* OS – File System, I/O and Protection

01 – Attributes and Operations on files

File Operations

- → Creating
- → Writing
- → Reading
- → Repositioning
- → Deleting
- → Truncating

File Attributes

- → Name
- → Identifier (extension)
- → Type
- → Location
- → Size (In Bytes)
- → Protection
- → Time and Date (creation or modification)

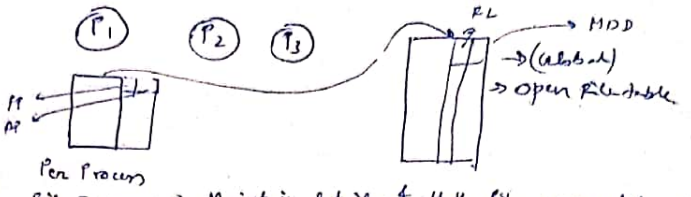
- Hard disk is divided into blocks which may accessed for read/write all at once.
- A large file is not generally stored contiguously in physical, all the blocks are contiguous logically.
- OS provides an abstract data structure which is called file to store the information logically contiguous.
- The logical file is taken by OS & Stored physically in the hard disk. The active thing is done by the file system
- Every information is saved as file by the OS into the hard disk.
- information → files → directory → File System
- OS maintains a pointer for each file to indicate till what part the file is written, so that after that point write operation can continue. (Same for reading)
- Sequential reading → read pointer
- Delete → Free space along the attributes of a file
- Truncate → Free up space, but not the attributes of file (Few changes are made to the attribute)

02- Open File Tables

Information required for accessing a file

i) File Pointer
ii) File open count
iii) Disk location of a file
iv) Access rights

- To access a file, the file must be opened first.
- Then OS keeps its details into a table for open files.



→ Maintain details of all the files accessed by P_1

- When File open count is zero, then that record from Global table is deleted.

03- Accessing Files

Access methods

→ Sequential access → eg- video audio files.
- In most of the OS, block size is 512 Bytes.
- Hard disk is further divided into sectors. Each sector is of size 512 Bytes.
- Logically file is sequential. Physically file may not be sequential
- 512 B is optimal (Experimentally proven)

→ Direct access
→ Indexed access

Database (indexed) (Array or tree).

eg- Database (OS can find the record using logical block number)

04- Directory Structure

- Hard disk is divided into partitions depending upon the number of OS or File system it will use. (volumes / mini disk)
- Each partition needs a special type of file called directory in which all the details of all the files is contained.
- Each directory entry holds the details of a file. (location, size etc)
- Data about data → Metadata → Directory (entries)
- Each partition contains atleast one directory.

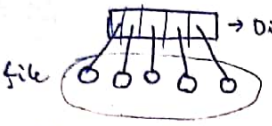
Directory operations:
i) Search
ii) Create
iii) Delete
iv) List
v) Traverse
vi) Rename

- For a large number of files level is used.
- One dir & all files → single level directory
- Various levels → multi level directory
  - Two level directory
  - Tree level directory
- Acyclic graph directories
- Graph directories.

05 - Single level VS Two level directory

Single level

- Entire file system will contain only one directory & it mentions all the files which are present in the file system.

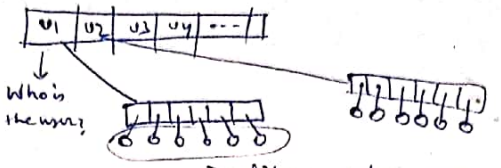


Adv → Implementation is very simple..
↳ If no. of files are very small, searching is very simple.

Dis → Naming Problem
↳ Access Right
↳ High search time
↳ Unorganized files

Two level

- Master file Directory



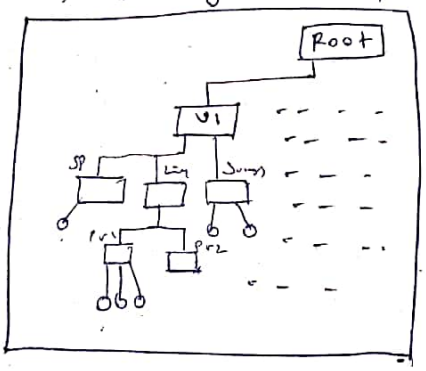
→ files created by user 1

PWD → Present Working Directory

- Diff dir, same file name ✓ (Adv)
- System file (Global) → Special directory
  ↳ All user share this
- Single level → Single user
- Two level → Multi user

06 - Tree Structured Directory

- Need for grouping of different types of files



Path → Address of file name (root/U1/bin/proj1/61.exe)

Absolute Path name :- Root to file complete address.

