1) Process Management and Synchronization

1.1) Process Management

→ A process is an instance of a program running in a computer.

→ In UNIX and some other OS, a process is started when a program is initiated.

→ A program it self is not a process but is a file containing a list of instructions stored on disks.

→ A program becomes a process when an executable file is loaded into the memory and executed.

→ When a program is loaded into the memory and it becomes a process it can be divided into four sections.

- ↳ Stack : Contains temporary data such as method / function parameters return address and local variables.
- ↳ Heap : Dynamically allocated memory to a process during run-time.
- ↳ Text : Includes current activity represented by the value of program counter & the contents of processor's registers.
- ↳ Data : This section contains the global and static variable.

1.1.1 Process States:

- ↳ New : The process is being created.
- ↳ READY : The process is waiting to be assigned a processor
- ↳ TERMINATED : The process has finished execution
- ↳ RUNNING : Instruction are being executed
- ↳ WAITING : Process is waiting for some events to occur.

last

1.1.2 Process Control Block (PCB)

→ PCB is a data structure in OS kernel containing the information needed to manage the scheduling a particular process.

→ while creating a process, the OS performs several operations.

- To identify process, it assigns a process identification number to each processes.
- Since OS supports multi-programming, it needs to keep track of all the processes. For this, PCB keeps track of processes's execution status.
- Each block of memory contains information about the process State, Program counter, Stack pointer, Status of opened files, scheduling algorithms etc. All these information is required and must be saved when the process is switched from one State to another. When the process changes states, the OS must update info in the processes's PCB.

Information contained by PCB.

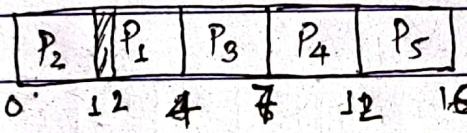
→ Naming the process
→ State of process
→ Resources allocated to process
→ Memory allocated to process
→ Scheduling information
→ I/O devices associated with processes.

Himal
Date
Page

⇒ FCFS CPU Scheduling (non-preemptive)

Process	Arrival time	Burst time
P_1	2	2
P_2	0	1
P_3	2	3
P_4	3	5
P_5	4	4

Gantt chart



Preemptive: If a process is running in the CPU then that process can be forcefully removed from the CPU & CPU can be allocated to some another process.

Turnaround time = Completion time - Arrival

Waiting time = ~~Burst time~~ T.A. - B.T.

$TA\ P_1 = 4-2 = 2$ $\quad WT\ P_1 = 2-2 = 0$

$TA\ P_2 = 1-0 = 1$ $\quad WT\ P_2 = 1-1 = 0$

$TA\ P_3 = 7-2 = 5$ $\quad WT\ P_3 = 5-3 = 2$

$TA\ P_4 = 12-3 = 9$ $\quad WT\ P_4 = 9-5 = 4$

$TA\ P_5 = 16-4 = 12$ $\quad WT\ P_5 = 12-4 = 8$

Final
Date
Page

⇒ Shortest job first / Shortest job next

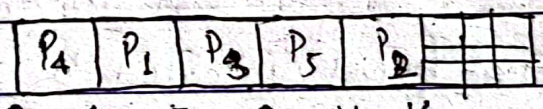
→ Pre-emptive (SRTF)

→ Non-preemptive

Processes	AT	BT
P_1	2	1
P_2	1	5
P_3	4	1
P_4	0	6
P_5	2	3

Ready queue
P_1 P_2 P_3 P_4 P_5

Gantt chart:



TA = Completion time − Arrival Time

WT = TA − B.T.

TA P_1	=	7 − 2	= 5	WT P_1	= 5 − 1 = 4
TA P_2	=	16 − 1	= 15	WT P_2	= 15 − 5 = 10
TA P_3	=	8 − 4	= 4	WT P_3	= 4 − 1 = 3
TA P_4	=	6 − 0	= 6	WT P_4	= 6 − 6 = 0
TA P_5	=	11 − 2	= 9	WT P_5	= 9 − 3 = 6

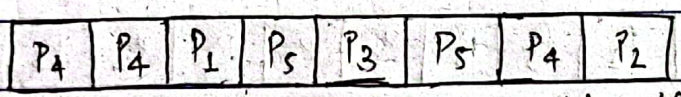
→ SJF (~~non~~ preemptive) ⇒ SRTF

Processes	AT	BT
P_1	2	1
P_2	1	5
P_3	4	1
P_4	0	6 8 4
P_5	2	3 2

Ready Queue

~~P_4~~, P_2, ~~P_1~~, ~~P_5~~, ~~P_3~~

Gantt chart



TA P_1 = 3 - 2 = 1 WT P_1 = 1 - 1 = 0

TA P_2 = 16 - 1 = 15 WT P_2 = 15 - 5 = 10

TA P_3 = 5 - 4 = 1 WT P_3 = 1 - 1 = 0

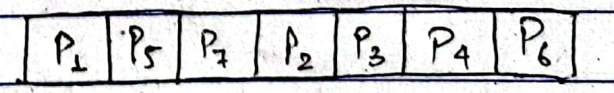
TA P_4 = 11 - 0 = 11 WT P_4 = 11 - 6 = 5

TA P_5 = 7 - 2 = 5 WT P_5 = 5 - 3 = 2

→ Priority Scheduling (Non-preemptive)

Processes	AT	BT	Priority
P_1	0	8	3
P_2	1	2	4
P_3	3	4	4
P_4	4	1	5
P_5	5	6	2
P_6	6	5	6
P_7	10	1	1

Gantt chart



~~P_1~~, ~~P_2~~, ~~P_3~~, ~~P_4~~, ~~P_5~~, P_6, ~~P_7~~

Priority Scheduling (Preemptive)

Processes	AT	BT	Priority
P_1	0	8 ~~3~~	3
P_2	1	2	4
P_3	3	4	4
P_4	4	1	5
P_5	5	~~6~~ ~~0~~ ~~8~~ 1	2
P_6	6	5	6
P_7	10	1	1

Ready: P_1 P_2 P_3 P_4 ~~P_5~~ P_6 ~~P_7~~

P_1	P_1	P_1	P_1	P_5	P_5	P_7	P_5	P_1	P_2	P_3	P_6
0	1	3	4	5	6	10	11	12	15	17	21	26

Round Robin Scheduling: (Pre-emptive) TQ = 3 (Time Quantum)

Processes	AT	BT
✓ P_1	0	~~8~~ ~~5~~ 2
✓ P_2	5	2
✓ P_3	1	~~7~~ ~~4~~ 1
✓ P_4	6	3
✓ P_5	8	~~8~~ 2

Ready Queue

~~P_1~~	~~P_3~~	~~P_1~~	~~P_2~~	~~P_4~~	~~P_5~~
~~P_5~~	~~P_1~~	~~P_3~~	P_5		

Gantt chart

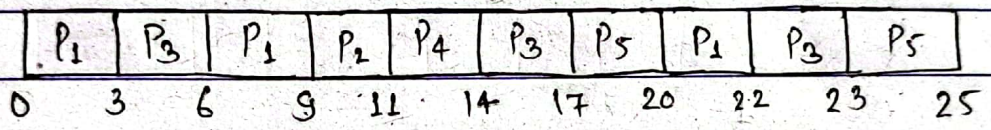
P_1	P_3	P_1	P_2	P_4	P_3	P_5	P_1	P_3	P_5
0	3	6	9	11	14	17	20	22	23	25

Round Robin (Preemptive) Time Quantum = 3

Processes	AT	BT
P1	0	~~8~~ ~~5~~ 2
P2	5	2
P3	1	~~7~~ ~~4~~ 1
P4	6	3
P5	8	~~8~~ 2

Ready Queue

Gantt Chart



⇒ Page and Frame Replacement Algorithm.

i) First In First Out algorithm (FIFO)

⇒ Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

⇒ 3 frames

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
	*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

15 page faults.

Optimal Algorithm

⇒ Replace page that will not be used for the longest period of time.

⇒ Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0
7	7	7	2		2		2			2			2		
	0	0	0		0		4			0			0		
		1	1		3		3			3			1		

7	0	1
7		
0		
1		

Total page faults = 8

Least Recently Used (LRU) algorithm.

⇒ Replace page that has not been used in the most amount of time.

⇒ Associate time of last use with each page.

P.T.O.

