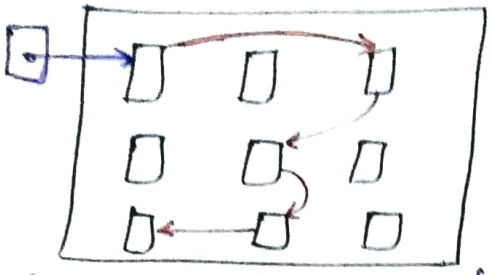
② Linked List [Free List]



Point one free block from another free blocks and Contains the location of first free block in disk.

Disk Scheduling [Detailing in COA]

Process take time in doing
① CPU Utilization
② I/O task.

Process generally do I/O with hard disk.

So if there are large no of process doing I/O and hard disk also works very slow. So all the process gets collected in hard disk.

Then, what will the order of request of process to get access for I/O.

Disk scheduling decide which request is going to execute next.

→ File systems must be accessed in an efficient manner, especially with hard drives, which are slowest part of the computer.

→ As a computer deals with multiple processes over a period of time, a list of request to access the disk build up. For efficiency purpose all request are aggregated together.

→ The technique used by OS to determine which request to satisfy next called disk scheduling.

FCFS Scheduling

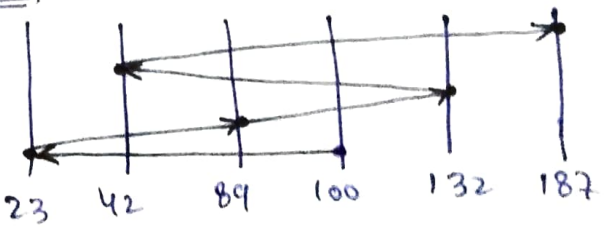
→ Simplest perform operations.
→ No ordering of work queue.
→ No starvation: every request is serviced.

Ex: A disk queue with request for I/O to blocks on cylinders
23, 89, 132, 42, 187.
Initially the disk head at 100

(disk head ↓ Read/write head.)

No. of cylinder crossed?

Soln.



No. of cylinder we crossed =
$(100-23)+(89-23)+(132-89)$
$+(132-42)+(187-42)$
$= 421$

Seek time & no. of cylinder cross

Note → Just go through it we will read the same in COA in details.

SSTF Scheduling

Like SJF, it is Shortest Seek time first.

→ Select the disk I/O request that requires the least movement of the disk arm from it's current position regardless of direction.

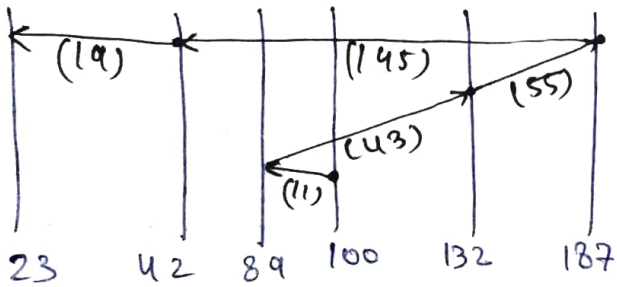
→ Reduce the seek time as compare to FCFS.

Disadvantage → ① Starvation is possible.

② Switching direction make things slow down

③ Not the most optimal ways.

Consider same example as FCFS.



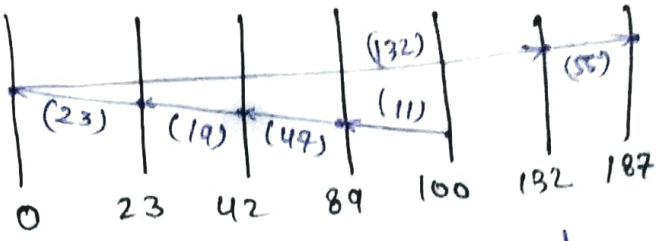
, 11 + 43 + 55 + 145 + 19

= 273

→ No. of cylinder crossed = 273 better as compare to FCFS.

Scan, C-scan [Elevator Algo.]