Relative Path name :- PWD/CD for each user, then file (bin/proj1/61.exe)

- New process created from a process also contain same path name

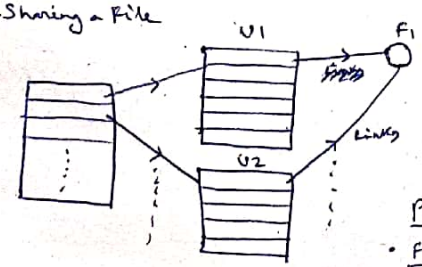
- permissioning :→ r/w/x
  Sharing is done using → read, write, execute

- Traversing is difficult

- Flexibility → User has the ability to group all the files together.
  ↳ User has ability to share a directory with other users.

07 - Acyclic Graph Structured Directory

- Sharing a file



- Even if it looks like a cycle, but due to directions there is no cycle, this is called acyclic graph directory (using links → Unix).

Problems

- File deletion :- Even after deletion of a file, the links still point to the deleted or emptied location. This is dangling pointer situation.

- Path Replication :- Same path of a file can be found at many places because of pointing.

[Not only file, directory can also be shared]

- At the ~~creation~~ creation time of a file, a separate data structure is used to store all the references by all the users. All its users are connected to the file by absolute links or relative links. But the problem is the data structure need to ~~be~~ be extended if there are too many links. So instead of storing all the ~~links~~ links, a counter is used to count the number of references. So, file is not deleted until all its references are deleted & count is set to zero. Unix follows this last approach.

08 - File System

- File System provides the mechanism for on-line storage and access to file contents, including data and programs.
- File System deals with following issues:
  - → File Structure
  - → To allocate diskspace
  - → Recovering freed space
  - → To track the locations of data
  - → Interface other parts of operating system to the secondary storage.

09 - FileSystem Structure

- One big software → divided into tasks → make various groups → a group of task is assigned to a layer to implement → Most OS follow this approach

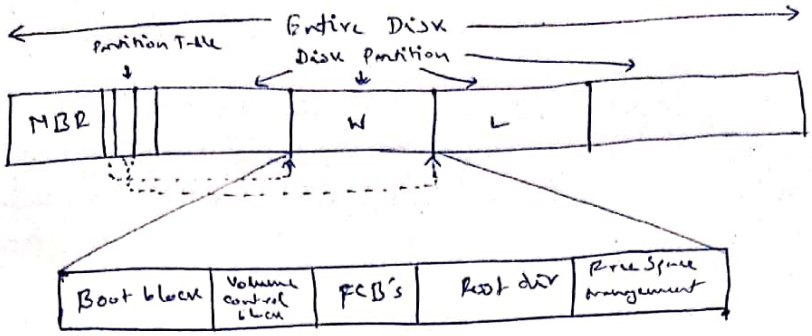
⇒ File Systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.

⇒

Application Programs
⇓
Logical File System (metadata, directory structure)
⇓
File Organization Module (Logical block ⇒ Physical block)
(Free space management)
⇓
Basic File System (Commands to the I/O control ~~buffering~~)
(Buffering)
⇓
I/O control (Consist device drivers, interrupt handlers)
⇓
Devices

- Hard disk block → buffers in main memory to be accessed by CPU (Buffering)
- Device drivers → access to secondary memory using given commands of BFS.
- Layered organization is used to prohibit interference between different modules.

10 - MBR



- BIOS (Basic Input Output System) :-

A small piece of code present on the ROM, which fetches the first block of the hard disk when a computer is ~~boot first~~ on for first time, ~~it~~ consists of MBR (Master Boot Record), loads into main memory. Now CPU starts executing from the first instruction of MBR. Now it will check whether the MBR is valid or not by checking its magic number & know ~~the~~ whether the disk is formatted or not. Then the message of selecting an OS is shown which is also contained in MBR & after that MBR contain Partition table, which tells from where each partition is going to start/begin.

- First block of every partition is Boot block, It knows where OS code is present & how to load that code.
- Depending on the File system, all the ~~to~~ blocks are going to differ.

11 - On Disk Data Structure Used in File System Implementation

- Data structures varies from one FS to another. (Win - NTFS, Lin - LFS, Win - FAT/EXT)

⇒ Several in memory and on disk structures are used to implement a File system. These structures vary depending on the operating system and the file system but general principles apply.

1st block Boot Control Block (Per Volume) :- It contains information needed by the system to boot an OS from the volume.

In Unix File System, it is called the boot block.

In NTFS, It is called the partition boot sector.

2nd block Volume ~~Cont~~ Control Block :- It contains volume ~~to~~ details such as the number of blocks in the partitions, size of the block.

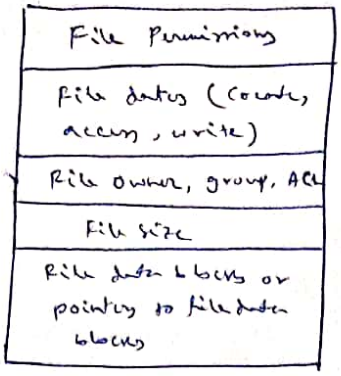
In UFS, it is called Super Block.