Final
Date
Page

⇒ Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
(3 frames)

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1			1			
	0	0	0		0		0	0	3	3			3			0			
		1	1		3		3	2	2	2			2			7			
x	x	x	x		x		x	x	x	x			x			x			

Total page faults = 12

LRU Has two implementation method

i) Counter implementation

ii) Stack implementation

Stack Implementation of LRU

4 7 0 7 1 0 1 2 1 2 7 1 2

↑ a ↑ b

2	7
1	2
0	1
7	0
4	4

Stack Before a Stack after B

Himal
Date
Page

LRU approximation algorithm

i) Reference bit

→ With each page, a bit is associated which is initially 0.

→ When page is referenced, the bit is set to 1.

→ Replace any with reference bit = 0 (if one exists)

→ The single bit doesn't give the idea of when the page was referenced. Thats why inorder to overcome such problem we used additional reference bit algorithm using shift register.

⇒ Addition Reference Bit

Each page: a shift register → 8 bit S/r

→ Now after a fixed interval of time, processor shifts the register to the right.

→ fixed interval of time's standard value ⇒ 100 ms.

Eg:

Initially ⇒ 00000000.

⇒ last eight time interval, page has not been referenced.

⇒ least value page is replaced

Himal
Date
Page

⇒ Second ~~Chang~~ chance Algorithm
→ Generally, FIFO + hardware provided reference bit.
→ Clock replacement
→ If page has reference bit = 0 then replace it.
→ If page has reference bit = 1 then,
↳ Set reference bit = 0, leave page in memory
↳ Replace next page, subject to same rules

⇒ Enhanced Second Chance algorithm.
→ Counting Algorithm
↳ Least Frequently Used (LFU)
↳ Most Frequently Used (MFU)

⇒ Page Buffering Algorithm
→ keep a pool of free frames, always
→ keep list of modified pages
→ keep free frame contents intact and note what is in them

NUMA → Non-Uniform ~~U~~ Memory access
→ To increase speed of memory.

Disk Scheduling

→ If we have to read data from many cylinders then it works just one cylind cylinder or platter and works on different request simultaneously. If multiple requests arrive then how the scheduling is ordered.

→ Try to calculate Seek Time

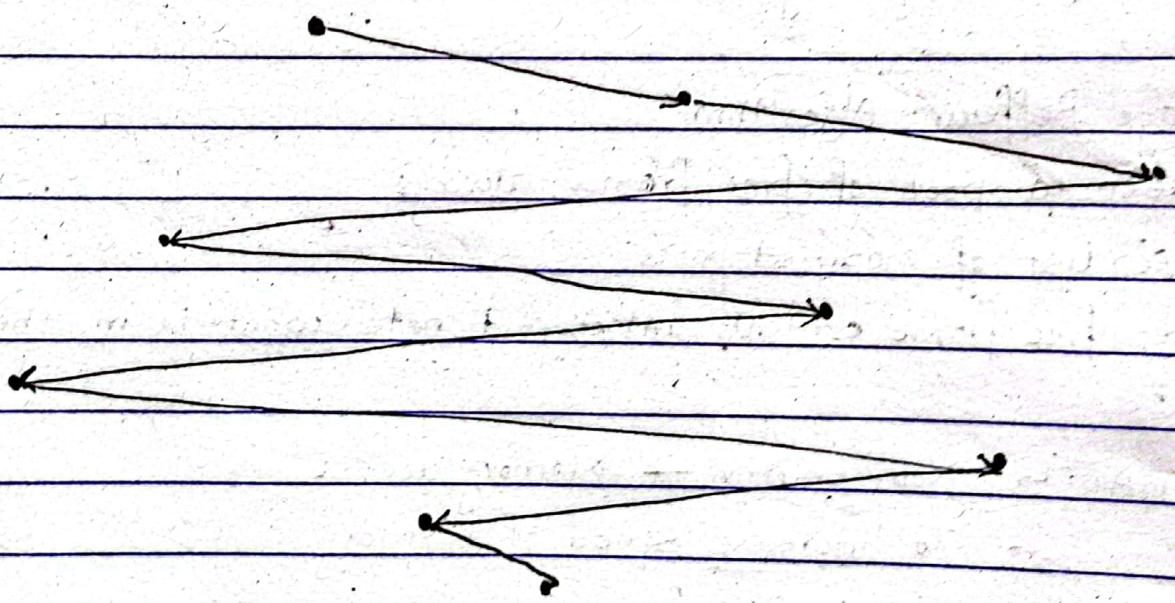
⇒ Lower the seek time, Higher the performance

1) FCFS (First Come First Served)

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

Heard Starts at 53

0 14 37 53 65 67 98 122 124 183



∴ (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

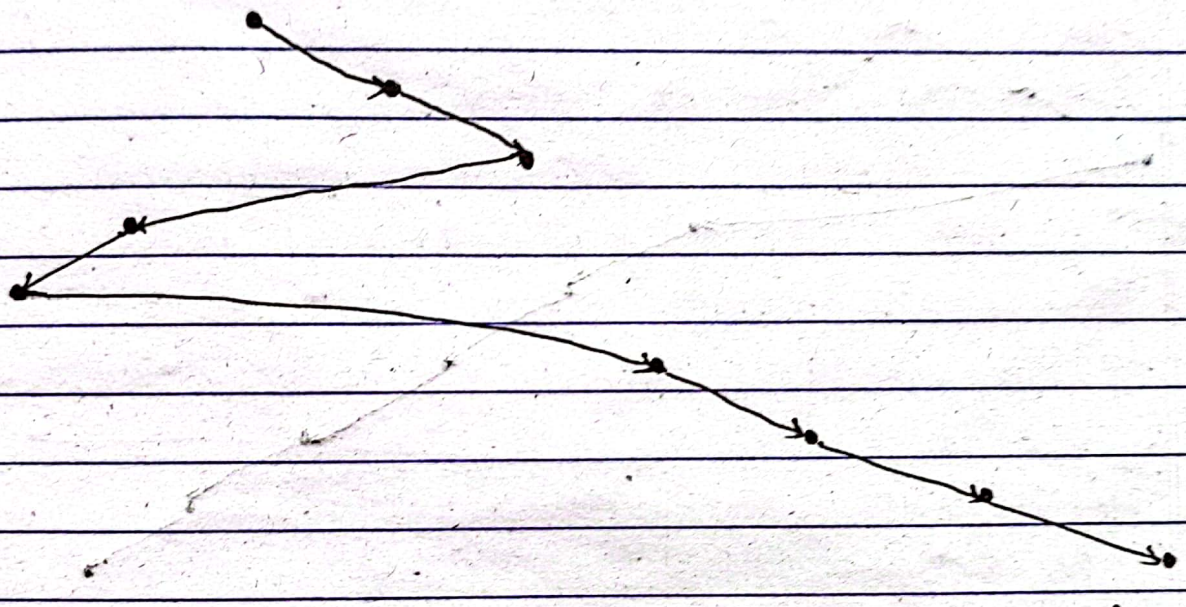
= 640

2) SSTF (Shortest Seek Time First)

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

Head start = 53

0 14 37 53 65 67 98 122 124 183 199



Total movement = (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124)