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



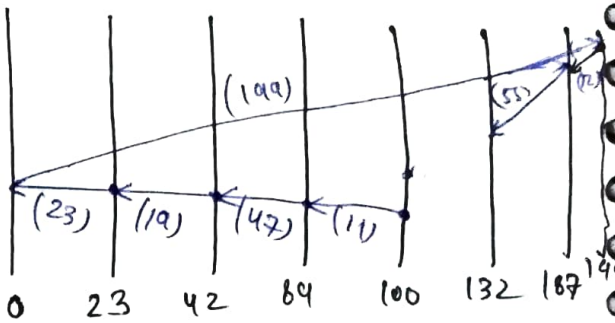
Moving in one direction and service the request come across the path.

Even though '0' is not requested but we go for (23 → 0).

No. of cylinder crossed =

11 + 47 + 19 + 23 + 132 + 55 = 287

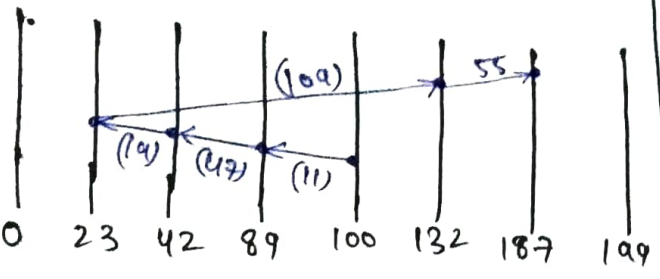
C-scan → Moves inwards servicing requests until it reaches the innermost cylinder, then jumps to the outermost cylinder of disk without servicing any requests



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

= 366

Look: Like scan but stop moving inwards (or outwards) when no more requests are in the directions.

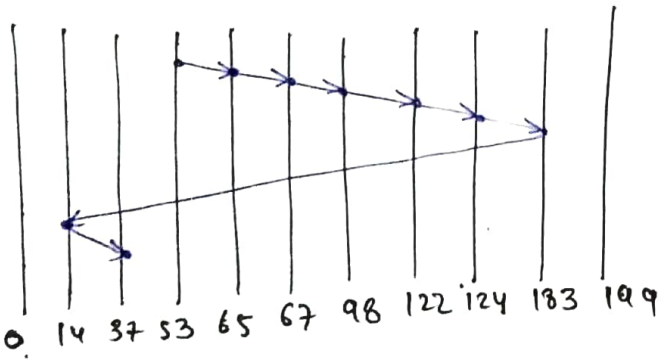


11 + 47 + 19 + 109 + 55 = 241

C-look & Circular look

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

Head pointer at 530



12 + 2 + 31 + 24 + 2 + 59 + 169 + 23 = 322.

→ The arms goes only as far as final request in each direction.

→ The reverse direction immediately without going the way to end of disk

→ When head reaches the outer end, it immediately returns to lowest cylinder requests and without serving any request on the return trip.

Note → Direction of movement will be given in the question otherwise assume the

→ OS is interface b/w user & hardware, also provides system calls, works as resource allocator, manages m/m, process & security.

→ PCB attributes are → Process Id, Program Counter, Process state, Priority, General Purpose register, List of open files/devices, Protection.

→ If one executable code tries to access other's space called segmentation fault.

→ Multiprogramming with preemption : Multitasking

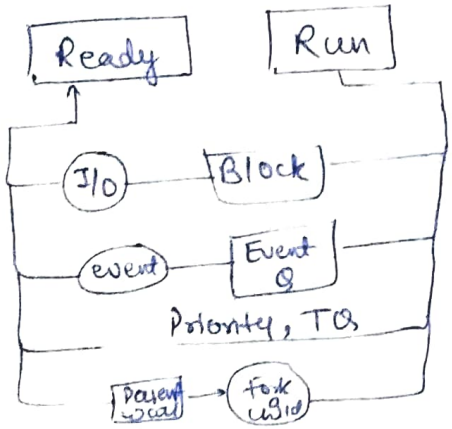
→ States of Process : New -- Ready -- Run -- Terminate -- Block/Wait -- Suspend ready -- Suspend block/wait

→ Long term scheduler do swap in/out between secondary m/m to New Queues

→ Short term scheduler do context switching work as dispatcher.

→ Medium term scheduler decided about suspend wait / block.

→ After Suspension process stay in secondary memory.



Consider a system with 'N' CPU's & 'M' process than,

	min	max
ready	0	M
run	0	N
block	0	M

Step 1 : find completion Time

Step 2 : TAT = (Completion - Arrival)

Step 3 : WT = (TAT - BT)

Step 4 : Response Time → process gets scheduled first time.