In NTFS, this information stored in master file table.

Directory Structure Per File System :- It is used to organize the files.

In UFS this includes file names and associated inode numbers.

File Control Block (FCB) :- It contains details about the file.



A typical File control block

[Mounting ⇒ whenever a new partition or volume is connected to the computer, it is called Mounting & information about this partition must be available in main memory.]

12 – In Memory Data structure in File System Implementation

The in-memory is used for both the file system management and performance improvement via caching. This data loaded at mount time and discarded at dismount.

i) In memory mount table :- It contains information about each mounted device.

ii) In memory directory structure cache :- It holds the directory information of recently accessed directory.

iii) System-wide open file table :- It contains the FCB of each open file.

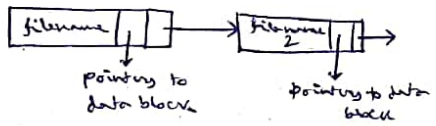
iv) Per process open file table :- It contains the pointer to the appropriate entry in the system wide open file table.

13 – Directory Implementation

⇒ The selection of directory allocation and directory management algorithms significantly affects the performance and reliability of file system.

Algorithms :-

1. Linear List :- • Files as a singly linked list.



• Entire list must be ~~scan~~ searched at the time of new file creation to ~~check~~ check whether the new file name exists or not then if it is doesn't exist it is added to the list (front or end).

• Open, delete the files large search time same as above.

2. Hash Table :-

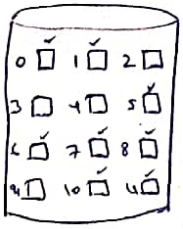
• Search will be faster
• So, open or deletion is better.
• Problem ⇒ fixed size, collision.

[If no. of files & directories are very large the B-trees & B+ trees are used.]

14 – Allocation Methods

⇒ How to allocate space to the files so that the disk space is utilized effectively and files can be accessed quickly.

Contiguous Allocation :- If blocks are allocated in such a way that ~~file~~ a file will get contiguous blocks. then it is called contiguous allocation.



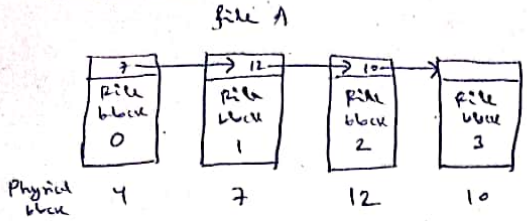
directory

file	start	length
jump	0	2
tr	5	4
test	10	2

Advantage :- • Simple to implement.
• Read performance is excellent.

Disadvantages :- The disk becomes fragmented. (Internal & External Fragmentation)

- Linked List Implementation :-

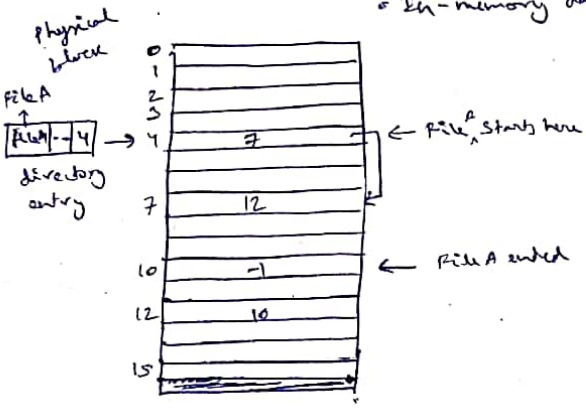


Adv :-
⇒ Unlike contiguous allocation every disk block can be used.
⇒ No space lost due to disk fragmentation (except for internal fragmentation in the last block of each file)

Disadv :- ⇒ Random Access is very slow.
⇒ Pointer takes up few bytes.

15 - File Allocation Table

- In-memory data structure (MM)



File Allocation Table

Adv :-
→ Random Access is easier

Disadv :-
→ Size can be too much i.e table can be very large.

16 - GATE IM on FAT

A FAT (File Allocation Table) based file system is being used and the total overhead of each entry in the FAT is 4 Bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size 10^3 bytes, the maximum size of a file that can be stored on disk in units of 10^6 bytes is ——.