= 236

Himal
Date
Page

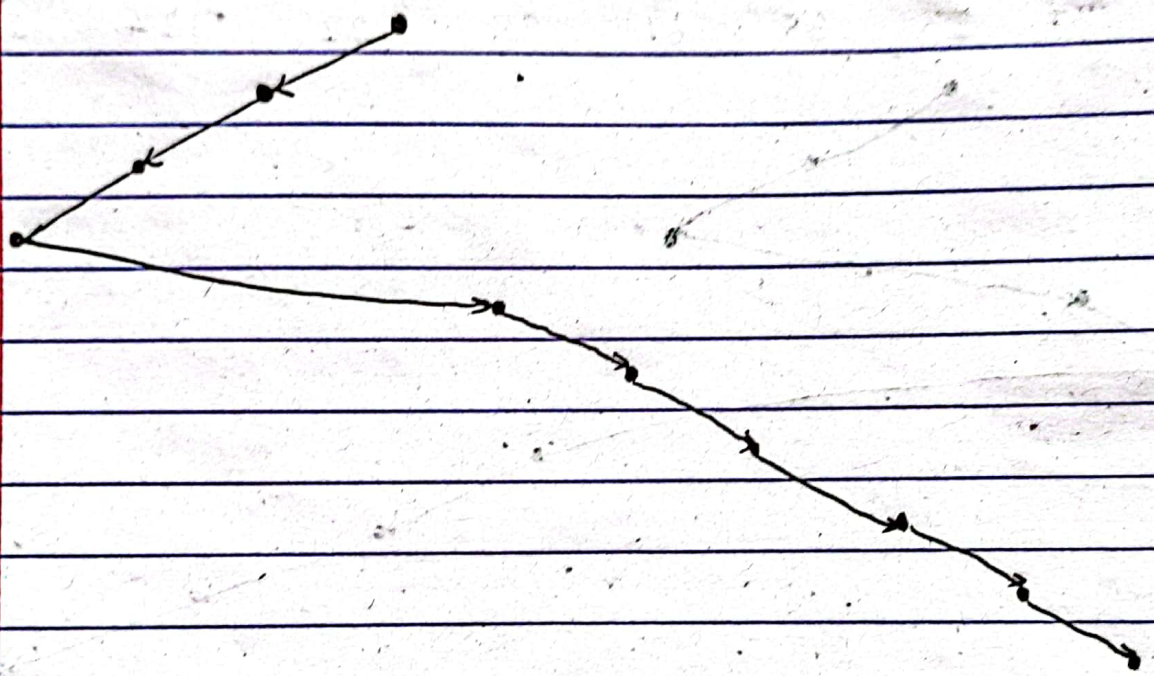
2) SCAN Algorithm or Elevator algorithm

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

Head Start : 53

Direction of Head is towards left

0 14 37 53 65 67 98 122 124 183 199



Total Movement = (53-37)+(37-14)+(14-0)+(65-0)+(67-65)+(98-67)+(122-98)+(124-122)+(183-124)

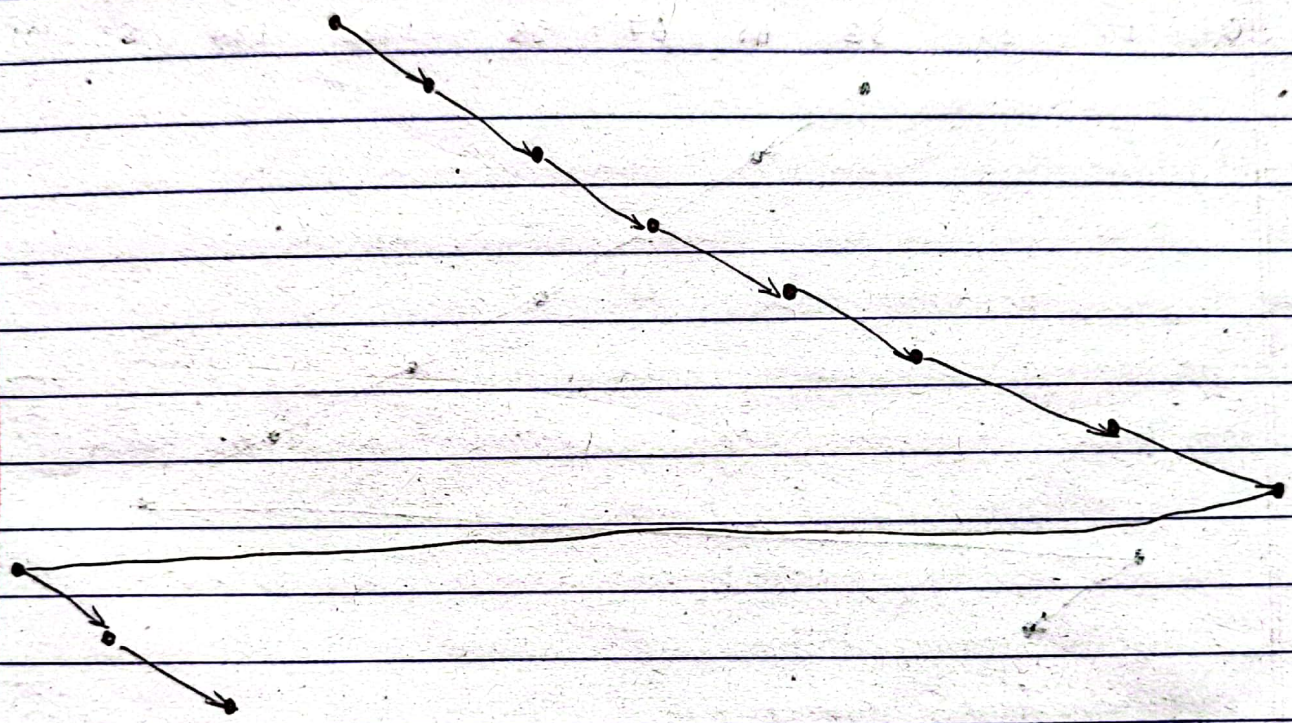
= 236

3) C-SCAN (Circular - SCAN)

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

Head Starts at 53

0 14 37 53 65 67 98 122 124 183 199



Total Movement = (65-53) + (67-65) + (98-67) + (122-98) + (124-12
+ (183-124) + (199-183) + (14-0) + (37-14)

= 183

Himal
Date
Page

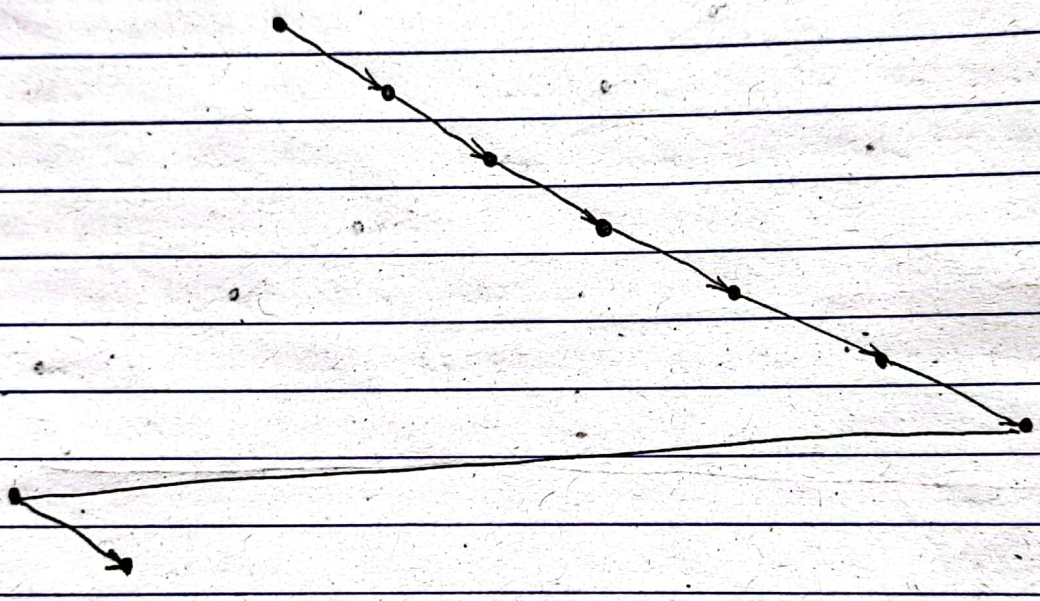
4) C-Look method (algo)

→ C-Look is a version of C-SCAN

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

Head start at 53

0 14 37 53 65 67 98 122 124 183 199



Total movement = (65-53) + (67-65) + (98-67) + (122-98)
+ (124-122) + (183-124) + (37-14)
= 153

⇒ **Thread:**

→ Is the smallest unit of processing that can be performed in an OS.

→ In modern OS, a thread exists in a process i.e. a single process can have multiple threads.

→ A thread contains a thread ID, a program counter as register set and a stack.

→ It can share with other threads belonging to the same proces such as, its code section, data section and other OS resources

⇒ **Properties:**

↳ Shares data and information

↳ Shares instruction, global and Heap regions but has its own individual Stacks and registers.

↳ Thread management consumes no or fewer system calls as the communication betⁿ threads can be achieved using shared memory.

⇒ Two types of threads

↳ User Level Thread (ULT);

→ Implemented in the user level library, they are not crea-ted using the system calls.

→ Thread switching does not need to call OS and to cause interrupt to the Kernel.

→ Kernel doesn't know about the user level threads and manages them as if they are single-threaded process.

ADVANTAGES

→ Can be implemented in OS that doesn't support Multithreading.
→ Simple to create since there's no intervention of kernel.
→ Thread switching is fast since no OS calls need to be made.

DISADVANTAGES:

→ No or less communication or co-ordination among threads and kernel.
→ If one thread causes a fault, the entire process is blocked.

↳ Kernel level Thread (KLT):

→ Kernel knows and manages threads. Instead of thread table in each process, the kernel itself has a thread table (master) that keeps track of all threads in the system.
→ OS kernel provides system call to create and manage threads.

ADVANTAGES

→ Since kernel has full knowledge about the thread, Scheduler may decide to give more time to processes having large no. of threads.
→ Good for applications that frequently block.