① FCFS : Non-preemptive, criteria is arrival time.

→ In case of non preemptive Algo. 'WT & response time is same'.

Convoy effect is drawback of both FCFS and SJF

T(n) = O(n) DS = Queues

② SJF : Non preemtive, shortest job first.

Throughput → No. of process / Total burst time

BT prediction Technique

Static
① Process size
② Process Type

Dynamic
① Simple avg.
② Exponential avg or ageing.

exp. avg → $\tau_{n+1} = \alpha \cdot t_n + (1-\alpha)\tau_{n-1}$

③ SRTF → Preemptive and selection criteria is BT.

④ Round Robin → Queue is used to make process sequence.

⑤ HRRN → Better as compare to SJF.

→ At the time when all process gets inside start calculate Response ratio $\left(\frac{W+BT}{BT}\right)$ for each remaining process. Starts from shortest job available.

→ Using Multilevel-feedback queue, we avoid starvation.

→ ~~Process Synchronization~~

User level thread → Implemented by users, doesn't recognized by OS, implementation is easy, less context switching time, no hardware support, all threads are independent of one thread gets block complete process will get blocked.

Kernel level thread → Implemented by OS and complicated, recognized by OS, hardware support needed, more context switch time, independent threads, if one thread blocked another thread can continue.

→ Each thread shares code section, data section and OS resources but don't share its program counter, registers set & stack self. Also shares message queue and file table.

Advantages of thread over process → Responsiveness, fast context switching, effective utilization of (MP), resource sharing.

→ Types of System Call: Process control, file Management, Device management, Information maintenance, comm^n.

Spooling → Simultaneous peripheral operation online.

Application: ① Used by I/O device like like keyboard, printer etc

② A batch OS used to maintain a queue of ready → sent jobs

③ E-mail is deliver by Mail transfer agent (MTA) to a temporary storage where it waits to be picked up by MA (Mail User agent).

Inter Process Comm^n → IPC may be either independent or cooperating. In independent one process can't effect other's environment.

Advantage of process Co-operation is → sharing info, computation speed up, modularity and convinient to work on same task at a same time.

Two models of IPC → (1) Shared m/m → All the process shared the m/m space.

(2) Message passing → Communication takes place by message exchanged mechanism, requires Kernel support.

→ In Batch OS we group all the jobs with same requirement

→ In timesharing OS the main aim is to reduce response time instead of Increase in CPU utilization. (Round Robin)

→ Kernel mode : Previlged mode ; User mode → Non-privited

→ User level threads are transparent to kernel level.

Hardware Interrupt : Generated by external device or hardware, don't increment Program Counter, invoked with some external device, low priority than SI, asynchronous event. Types – Maskable or Non maskable.

Software Interrupt : Generated by Internal System of computer, increment program counter, invoked by INT instruction, higher prority among all interupts, it is away to trigger system call or contact with kernel, synchronous, [Normal Interupts & Exceptions] are its types.

→ In both Synchronous | asynchronous ISR invoked after completion of I/O.

→ fork() → return 0 for child
→ return(+ve) for parent

Imp. Qus → 2, 5, 10, 12, 13, 15, 16, 17, 22, 23, 25, 27, 29, 30, 36, 37, 40, 43, 44, 47, 51, 56, 61, 63.

Process Synchronization :-

Need of Synchronization to maintain consistency in database.
If more than one process accessing same variable then final result depends upon order of execution called race condition.

Each synchronization must satisfy : Mutual Exclusion, Progress, Bounded wait, Architetural Neutrality.

Progress means one process is not stoping other process.

Busy waiting means process is inside CPU but stuck in ∞ loop

Bounded waiting means process will definitely get CPU after some time.

Busy waiting soln

① Lock variable : Doesn't provide mutual exclusion.

② TSL : Provided mutual exclusion and progress only.

③ spin locked → CPU pointing P_2 but P_1 is inside critical section, TSL is deadlock free. but not starvation free.

④ Turn variable : follows every propety except progress. If we include interest variable with turn the progress is guaranted but bounded waiting is not guaranteed.

→ Peterson : It supports all the four propeties. Process that access turn variable first will get CS first.