FAT size = no. of entries * size of each entry

$$= \frac{100 \times 10^6}{10^3} \times 4 = 100 \times 10^3 \times 4 = 0.4 \times 10^6 \text{ B}$$

Free space available $= (100 \times 10^6) - (0.4 \times 10^6)$

$$= 99.6 \times 10^6 \text{ B}$$

$\therefore$ 99.6 files A

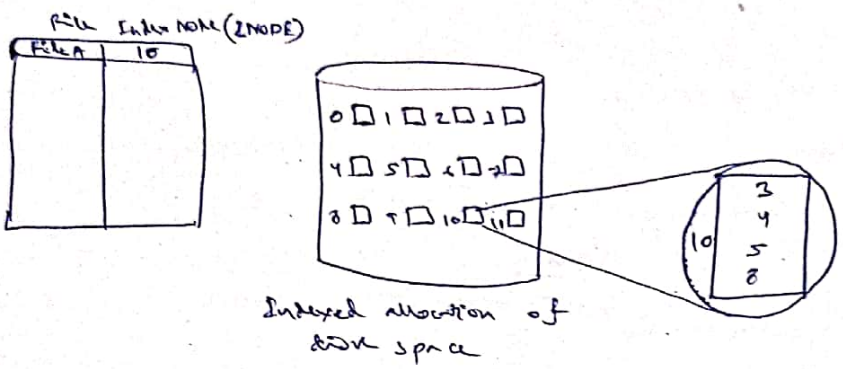
17 - Indexed Allocation :-

→ Linked allocation solves the external fragmentation and size declaration problem of contiguous allocation.
→ In the absence of FAT, linked allocation cannot support efficient direct access.
→ Indexed allocation solves the problem by bringing all the pointers together into one location : the index block

(At any point of time, some files are open, so maintaining information about all the files for all the time is wastage of memory. Only the files we need, those information is needed to be in main memory. So for that reason one block is maintained for every file called index block or inode.)

↓ ✓

Physical block in which particular file is present.



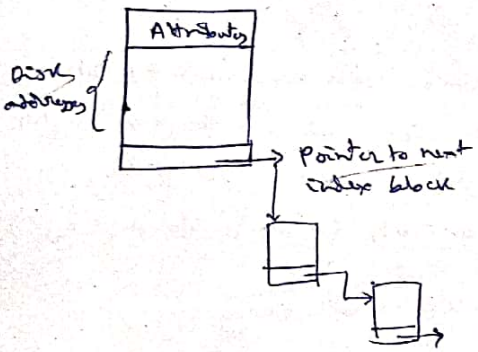
Indexed allocation of disk space

18 – Indexed Allocation -2

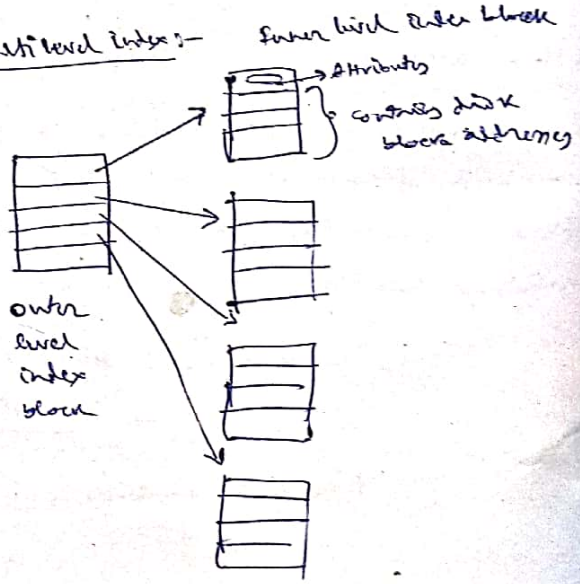
→ If the index block is too small, however it will not be able to hold enough pointers to hold for a large file.

~~→ Index block~~

Linked Schema:- We can link several index blocks.



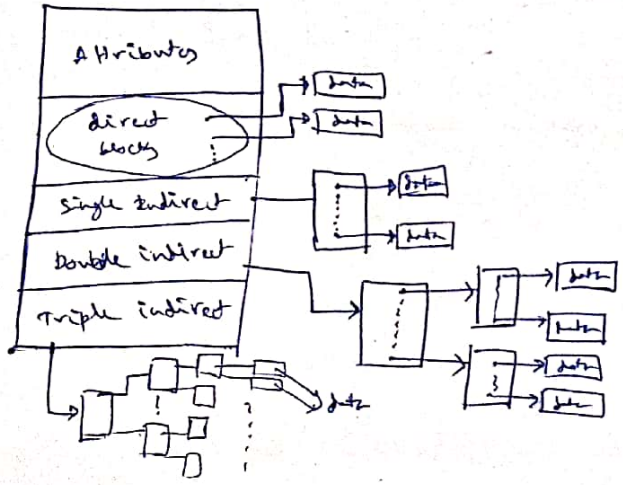
Multilevel Index:-



19 – Indexed Allocation -3

Combined Scheme (UNIX):-

(Inode structure)



GATE CS-2004

A unix style i-node has 10 direct pointers and one single, one double and one triple indirect pointers. Disk block size is 1K-byte, disk block address is 32 bits and 48-bit integers are used. What is the maximum possible file size?

