Disk specifications cite average seek time, which is usually in the range of 5 to 10ms
- However, a full-stroke seek can take as long as 15 to 20ms
- *Schedule* disk access such that we minimize seek time
- Scheduling can be done either by OS or Disk controller

CIT 595

18 - 26

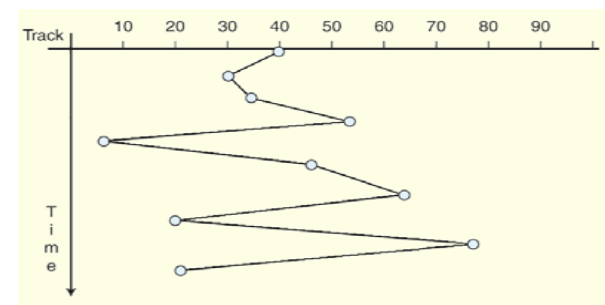
### Disk Scheduling

- Several algorithms exist to schedule the servicing of disk I/O requests
- Algorithms Minimize Seek time  $\approx$  Seek distance
  - i.e. how many tracks need to traveled between the current head position and the next request
- All examples illustrated with 0-99 tracks
  - Requests Order in Queue: 28, 35, 52, 6, 46, 62, 19, 75, 21
  - Current Track Serviced i.e. Current Head Position = 40

CIT 595

18 - 27

### First Come First Served (FCFS)

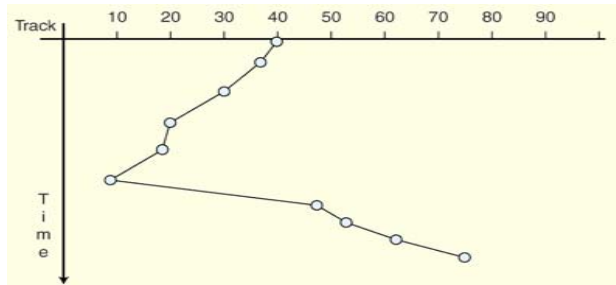


- Order of Service: 28, 35, 52, 6, 46, 62, 19, 75, 21
- Disk arm changes directions 6 times
- Traverses total of 291 tracks

CIT 595

18 - 28

### Shortest-Seek Time First



- Order of service : 35, 28, 21, 19, 6, 46, 52, 62, 75
- Disk arm changes direction only once for this example
- Traverses total of 103 tracks

CIT 595

18 - 29

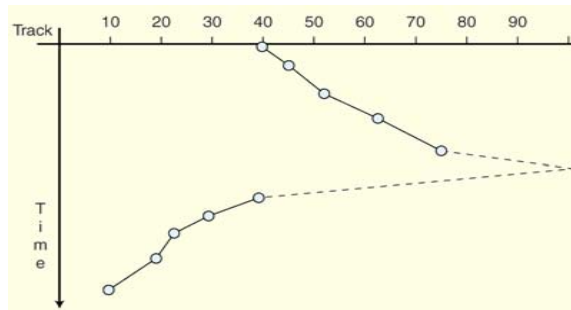
### SCAN

- Avoids SSTF starvation risk
- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests along the way
- When it gets to the other end of the disk the head movement is reversed and servicing continues
- Sometimes called the *elevator algorithm*
- While SCAN entails a lot of arm motion, the motion is constant and predictable

CIT 595

18 - 30

### SCAN (Cont.)



- Order of service : 46, 52, 62, 75, 35, 28, 21, 19, 6
- Disk arm changes direction only twice
- Traverses total of 99 tracks

CIT 595

18 - 31

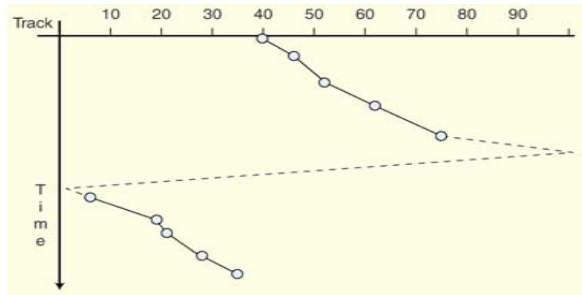
### C-SCAN

- A variant of 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
  - E.g. start at 0, reaches 99, and then starts again at 0
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

CIT 595

18 - 32

### C-SCAN (Cont.)



- Order of service : 46, 52, 62, 75, 6, 19, 21, 28, 35
- Disk reads in one direction only

CIT 595

18 - 33

### LOOK and C-LOOK

- Reduces the disk arm motion of SCAN and C-SCAN
- Instead of sweeping the entire disk, the disk arm travels only to the *highest*- and *lowest*-numbered tracks for which access requests are pending
- For example, the arm will traverse only 69 tracks
  - Traverse between 6 and 75
  - Saving 30% arm-motion

CIT 595

18 - 34

### Disk Performance

- File placement greatly influences performance
  - Even the best scheduling algorithm will suffer
- Even though most file-system use linked allocation, with deletion and creation of files disk becomes fragmented
  - The disk should be periodically defragmented
  - Chances are that blocks allotted to files will be continuous – i.e. over sectors belonging to the same track
- All modern computer systems come with defragging tool

CIT 595

18 - 35

### Disk Reliability

- Disk are highly unreliable
- Improve reliability by
  1. ECC
  2. Backup data on removable media
  3. Disk Redundancy
    - Duplicate or Mirrored Disk
    - Spread work on multiple independent disks
    - E.g. RAID systems
- All schemes have some cost associated
  - Hardware
  - Data overhead – extra/redundant/parity bits

CIT 595

18 - 36