Without Busy waiting : To avoid busy waiting we use wake_up() and sleep() mechanism.

Ex→ Producer Consumer problem.

for producer if [Count == N] then sleep(), for consumer if [Count == 0] then sleep().

→ It can also not able to avoid deadlock. for this we have semaphores. Semaphore need support from OS. we can't access it in user mode.

① Counting semaphore [for n-process]

Symbol used

Down, P, Wait (decrement) ; UP, V, signal (increment).

② Binary Semaphore : It has only two value 0, 1 and allow only two procss at a time.

Barrier → A point in a program that every variable needs to pass.

→ Swap space is a part of virtual m/m that resides in Disk. Appn which are not in use or less used can be kept in swap space.

→ B_+ Tree supports range query but hashing doesn't.

Semaphores : PYQ : Review all the PYQs.

→ Producer first do P(size) then V(count) but consume do P(count) then V(size).

Deadlock: Starvation is long waiting but deadlock is infinite waiting. (11)

Necessary Condition:- Mutual Exclusion, Hold & wait, No preemption, circular wait.

Deadlock Handling:- Ignorance, prevention, avoidance, detection & recovery.

In prevention we try to prevent the anyone of 4 conditions.

Mutual exclusion → Spool everything.
Hold & wait → Request all resource initially
No preemption → Take resource away.
Circular wait → Order resource numerically

} Approach

→ Only circular wait can be implemented practically.

Bankers Algo: Safe & Unsafe State

Resource Allocation Graph → Single & Multiple instances.

In single instance cycle is sufficient but in multiple instance cycle is necessary condition.

Deadlock Recovery: Resource preemption, Rollback, kill the victim kill all process.

PYQ → Review all the Questions.

Memory Management: Say CPU utilization is $(1-p)$ if p → I/O utilization then efficiency = $1-p^k$

where k → degree of multiprogramg. $(p<1)$

Object code contains → header, code, data segment, relocation info., symbol-table, debugging info.

Linker resolved the unresolved symbol and give relocatable addr.
Loader loads the program in main m/m and gives absolute address.
After Compiling we get code with relocatable add., after linking we get relocatable add. for combine modules. Dynamic linking require OS support.

Fixed Partitioning → Both Internal & External fragmentation
Variable Partioning → Avoid Internal fragmentation only.

for external fragmentation → Compaction, Bit mapping, Linked List
We have also have overlays concept means virtual m/m implementation by user.

LAS/VAS : [No of Pages | Pg size] PAS : [No of frames | Pg size]

Page table size : (No. of pages) * (No. of frame bits) → convert both into byte first to

Page table entry : Caching disabled, modified/dirty bit, referenced, protection, present/absent

Multi level paging → Revise marked qution in table of notes

Optimal Page size = $\sqrt{2se}$ or $\sqrt{2se/n}$ e → entry size (pg.S → (VAS) bits

TLB : Translation lookaside buffer :

EMAT = p(t+m) + (1-p)[t + m + nm]

for n-level page tables, p → hit ratio,

t → TLB time, m → ma times

→ If large no. of page fault occurs called thrashing.

EMAT = p(service time + m/ma cycles) + (1-p)(m/ma cycles)

p → page fault rate

Inverted page table → from copy.

frame allocation methods :- Equal allocation, weighted allocation, Dynamic allocation

Page Replacement : Load the page if required called as demand paging. (Local or Global Page replacement).

LRU → Less used in past Optimal : Less used in future

on increase frames if page fault ↑ called Belady's anomaly.

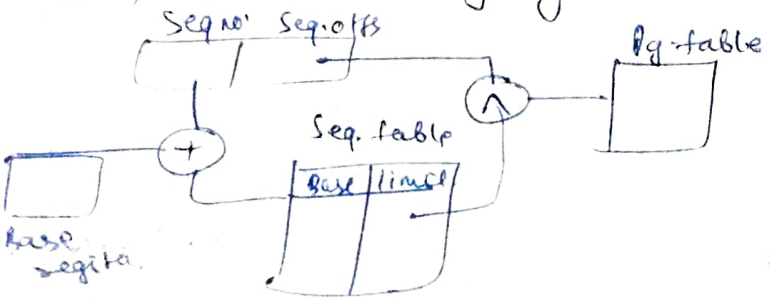
Only FIFO and Random page Replacement follow it.