* $(10 + 170 + 170\times170 + 170\times170\times170) * 1024\,B \cong 2^{31}$ (2GB)

(10: direct pointers; single; double; triple; 1024 B: block size)

Disk block size / address pointer size

1 KB ← 48 bits

$= \frac{1024\,B}{6\,B} = 170.66 \approx 170$

↑ no. of addresses per block / pointers per block

20 — GATE 2012 on Indexed Allocation

A file system with 300 GB uses a file descriptor with 8 direct block addresses, one indirect block address and one doubly indirect block address. The size of each disk block is 128 bytes and the size of each disk block address is 8 bytes. The maximum possible file size in the file system is:

* No of address or pointers $= \frac{300\,GB}{128\,B} \frac{128\,B}{8\,B} = \frac{2^7}{2^3} = 2^4 = 16$ data blocks

Direct → 8

Single indirect → 16

Double indirect → 16 × 16

* $(8 + 16 + 16\times16) * 128$

$= (24 + 256) * 2^7$

$= 280 \times 2^7$ Bytes

$= 8(1 + 2 + 2\times16) * 2^7$

$= 2^3 * 35 * 2^7 = 35\,KB$

21 — Free Space Management

Main responsibility of file system :-

→ Allocate blocks to each file & keep track of it.
→ It should also keep track of free space available.

1. Bit vector :-

⇒ The free space list is implemented as a bitmap (or) bit vector.
⇒ Each block is represented by 1 bit.
⇒ If the block is free bit is 1 else it is 0.

2. Linked List (Free list) :-

⇒ Another approach to free space management is to link together the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

22- Disk Scheduling

⇒ File Systems must be accessed in efficient manner, especially with hard drives, which are slowest part of a computer.

⇒ As a computer deals with multiple processes over a period of time, a list of requests to access the disk build up. For efficiency purposes all requests (from all processes) are aggregated together.

⇒ The technique that operating system uses to determine which request to do satisfy next is called disk scheduling.

Seek Time :- Moving the Read/Write head to appropriate cylinder or track is called seek time.

[Many disks → ~~Sys~~ Cylinder, One disk → Track]

∴ Only way for efficiency is to reduce seek time.

23 - FCFS Scheduling

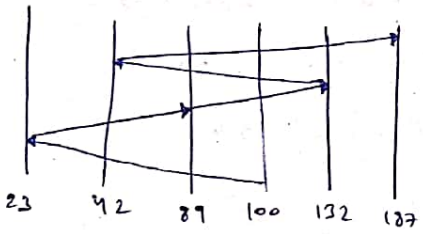
⇒ Simplest. Performs operations in order requested

⇒ No ordering of work queue.

⇒ No starvation: every request is serviced.

Ex- A disk queue with requests for I/O to blocks on cylinders:

23, 89, 132, 42, 187 with disk head initially at 100



Sum of seek time = 77 + 66 + 43 + 90 + 145 = 421 cylinders

24 - SSTF Scheduling (Shortest Seek Time First)

⇒ Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position regardless of direction.

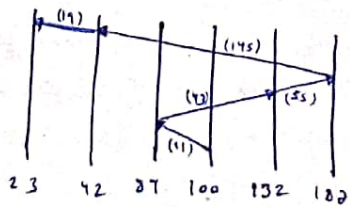
⇒ Reduces total seek time compared to FCFS.

Disadvantages:-

⇒ Starvation is possible.

⇒ Switching directions slows things.

⇒ Not the most optimal.

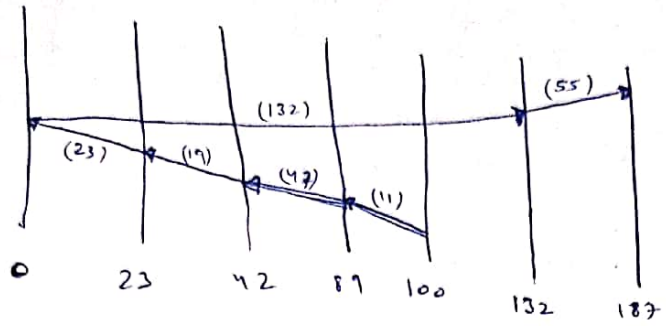


11 + 43 + 55 + 145 + 19 ~~+ 273~~ = 273

25 - Scan, C-Scan

SCAN (Elevator algorithm) :-

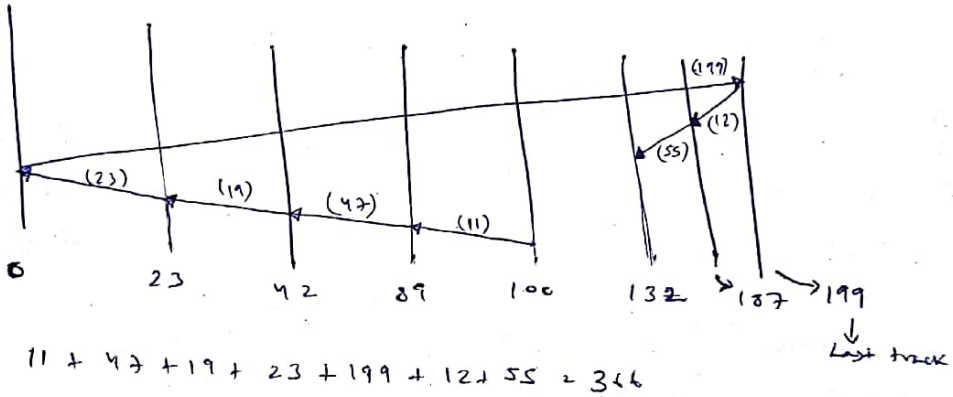
⇒ Go from the outside to inside servicing requests and then back from the inside to the outside servicing requests.