DISADVANTAGES

→ Slow and inefficient
→ Requires thread control block so it is an overhead.

Himal
Date
Page

⇒ Benefits of Multithreading

i) Responsiveness: application continues to run even if part of it is blocked or performing a lengthy Operation.

ii) Resoure Sharing: Allows application to have several different threads of activity within same address space

iii) Economy: Allocating memory and resources for each task is costly. Since threads share the resources of the process to which they belong, it is economi-cal to create and context switch threads.

iv) Utilization of multiprocessor architecture:

⇒ Multithreading Models

→ The user threads must be mapped to kernel, by the following Strategies.

↳ Many to one Model

↳ One to one Model

↳ Many to Many Model.

Himal
Date
Page

⇒ Inter-process communication

→ Is a mechanism that allows the exchange of data between processes

→ Processes frequently need to communicate with each other. Eg: output of the first process must be passed to the second one and so on.

→ Is also a technique for the exchange of data among multiple threads in one or more processes.

→ Processes executing concurrently in the operating system may be either independent process or co-operating process.

⇒ Two Ways of IPC:

i) Message passing:

↳ Communication takes place by means of messages exchanged between the co-operating processes.

↳ Useful for exchanging smaller amount of data.

↳ Slower than shared memory as message passing systems are typically implemented using system call which requires more time consuming task of kernel intervention.

ii) Shared Memory:

↳ A region of memory that is shared by co-operating processes is established.

↳ Process can exchange information by reading and writing data to the shared memory.

↳ Allows maximum speed and convenience of communication as it can be done at the speed of memory within the computer.

↳ System calls are required only to establish shared memory regions

⇒ Race Condition:

Situation where two or

⇒ Deadlock

→ In a computer system, we have finite number of resources distributed among a number of competing processes.

→ A process must request a resource before using it and release the resource after using it.

→ It is clear that the number of resources requested cannot exceed the total number of resources in the system.

→ Deadlock is a condition where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Conditions for deadlock:

i) Mutual Exclusion;

→ Resources involved are non-sharable.

→ At least one resource must be held in non-sharable mode, i.e. only one process at a time claims exclusive control of the resource.

ii) Hold and Wait;

→ Here, a requesting process holds resources and waiting for the requested resources.

→ A process, holding a resource waits for an additional resource(s) that is currently held by other process.

iii) No-preemption:

→ Resource already allocated to a process cannot be preempted.

→ Resource cannot be removed forcefully from the process.

→ After completion, they will be released voluntarily by the process holding it.

iv) Circular wait:

→ The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

⇒ Deadlock Prevention;

Havender's Algo:

i) Elimination of "Mutual Exclusion"

→ It holds for non-shareable resources. i.e. several process cannot simultaneously share a single resource.

→ This condition is difficult to eliminate because some resources such as the tap drive and printer are inherently non-shareable.

ii) Elimination of "Hold and wait"

→ Two possibilities:

i) A process request be granted all the resources it needs at once, prior to execution.

ii) Disallow process from requesting resources whenever it has previously allocated resources. The system must grant resources on "all or none" basis. If complete set of resources is not available, then the process must wait until the complete set is available. While the process wai

however, it may not hold any resources. Thus the "wait for" condition is denied. and deadlocks simply cannot occur.

iii) Elimination of No preemption

→ It can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated, to relinquish all its currently held resources so that other process may use them to finish their needs.

iv) Elimination of circular wait

→ Can be eliminated by imposing a total ordering on all the resources and then forcing all the processes to request the resources in order.

→ This strategy imposes a total ordering of all resource types and requires that each process requests resources in a numerical order.

⇒ PAGE TABLES PAGING FRAGMENTATION

→ Fragmentation

↳ Fragmentation means whenever some sectors in memory is damaged then data is spread all over.

i) External fragmentation (process)

→ Whenever we want to swap in an instruction to the memory and no hole is present for such process o but other free hole combine together to make enough room for such process is called External fragment

Final
Date
Page

)) Internal fragmentation

→ When we swap in a process to the memory and the particular hole which the process is in is larger which creates a situation that there is unwanted space left which hampers efficiency of memory.

→ Using first fit analysis 1/3 portion of the memory will never be used, such rule is called '50-percent rule'.
(Because of External and Internal fragmentation)

⇒ Reduce external fragmentation by compaction.

Compaction ⇒ Dynamic Memory Allocation, or Contagious Memory Allocation

↳ Shuffle memory contents to place all free memory together in one large block

↳ Compaction is possible only if relocation is dynamic and is done in execution time

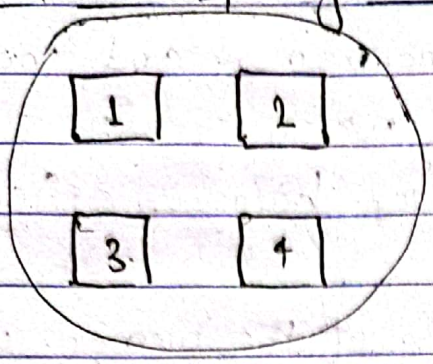
⇒ SEGMENTATION

→ A process is made up of number of segments

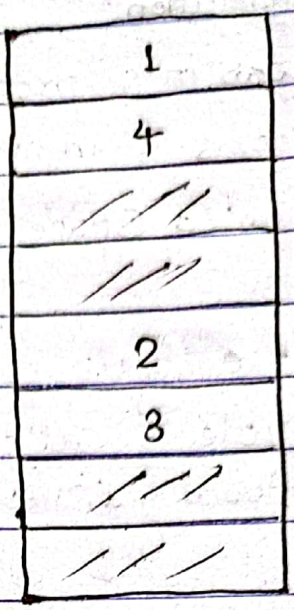
→ A segment is a logical unit such as;

↳ Main program
↳ Procedure
↳ Function
↳ Method
↳ Object
↳ Local or Global variables
↳ Common Block
↳ Array etc.

Logical View of Segmentation



User Space



Physical Memory Space

⇒ Segmentation Architecture

→ Logical address consists of a two tuple
<Segment-no, offset>

→ Segment table
↳ Is used to map logical address to physical address
↳ Base : Starting address of a segment.
↳ Limit : Specifies length of the segment

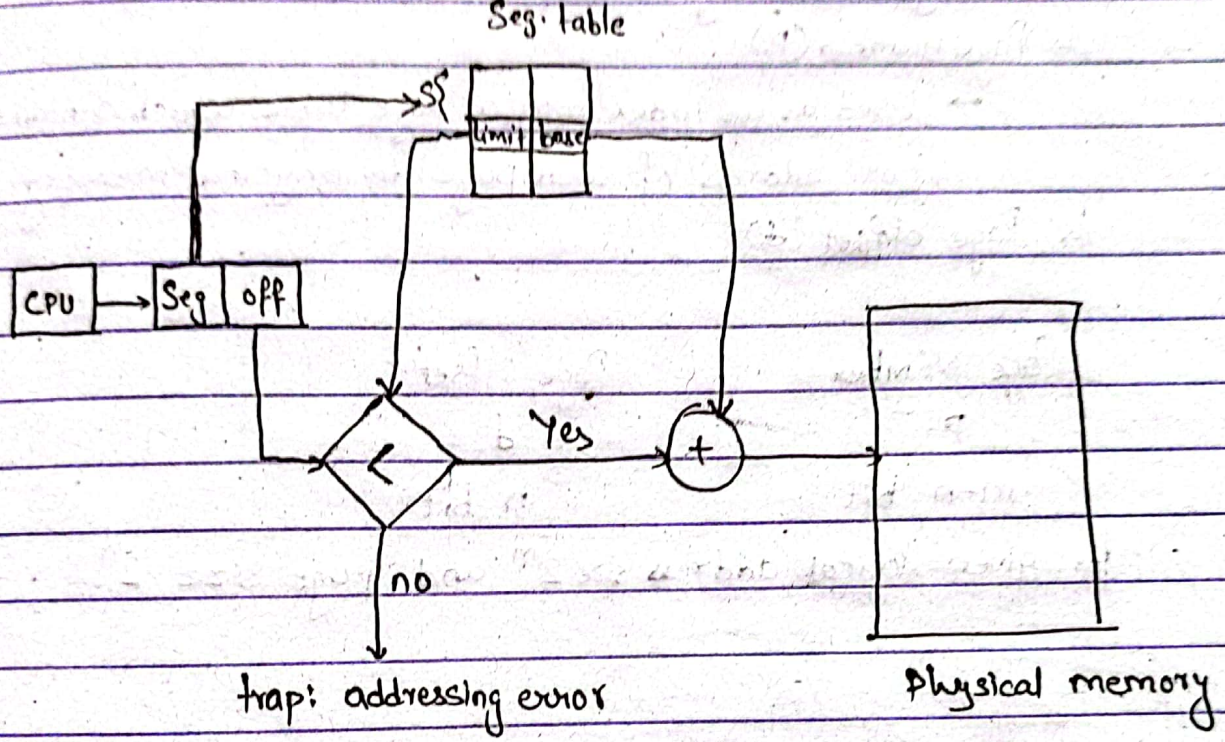
→ Segment-table base register (STBR)
↳ points to the segment table's location in memory.