Algo who doesn't follow Belady's called Stack Algo.

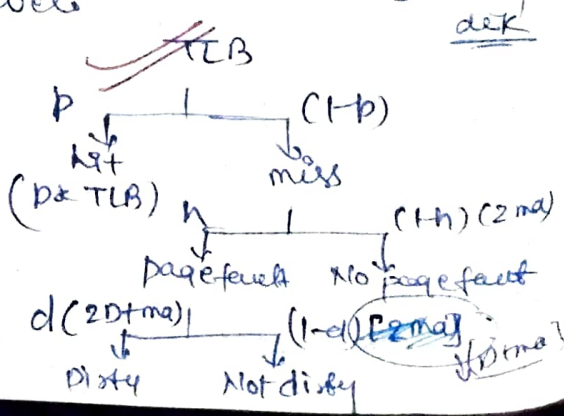
Pages(m) ⊆ Pages(m+1)

Working set Algorithm → Dynamic approach

Segmentation : Paging at user level.



D → Time for secondary dek



File System : (info) → (files) → (Directory) → (file system)

file is an ADT with attributes like name, identifier, type, location, size, protection, time & date.

file operations → create, write / Read, Repositioning, Deleting, Truncate

As a process open a file it creates a file table with info of file pointer, access rights, file name.

OS also maintains an open table for all process with info of file name, location, open counter.

If open counter = 0 it means no process is reading that file means delete the entry from open table.

We can access the file sequentially, direct, indexed [B/B+]

for direct access we need info of no of records per block, no. of records per file, block size.

Directory Str. If we have more than 1 OS in system so we have to maintaintain more than 1 directory for each.

→ Each directory have info corresponds to particular partition (OS).

We can create, rename, delete, traverse, search and list them.

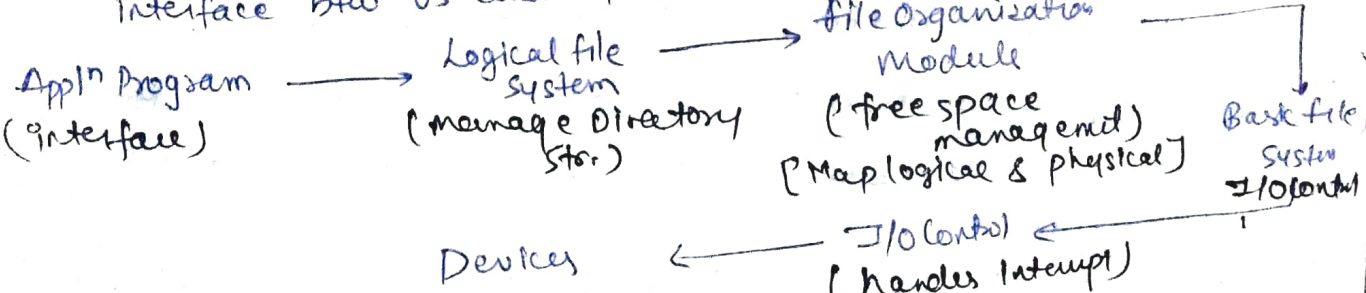
→ 2 level, tree directories

Single level directory → Simple & creating & searching, naming & security is difficult.

Two level : Master at top level, two user can have same file name, security is easy, OS take care of access rights, separate part for system files.

Acyclic graph → two directories are allowed to access same file but problem is if one deleted it then. for this we have to maintain a counter.

File System : Deals with structure, allocation in disk, Recovery, Interface btw OS and m/m.



MBR : (Master Boot Record)

It loads the booting information to the RAM first & it contains info about all the partitions avaible for various OS.

Two types of Data str. → ① On disk ② In memory.

On disk: ① Boot Control Block: It contains the booting info.

② Volume Control Block: Details of no. of blocks per partition, Basically how system uses the complete partition block.

③ File Control block: It contains permission, date, owner, size, data blocks & pointer to file data block.

In memory: Mounting means creating new partition.