(circular) 11 + 47 + 19 + 23 + 132 + 55 = 287

C - SCAN :-

⇒ Moves inwards servicing requests until it reaches the innermost cylinder, then jumps to the outside cylinder of the disk without servicing any requests.

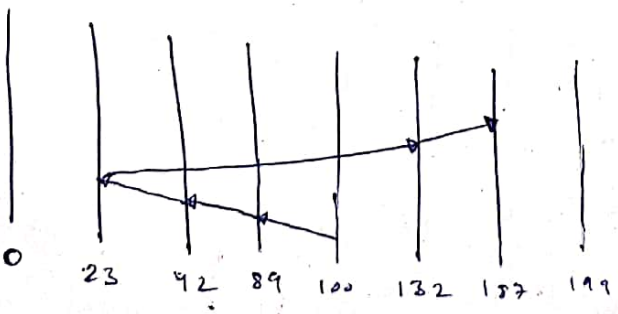


11 + 47 + 19 + 23 + 199 + 12 + 55 = 366

26 - Look

⇒ Like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist.

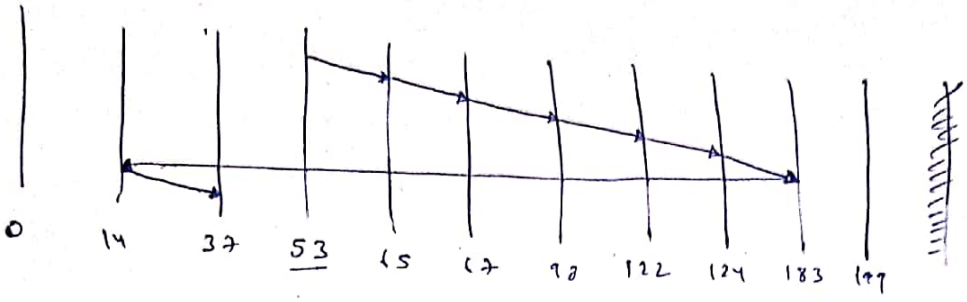
23, 89, 132, 42, 187



11 + 47 + 19 + 109 + 55 = 241

27 - C-Look

98, 183, 37, 122, 14, 124, 65, 67 head starts at 53



- Head → Start → Upto highest reqest → lowest req → remaining request

⇒ In C-Look Scheduling the arm goes only as far as final request in each direction.

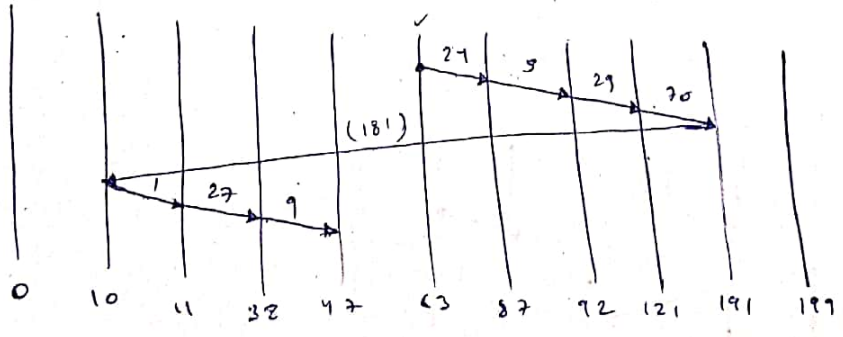
⇒ Then reverse direction immediately without going all the way to the end of the disk.

⇒ When head reaches the other end

- It immediately returns to the lowest cylinder request and without servicing any requests on the return trip.

28 - GATE 2016 Question on C-Look

Consider a disk queue with request for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-Look scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are ~~numbered~~ numbered from 0 to 99. The total head movement (in number of ~~cyl~~ cylinders) incurred while servicing these requests is _____.