→ Segment-table length register (STLR)
↳ Indicates no. of segments used by a program.

Segment number 'S' is legal if S < STLR

→ To protect Segment table, in each entry of segment table anextra bit is stored which is either '1' or '0' such that '0' → invalid and '1' → valid

Eg:



⇒ PAGING

→ Avoids External fragmentation

→ Paging is a scenario where when it is used, it avoids external fragmentation. It divides main memory into fixed number of parts known as frames.

→ Avoids problem of varying sized memory chunks.

→ frame size = process

→ Dividing logical memory into blocks of same size is called pages.

→ Setup page table to translate logical to physical addresses.

⇒ Address Translation Scheme

→ Address generated by CPU is divided into:

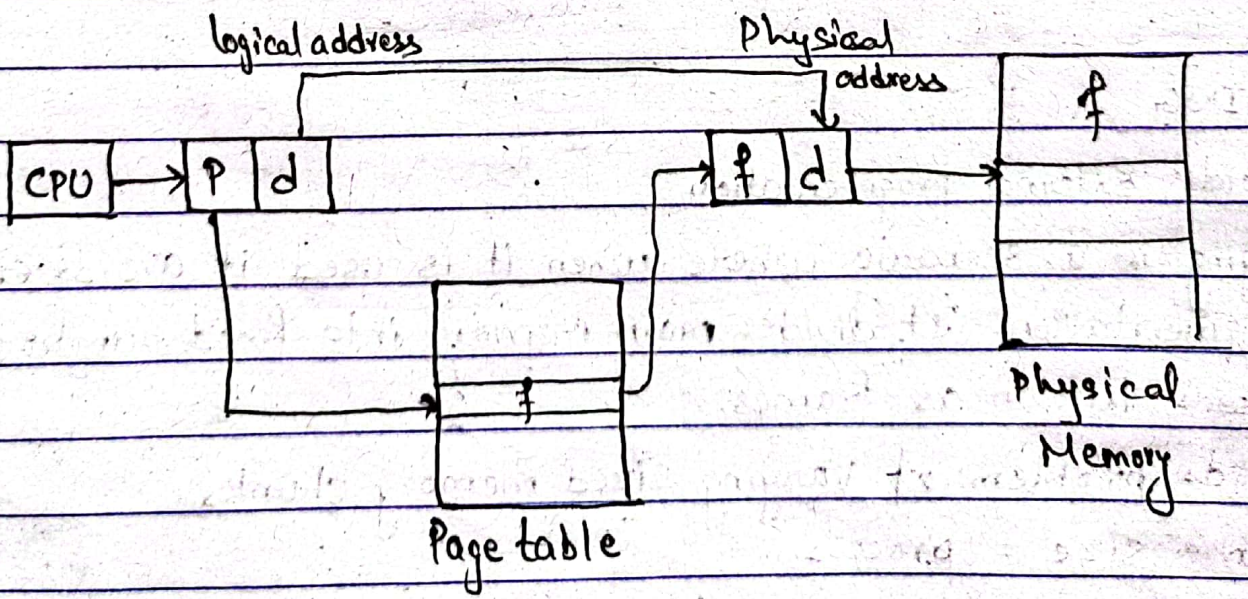
↳ Page number (p)

→ Used as an index into a page table which contains base address of each page in physical memory

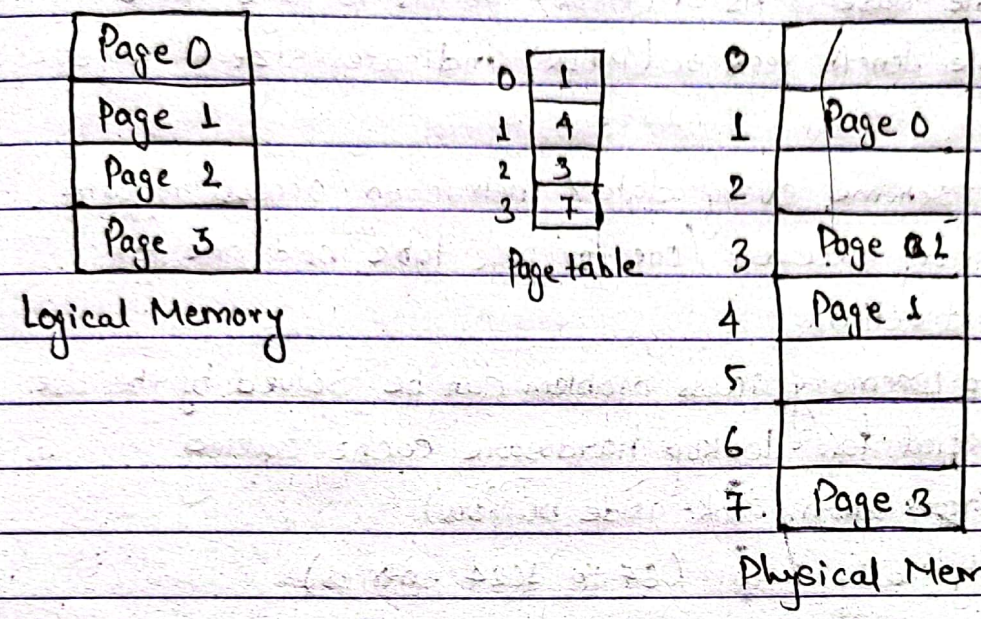
↳ Page offset (d)

Page Number	Page Offset
p	d
m-n bit	n bit

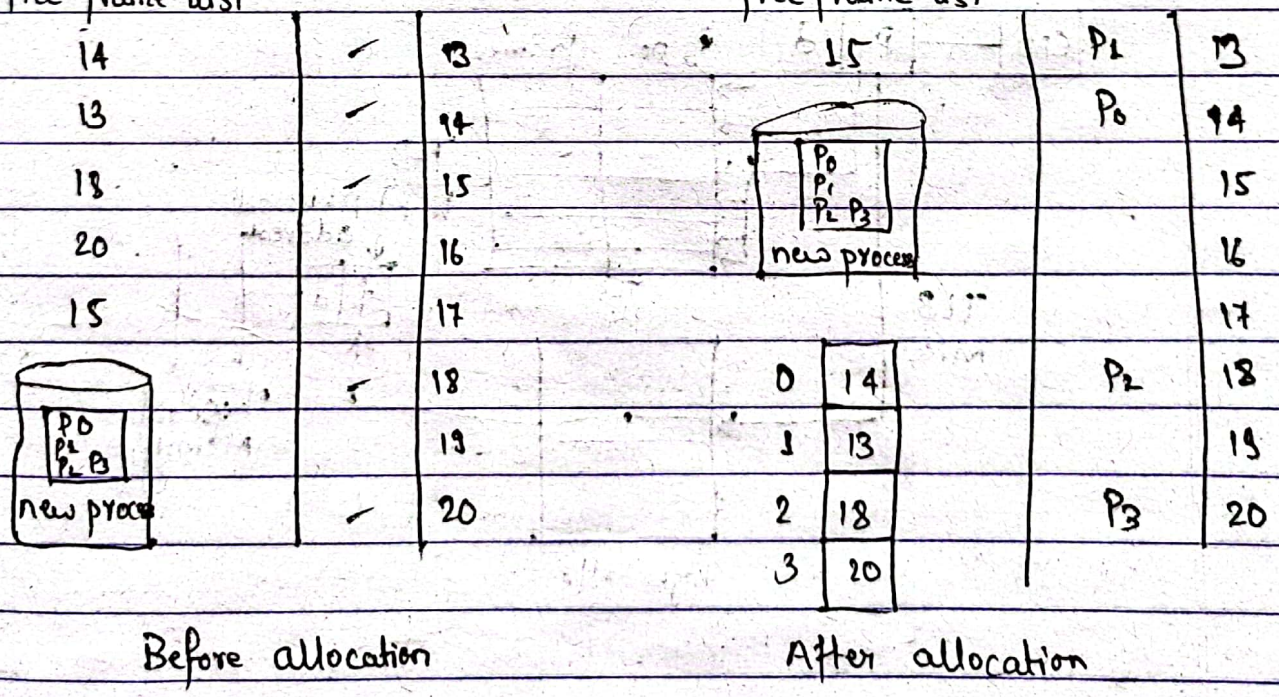
for given logical address size 2^m and page size 2^n