In m/m mount table: info. of mounted volumes

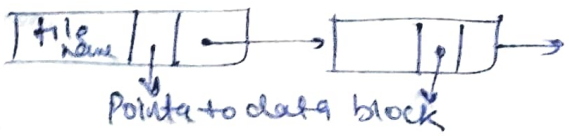
In m/m directory str. cache: info of recently accessed directory

per process open file: pointer to appropriate entry in wide system file.

wide-system open file table: info of all files open by OS.

Directory Implementation: Selection of implentation effects the performance and reliability of file system.

① Liner List:



② Hash table: (key, value) pairs

Allocation methods

① Contiguous allocation → Both Internal & External frag. is possible but we can remove only ext. fragmentation here

② Linked List Allocation: Random search is slow, no ext. fragments, less disk wastage.

③ File allocation table: Random search is fast, no space for pointers, but size may be large.

④ Indexed allocation: We collect all the pointer in a singe node (inode) and avoid the large size of tables

(i) Linked scheme: We can link several index block

(ii) Multilevel index: Outer level block contain info. of cell Inodes

Combined scheme : Attribute - single / direct / double / triple indirect

Managing free spaces : first responsibility of file system is allocation & packing.

Second is recovery of free blocks.

① **Bit Vector** : 1 → filled 0 → freed.

② **Linked List** : Point one to other free nodes.

Disk scheduling: Used to mange I/O operations done on HD.

① **FCFS** : No starvation, No queue, simplest.
No. of cylinder cross & seek time.

② **SSTF** : Shortest seek time first → Starvation is possible, Changing direction take more time. Better as compare to FCFS. [Nearest Cylinder next] → Best Throughput.

③ **C-Scan**: [Elevator Algo.] : ~~Move inwards upto min and then move outwards upto max.~~

④ ~~Scan: Move inwards and then outwards, inwards upto min~~

⑤ ~~Look, C-look~~ : look same as ~~scan but~~ stop inwards ~~and outwards~~ upto limits

→ for C-look, see the copy notes.

PYQ : ① If process CPU follows Round Robin and BT is very high, then waiting time will get increases.

② In HRRN we support process with lower BT and limits the waiting time of higher BT.

③ We use RR in time shared algorithms.

④ System calls are invoked by software interrupts.

⑤ A multi user & multi-processing OS can be implement on hardware that supports previleged & non-previliged.

⑥ To change the mode from previliged to non-previleged a non-previliged instruction that doesn't generate interrupt needed.

⑦ SJF gives the max. throughput.

⑧ Gang Scheduling → Thread, Rate-Monotonic Scheduling → Real-time CPU, fair-share → Guaranteed.

→ In RR if TQ > max(BT) then it degenerate to FCFS.

→ In both synchronous and asynchronous ISR invoked after I/O completion.

In synchronous I/O after invocation of ISR CPU will place the process from block state to ready state.

→ Round Robin is better than FCFS in terms of response time.

→ Switching between user to kernel takes less time than context switching.

→ User level threads are not scheduled by kernels

→ Thread of a process shares both heap & global variables.

→ In Process Synchronization look at the only IO questions as— 40, 39, 38, 36, 35, 32, 31, 30, 27, 26

→ The link-load and go scheme takes more space as compare to link and go space.

→ A link-editor is a program that acts as a link between compiler and user program. (wrong)

→ The amount of virtual m/m limited by the secondary storage.

→ Monitor's doesn't use indivisible machine instruction to resolve Critical Section issues.

→ The LRU page replacement policy may cause hashing for some type of programs.

→ 2 pass assembler: Pass 1 ↳ Define symbol and literals and use in symbol & literal tables.
Keep track of location counter.
Process pseudo-operations.

Pass 1 → info. collected
Pass 2 → code generation

Pass 2 → Generate object code, generate data for literals and look for values of symbols.

→ In segmentation, each segment table have its own page table because of its large size.

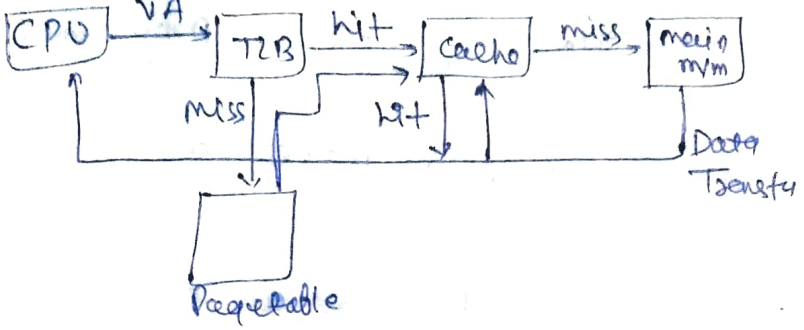
→ Thrashing implies excessive page I/O.