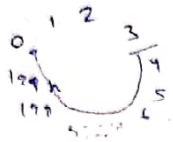
24 + 5 + 29 + 70 + 181 + 1 + 27 + 9 = 346

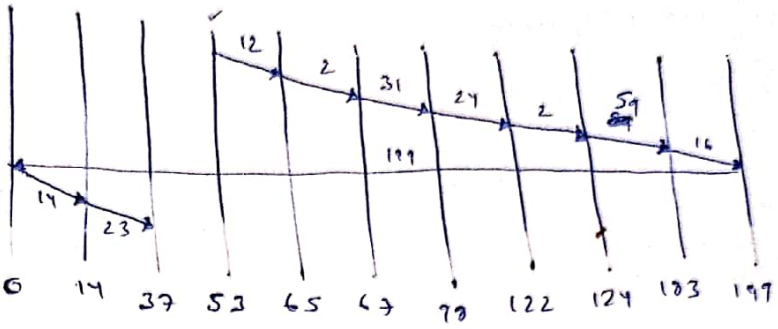
29 - C-SCAN Scheduling

⇒ Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.

⇒ When the head reaches the other end, however it immediately returns to the beginning of the disk without servicing any requests on the return trip.

(Difference between SCAN)





Head starts at 53
(Assume head is moving towards higher cylinder side)

6 14 37 53 65 67 79 122 124 183 199

Total movements = 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 23 =

* OC - Thread and System Calls

01 - System Calls vs Function Calls

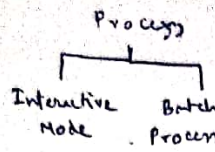
- System Call :⇒ Functions of OS is called by a user process to service its work, is called System Call.
  Ex - Write, Read, Open, Close etc, or using system library function.
- At the time of system call, user mode is changed to super mode (kernal mode).
- Context Switch is done at the time of system call.
- Parameter must be passed along with system calls by -
  i) Registers, ii) Storing it in a block of memory, iii) Stack (like function calls)
  ↓
  Problem → cannot handle too many parameters
- The main difference between System Call & Function Call is -
  If a function ~~of OS~~ is being provided as a part of the process itself then there is no need of context switching, where in the case of system call context switching is ~~there~~ there as ~~no~~ user ~~mode~~ mode is changed to super mode.

02 - Process Control System Calls

→ End, abort
→ Load, execute
→ Create process, terminate process
→ Get process attributes, set process attributes
→ Wait for time
→ Wait event, Signal event
→ Allocate, free memory

03 - File Related System Calls

→ Create file, delete file
→ Open, close
→ read, write, reposition
→ get file attributes, set file attributes



(In C, repositioning → seek)

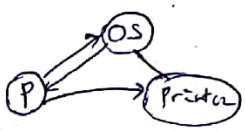
At the time of execution of a process, if nothing else can be executed, it will be running in the foreground. I/O can be done by using keyboard or mouse etc directly.

A large number of processes will be executed in the background. I/O can be only done by using files. Process → user files → OS. So system calls are needed to use these files.

04 - Device ~~Manipulation~~ Related System Calls

Device Manipulation :-

→ request device, release device
→ Read, Write, reposition
→ get device attributes, set ~~Attributes~~ device attributes
→ logically attach or detach devices. (Enable device drivers / disable device drivers)



05 - Information Related System Calls

Information Maintenance :-

→ Get time or date, Set time or date
→ Get system data, Set system data
→ Get process, file or device data
→ Set process, file or device data

06 - Communications Related System Calls

→ Create, delete communication connection
→ send, receive messages
→ Transfer status information
→ Attach or detach remote devices

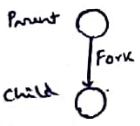
~~Process~~

• Interprocess Communication

DOS + System calls
↓
Windows NT
(New Technology)

07 - Fork System Call

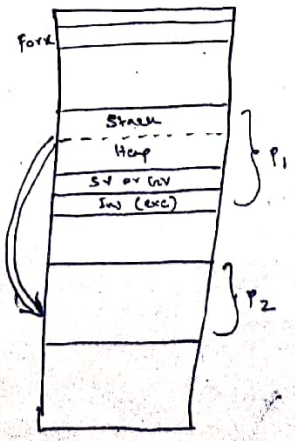
Fork / execve :-



• Problems can be handled by fork that can satisfy many requests at the same time.

Instruction → System call (Fork) → OS
↓
Allocates ~~more~~ space for new child (replication of parent process)

Both process ~~run~~ ~~parallely~~ executes simultaneously



• Context Switch is done because of user mode to system mode. So, this is why we needs threads.

(calling fork → int i = fork();
if (i == 0) { child (zero) }
if (i != 0) { parent }

- fork returns 0 for the child process & non-zero for parent process.