⇒ Paging Model of Logical and Physical Memory;



⇒ Free Frames



⇒ Implementation of Page table

→ Page-table base register (PTBR) points to page table

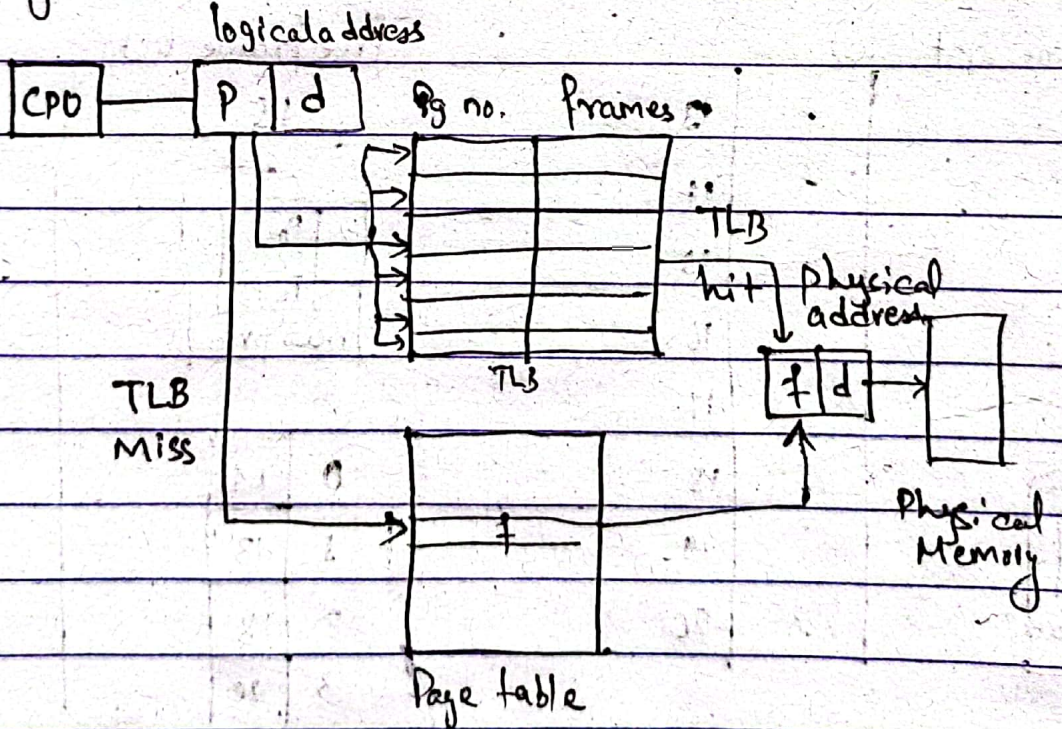
→ Page-table length register (PTLR) indicates size of page table.

→ In this scheme, every data / instruction access requires two memory accesses. (one for page table and one for data / instruction)

→ The two memory access problem can be solved by the use of a special fast-lookup hardware cache called TLBs (Translation look-aside buffers)

→ TLB is typically small (64 to 1024 entries)

Paging with TLB



→ Structure of the page table

↳ Consider a 32-bit logical address space as ~~not~~ on modern computers

↳ Page Size of 4kB (2^{12})

↳ Page table would have 1 million entries ($2^{32}/2^{12}$)

↳ If Each entry is 4Bytes → 4 MB physical address space for page table alone

↳ This memory is costing a lot

↳ Would not want to allocate that contiguously in main mem

To overcome this, we have,

i) Hierarchial paging

→ Break up the logical address space into multiple page tables.

→ A simple technique is a two-level page table

→ we then page the page table

Two-level paging Eg

→ A logical address (on 32-bit machine with 1k page size) is divided into

↳ A page no. consisting of 22 bits

↳ A page offset consisting of 10 bits

→ Since the table is paged, the page number is further divided into:

↳ A ~~10~~ 12 bit page no.

↳ A 10-bit page offset

→ Now the logical address is as follows

P_1	P_2	d
12	10	10

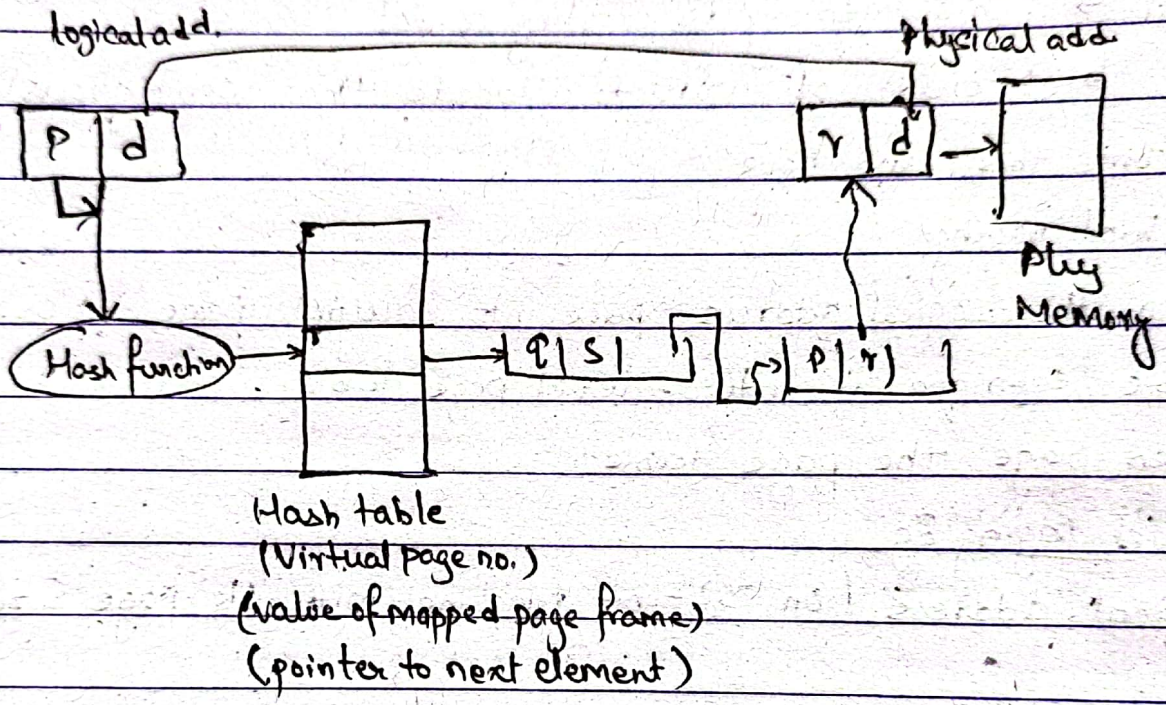
where, P_1 is an index of the outer table and P_2 is the displacement within the page of the inner table

⇒ Hashed Page Tables

→ Common in address bit greater than 32 bit

→ Contains a chain of elements hashing to the same location.

→ Virtual page numbers are compared in this chain searching for a match. If match is found, the corresponding physical frame is extracted.



⇒ Swapping:

→ Is a mechanism in which process can be swapped out temporarily out of main memory to secondary storage and make that memory available to other processes.

→ At some time later, the system swaps back the process from the secondary storage to main memory.

→ Swap-out: Process of removing a process from memory and adding it to hard disk.

→ Swap in: Removing process from secondary storage and putting it back to memory.

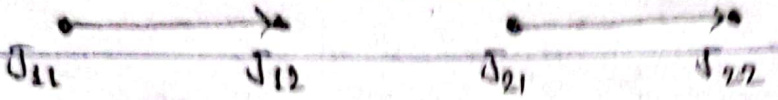
Himal
Date
Page

Weighted Round Robin Approach

→ It is commonly used for scheduling the time shared applications
→ When jobs are scheduled in Round Robin basis, every jobs join FIFO queue when it becomes ready for execution.
→ Each job in the queue executes for one time slice.
→ If it is not completed, it is preempted and placed at the end of the queue for its next turn.
→ When there are 'n' jobs in queue, each job will get $1/n$ share of processor.
→ That's why it is also known as processor sharing algorithm.
→ By giving each job a fraction of the processor, it delays completion of every job.