→ Dirty bit for a page avoids unnecessary writes on paging device.

→ In belady's anomaly, it is not happening always that ↑ frames → ↑ page faults.

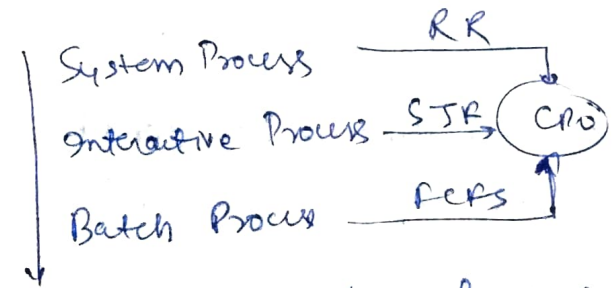
→ Instruction opcode is not a form of m/m.

→ We have need for multi-level paging to avoid the large m/m overhead of maintaining page tables.



TLB Hit * Cache hit
+ (TLB Hit * Cache miss)
+ (TLB Miss * Cache hit
+ (TLB Miss * Cache mis

→ The min no. of page frames that must be allocated to a running process is determined by the instruction set architecture.



we can use the same algo. for each queue or we can use different one too.

→ Starvation is problem here.

To avoid stevation we use Multilevel feedback Queues we upgrade process to high priority queue from low priority queue using feedback.

→ Peterson's and dekker's algo satisfy all 3 problem's solution for synchronisation.

→ TLB Hit [TLB access time + Cache hit * Cache access + Cache miss [Cache + ma]]
+
TLB miss [TLB access + n* page table access + Cache hit * Cache access + cache miss [Cache + ma]]

→ Each entry / page / frame of main mm contains element / word

No. of words in a frame = $\frac{\text{Size of frame}}{\text{word size}}$

→ Dirty page also contributes in page fault

→ On receiving an interrupt from I/O device the CPU branches off the ISR after completion of current Instn

→ Data transfer between memory and I/O is faster in case of Interrupt driven I/O as compare to programmed I/O.

→ The root directory of a disk should be placed at fixed location on system disk.

→ I/O redirection means it can be use an existing file as input file for a program

→ When an Interrupt occur OS may change the state of interrupted process to 'blocked' and schedule another process.

→ Enable should be 1, to process interrupts for CPU and CPU also check interrupts before executing new instruction.

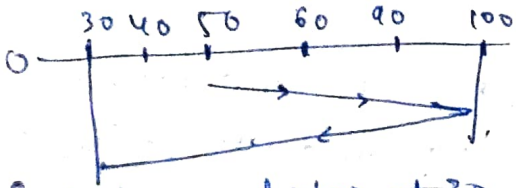
→ Larger block in fixed allocation index increases throughput but reduces disk utilization.

→ ln -s f1, f2 means replace f2 by f1 in directory.

→ Maximum cardinality of request set so that head changes it direction : for this use $new = (2 * old) + 1$

(Elevator)

Scan → follow the direction reach upto either max. or min and then come back



go to 100 and stop at 30 don't need to go for 0.

C-SCAN ① Move in same direction to max or min cyclicly.

② Come back to min or max & again start traversing the cylinder

→ Preemption with time Quantum allow the process to improve interactiveness.

→ Commands in UNIX

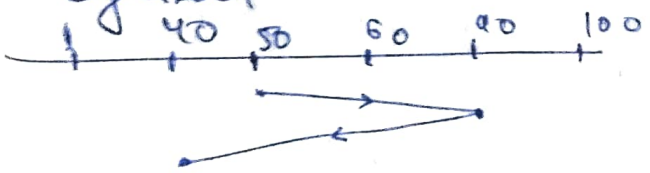
1. Nice : To set priority of process (NI)
2. ls : It lists all the file system directories.
3. ps : process status inside the system.
4. grep : It search in a file for a particular pattern of char.