0 → child

pid of child → parent

Shell → command → fork is called → child process with execve ls -l, cat etc

child → exit first

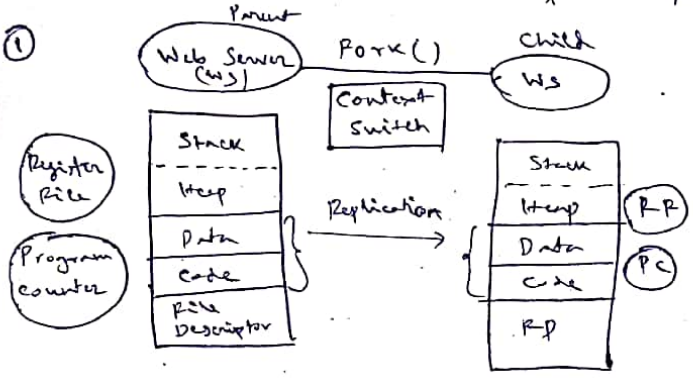
parent → exit last

- The code generally written in such a way that the body of the child will be replaced by some other program & start executing that.
  - execve();
- Parallel execution of same thing using fork/execve is not useful because of context switch due to change of mode (user to system).

08 - Process VS Threads

- Parallelism in case of processes is achieved by multiprogramming, where each process can perform different tasks but the problem is it is very expensive as context switch rate is high due to switching between processes.
- In case of same task by different processes, multiprogramming is not a good approach

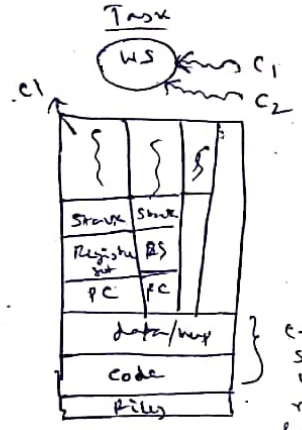
① Parent



Using process

- CPU & memory wise is expensive (;

② ~~Thread~~ Thread :- (user level threads)



- A thread is a sequence of execution. It is also called lightweight process.
- Fork() is a system call, serviced by OS. So multiprogramming is expensive.
- Thread is lightweight & share same piece of memory & child thread is created by user, not OS. So it is cheap & space needed is also very less compared to Fork.
- Switching from one thread to another need not to call OS, so it is inexpensive.

Thread State :-

New, Ready, Running, Blocked, Terminated

Process (Fork)	Threads (user level)
→ System calls are involved.	→ No system call.
→ Context Switching is required.	→ Register set Switch (including PC).
→ Different copies of code & data.	→ Same copy of code & data.
→ One process cannot change the content of the stack of another process.	→ One thread can change the content of the stack of other threads.

- Collection of all threads is known like a single process to OS, so due to system call if the resource is not available, OS will block all threads instead of only that one.

↓

Disadvantage of thread

↓

Solved by kernel level thread but for this again OS must be called (context switch a thread is created (kernel)).

→ But a disadvantage as designed to do same work

- All the thread also gets same time as a single process as there are user level threads.

09 – User level vs Kernel level Threads

Disadv of user level threads :-

⇒ Blocking System Call will block the whole task.

⇒ Unfair scheduling.

↓ Solution

Let the kernel know that these are threads.

↓ Solution

Let the kernel know that only one thread is getting blocked & other threads can be scheduled.

Kernel level threads

A thread wants to make another thread → System call → Kernel → New thread created.

∴ so kernel can know total no. of thread created & can keep track of them.

Disadv :- ⇒ Expensive compared to user level thread. (Less expensive than creation of a process)

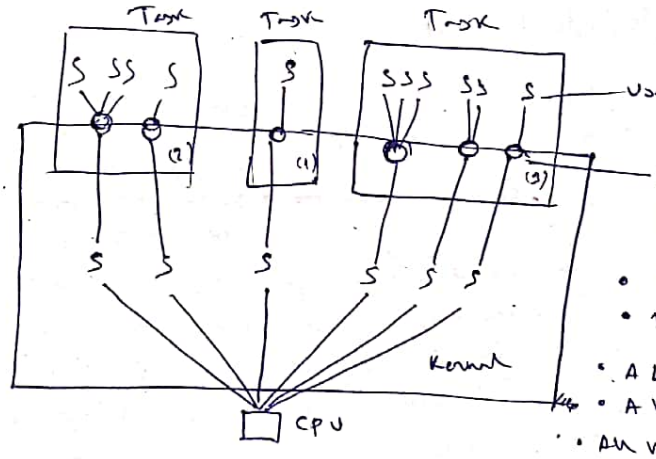
⇒ Switching a thread requires system call. (only registers are have to be changed)

↳ P > KLT > ULT

- user & kernel level threads are combined & called hybrid level threads. ↳ (Better than multiprogramming of process)

10 – Hybrid Threads

Solaris 2 : (version of Unix)



- For every ULT → A LWP
- For every LWP → A KLT
- A LWP → many ULT
- A KLT → One LWP
- All KLT gets time from the CPU (i.e Scheduled by CPU)

Switching

- One ULT to another ULT → No System Call → Faster
- One LWP to another LWP → Some Context Switching & System call
- One KLT to another KLT → Context switching & System Call

→ Slower

- So if one thread got blocked, our entire task will not get blocked.