→ Weighted Round-Robin scheduling is a reasonable approach since a job and its successor can execute concurrently in a pipelined fashion.
→ WRR algorithm has been used for scheduling real-time traffic in high speed switched networks. It builds on the basics of RR scheme
→ Rather than giving all the ready jobs equal shares of processor, different jobs may be given different weights.
→ Here the weights of jobs refers to the fraction of processor time allocated to the job.
→ Specifically, a job with weight 'wt' gets 'wt' time slices every round, and the length of a round is equal to the sum of weights of jobs.

Eg: Let us consider two sets of jobs.
$J_1 = \{J_{11}, J_{12}\}$ and $J_2 = \{J_{21}, J_{22}\}$



→ In WRR each job J_i is assigned a weight 'w_i' then that job will receive 'w_i' consecutive time slices each round and the duration of round is $\sum_{i=1}^{n} w_i$

→ Equivalent to regular round robin if all weights is equal to 1.

→ Simple to implement since it doesn't require a sorted priority queue.

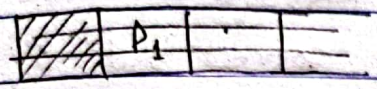
→ Offers throughput guarantee. Each job makes certain amount of progress each round.

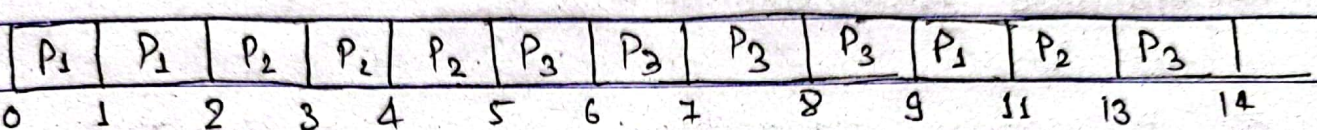
Eg:

Processes	A.T.	B.T.	Time Slot	release time	R.T.
P_1	0	4→2→0	2	0	0
P_2	2	5→2→0	3	2	2
P_3	3	5→1	4	5	5

Himal
Date
Page

Gantt chart





So,

~~WT P1 =~~

Turnaround time = process completion time - Arrival time

Waiting time = Turn around time - ~~Waiti~~ Burst time

So,

$TA_{P1} = 11 - 0 = 11$ $\quad WT_{P1} = 11 - 4 = 7$

$TA_{P2} = 13 - 2 = 11$ $\quad WT_{P2} = 11 - 5 = 6$

$TA_{P3} = 14 - 3 = 11$ $\quad WT_{P_2} = 11 - 5 = 6$

$$Avg(TA) = \frac{TA_{P1} + TA_{P2} + TA_{P3}}{3}$$

$$= \frac{33}{3}$$

$$= 11$$

$$Avg(WT) = \frac{WT_{P_1} + WT_{P_2} + WT_{P_3}}{3}$$

$$= \frac{7+6+6}{3}$$

$$= \frac{19}{3}$$

$$= 6.33$$

Real time Systems:

→ A real time system is a computer system that requires not only that the computing results be correct but also that the results be produced within a specific deadline.

→ Real time Systems are of two types:

i) Soft RTS:

↳ System is less restrictive, simply providing that a critical Real time task will receive priority over over other task.

ii) Hard RTS:

↳ Has the most stagrant requirement guaranteeing that critical real time task be completed within their deadlines.

→ Before scheduling hard RTS, we must define some characteristics of the processes that are to be scheduled.

i) The processor are considered periodic, i.e. They require the CPU at constant intervals.

ii) Each periodic process has a fixed processing time 't' once it acquires the CPU. A deadline 'd' when it must be serviced by the CPU and a period 'P'.

iii) The relationship of the processing time, the deadline and the period can be explained as: $0 \leq t \leq d \leq P$.

iv) The rate of periodic task is $1/p$.

⇒ Some scheduling algorithms that address the deadline requirements of Hard RTS are:

i) Rate monotonic Scheduling
ii) Earliest Deadline first Scheduling
iii) Proportional Share Scheduling
iv) Pthread Scheduling

i) Rate monotonic Scheduling:

Task	Capacity (C)	Period (P)
T_1	3	20
T_2	2	5
T_3	2	10

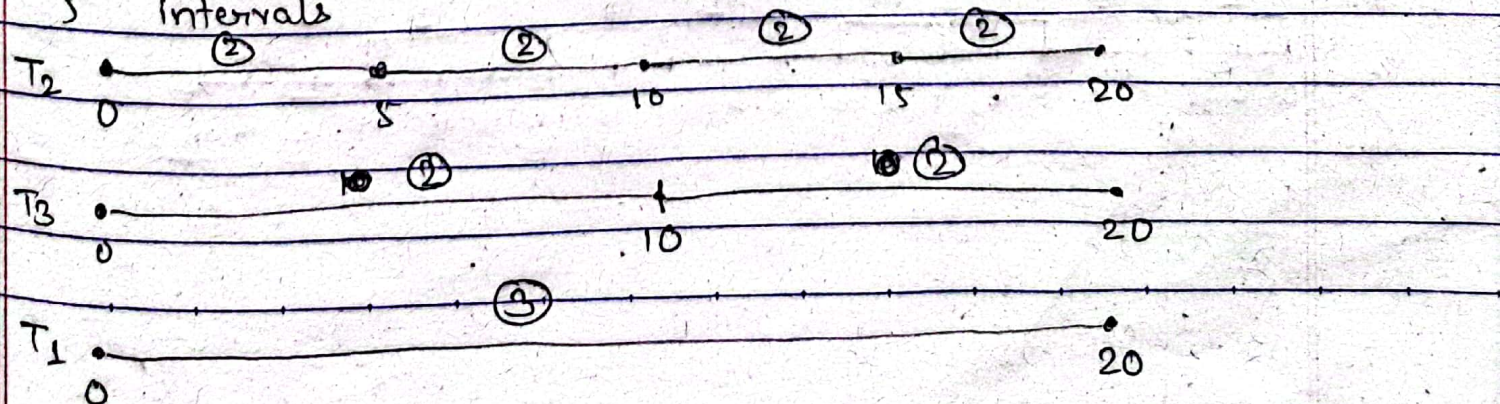
① Scheduling time: LCM (periods)

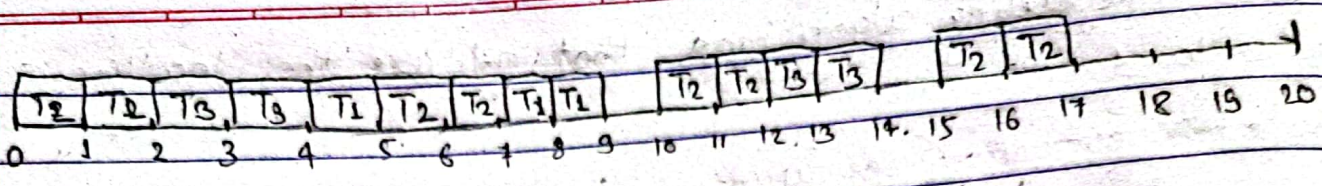
→ LCM (20, 5, 10) = 20

② Priority: whichever task having the least period will have highest priority.

→ $T_2 \rightarrow 1$, $T_3 \rightarrow 2$, $T_1 \rightarrow 3$ ($T_2 > T_3 > T_1$)

Now, Since T_2 has highest priority So, T_2 will execute two times from 0 to 5 (C = 2) i.e. it has to execute twice ever 5 intervals