→ In fork() → Child, parent can be execute in any order.

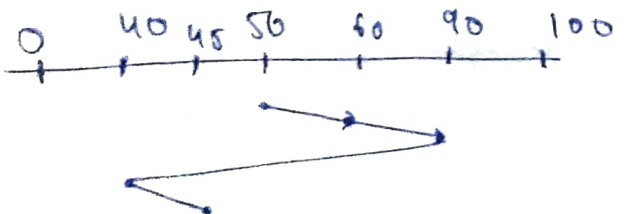
⇒ Priority Inversion may occur when CPU is running priority based preemtive scheduling algo. and program uses any busy waiting soln [Peterson, TSL, Lock, swap etc]

→ Related Kernel level threads can be scheduled on different processors in multi processing environment.

Look → Same as Scan but dont need to go for terminals cylinder



C Look →



Same as C-scan but no need to include terminal cylinder

→ Paging suffers internal and segmentation suffers external fragmentation.

→ Process and Kernel thread can run in parallel in Multiprocessing system.

→ Reading of clock can be done in user mode.

→ Accessing I/O device need previliged instruction.

→ In dining philosopher each philosopher picks up the left fork first then right fork.

```
void Philosopher() {
while(true){
 Think();
  take_fork(i) ; take_fork((i+1)%N);
   Eat();
    put_fork(i) ; put_fork((i+1)%N);
 }
```

- Kernel stack can be used to store context of a process not the user program functions & stack by user stack)

→ User threads are fast bcoz it doesn't involves kernel

→ User threads are light bcoz of no system calls

→ Kernel is not aware of blocked & runnable user threads

→ In kernel mode we can change mapping from virtual to main m/m, disable the interrupt, Mask and unmask interrupt

→ No of entries in inverted page table = no. of frames.

→ Overlays are not used to increase main m/m size.

→ Aging is used to resolve starvation problem.

→ By increasing no. of buffers in Producer & Consumer problem throughput ↑, chance of deadlock ↑, implementation complexity ↑.

→ Race Condition : no mutual exclusion, or output depends on sequence of operations.

→ Dispatcher is only responsible for switching threads, priorities are decided by scheduler according to Algorithm.

→ To create a good solution for mutual exclusion problem, there should be no assumption about number of CPU and its speed.

→ VAS can be greater or smaller than the physical memory.

→ Threads have separate execution states.

→ On fork() both child and parent gets different PID and child gets the exactly same file descriptor table opened by parent.

→ A system call always results in context switching because it needs OS in kernel mode.

→ If demanded page string have loops then optimal behave as MRU and if reverse strings demanded then optimal behave as FIFO and LRU.

→ Remeber the concept of MRU.

n- fork() statements will have $2^n - 1$ child.

→ Paging may lead to internal fragmentation.

→ for deletion in B-tree we have two techniques, either borrow from seiblings or coalesce

→ Best throughput : SJF
min. response time : Round Robin
min waiting time : SRTF

Turn Variable

```
while (turn != 0);
   CS
turn = 1;
Non CS
```

```
while (turn != 1);
   CS
turn = 0;
Non CS;
```

Lock Variable

```
while (Lock != 0);
Lock = 1;
   CS
Lock = 0;
```

Dirty bit : Right - back policy
R/W : Page protection
Reference : Page replacement policy
Valid : Page initialization

Locality of reference implies that the page references being made by a process is likely to be a page used in last few pages

→ Threshing reduces page I/O.

→ Virtual m/m have no relation with context switching overheads

→ In FIFO sometime page fault ↑ with ↑ frame not always

→ If page table size > Page size means large m/m overhead in maintaining page tables

→ EMAT = $t_n [t_a + C_n[C_a] + C_m[C_a + m_a]]$
$+ t_m [t_a + (n \times m_a) + C_n[C_a] + C_m[C_a + m_a]]$

→ Shortest seek time first gives best throughput

→ Question of max. possible file size first find.

$$n = \frac{\text{block size (Bytes}}{\text{(address size) Bytes}}$$

then,

$[(\text{no. of direct}) + (n \times \text{single}) + (n^2 \times \text{double}) + (n^3 \times \text{triple})] * \text{disk block size}$

→ Sometimes value of n is directly given as each block contain 128 blocks then, $n = 128$