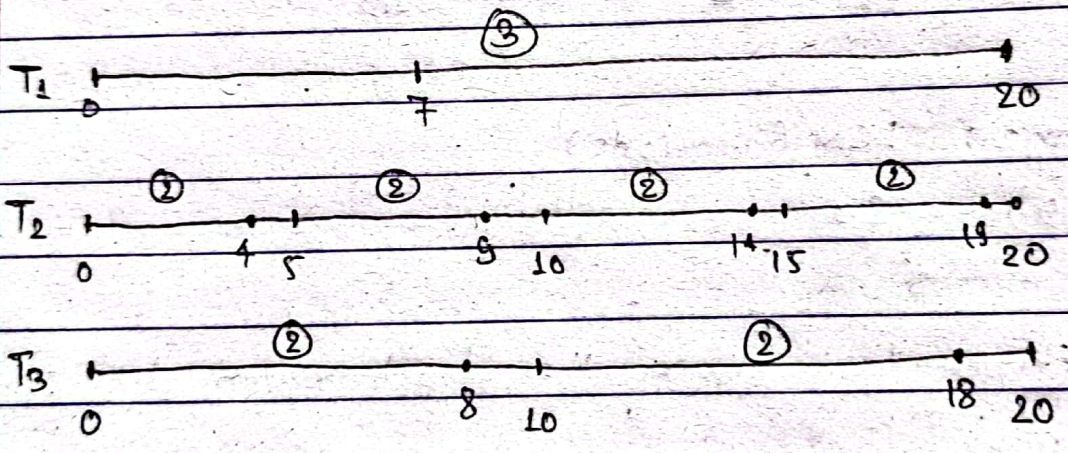


ii) Earliest Deadline first Scheduling

Task	Capacity (C)	deadline	Period
T_1	3	7	20
T_2	2	4	5
T_3	2	8	10

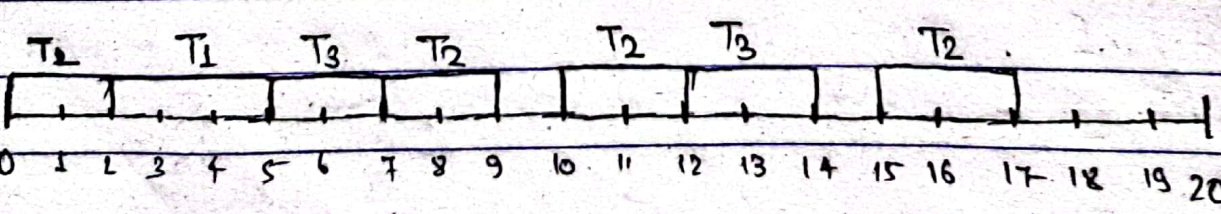
→ Scheduling time : LCM (periods)

LCM(20, 5, 10) = 20



Here,

→ T_2 has the earliest deadline So T_2 will execute first



iii) Proportional Share Scheduling

→ It works by scheduling works by dividing the total amount of time available up into an equal number of shares and then each process must request a certain share of the total when it tries to start.

→ Assume that a total of T = 100 shares be divided among 3 processes A, B and C. where,

A = 50 shares

B = 15 shares

C = 20 shares.

→ Proportional share scheduling works with an admission-control policy, not starting any task if it cannot guarantee ~~that~~ the shares that the task say that it needs.

→ If a new process 'D' requested 30 shares (100-85) = 15 left, the admission controller would deny 'D' to the system.

Himal
Date
Page

⇒ ~~Page Replacement Algorithm~~

⇒ Banker's Algorithm

↳ is a resource allocation and deadlock avoidance algo-rithm that tests for saftey by simulating the allocation for pe predetermined maximum possible amounts of all resources, then makes "s-state" check to test for possible activities before deciding whether allocation should be allowed to continue

↳ It also helps the OS to successfully share the resources between all the processes.

⇒ keep 3 things in mind

i) How many instances of each resources, each process can max request [max]

ii) How many instances of each resource, each process currently holds [Allocation]

iii) How many instances of each resource is available in the system [available]

① Need Matrix ? ② Is system in safe State? If yes then find safe Sequence

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0	1	5	3	2
P_2	1	3	5	4	2	3	5	6	2	8	8	6
P_3	0	6	3	2	0	6	5	2	2	14	11	8
P_4	0	0	1	4	0	6	5	6	2	14	12	12
									3	14	12	12

Banker's Algo:

Let n = no. of processes

m = no. of resources

→ Any 2 out of 3 (Allocated, Max, Need) must be given

Step 1: flag[i] = 0 for i = 0 to (n-1)

Step 2: find need[n][m] = max[n][m] - Allocated[n][m]

Step 3: Find a process Pi such that flag[i] = 0 and Needi <= Available

Step 4: If such i exist then,

flag[i] = 1

Available = Available + Allocatedi;

go to step 2

⇒ Free Space Management

↳ As we know that the memory space in the disk is limited. So we need to use free space of the deleted files for the allocation of new file.

↳ System maintains a free space list by keeping track of free disk space. free space list contains all the records of free space disk block

↳ These free blocks are those which are not allocated to other file or directory.

Some method of free space management are:

i) Bitmap: (Bit vector)

↳ when free space is implemented as bitmap, then each block of the disk is represented by a bit.

↳ when the block is free, the bit is set to 1 and when the block is allocated, it is set to 0.

↳ The main advantage is that, it is relatively simple and efficient in finding first free block & also the consecutive free blocks in the disk.

Calculation of block numbers:

no. of bits per words × no. of 0-valued word + offset of 1 bit

ii) Linked list:

↳ Linked list of all the free block is maintained

↳ In this, there is a head pointer which points the first free block of the list which is kept in a special location in the disk.

↳ Block contains the pointer to the next block and the next block contains the pointer to the another next block.

↳ This technique is not sufficient to ~~travel~~ traverse the list because we have to read each disk block that requires I/O time.

iii) Grouping:

↳ There is modification of the free-list approach which stores the address of the 'n' free blocks.

↳ In this the first n-1 blocks are free but the last block contains the address of the 'n' blocks.

↳ In this approach, we cannot keep a list of 'n' free disk address but we can keep the address of the first free block.

iv) Counting:
↳ Generally, some blocks are contiguously allocated by but some are free simultaneously.
↳ When free space is allocated, to a process according to contiguous allocation algorithm, we cannot keep the list of 'n' free block address but we can keep the address of the first free block & then the number of 'n' free contiguous block which follows the first block.

⇒ SCAN EDF
↳ Is a combination of the SCAN and EDF mechanisms.
↳ The seek optimization of SCAN and the real time guaran-tees of EDF.
↳ Like in EDF, the request with the earliest deadline is always served first, the specific one according to the SCAN direction is served first, among the remain requests, the principle is repeated until no request with its deadline is left.

⇒ Difference between threads and process.

Threads:
↳ Single System call can create more than one thread.
↳ Threads share data and information
↳ Threads share instructions, global & heap regions and has its own register and stack.
↳ Thread management consumes very few, or no system call because communication can be achieved by shared memory

↳ Is a segment of process
↳ Lightweight

Process:
↳ Requires separate system calls for each process
↳ it is an isolation execution entity and doesn't share data & info.
↳ Uses IPC for communication
↳ process management takes more system calls.
↳ A process has its stack, heap memory with memory & data map.

⇒ Access Matrix;
↳ It is a security model of protection state in computer system.
↳ It is used to define the rights of each process executing in the domain with respect to each ~~doma~~ object.
↳ The row of matrix represents domain and columns represent objects.
↳ Each cell of matrix represents set of access rights which are given to the processes of domain.
↳ Each entry (i,j) defines the set of operations that a process executing in domain 'di' can ~~execute~~ invoke on object O_j

Eg:

	f_1	f_2	f_3	Printer
D_1	read		read	
D_2				Print
D_3		read	execute	
D_4	read, write		read, write	

Ⓛ According to above example;

→ There are 4 domains and 4 objects.

→ A process executing D_1 can read file f1 and f3.

→ process executing D_4 has same rights as D_1 but it can also write on files.

→ -

-

→ Association bet^n domain and process can be either static or dynamic

→ Access matrix provides an mechanism for defining the control for this association bet^n domain and process.

→ when we switch a process from one domain to another, we execute a switch operation on an object

→ We can control domain switching by including domains among the objects of the access matrix.

	f_1	f_2	f_3	Printer	D_1	D_2	D_3	D_4
D_1	read		read			Switch		
D_2				Print			Switch	Switch
D_3		-read	execute					
D_4	read write		read write		Switch			

Final
Date
Page

→ File System: (Log Structured)

- Is a file system in which data and metadata are written sequentially to a circular buffer, called log.
- we get journaling automatically; all file operations are effectively atomic, reducing the need for consistency checking.
- Old versions of files are still present in the log, so we can undo mistakes or recover old versions of file.
- It is quicker than traditional method because most of the actions are made to be writes insteads of seeks.
- Seeks often are performed by memory and file systems to find a file, but a log-Structured file system usually has very few seeks, because seeks tend to take a lot of time.
- When a file system has to rewrite or change information, it normally does so by going to the individual datum and alter-ing it. (this system may be slow sometimes)
- Instead of adding this information directly to the file being changed, the log-Structure system has a log to the side and all alter--ations are saved to the log. Newer edits are added on top, regardless of what program is being edited.
- Search on log rather than whole system.
- If a log Structure file system is integrated with another system that already has minimal seeks and does not perform many memory writes, then there can be a problem.
- Writes are not well supported, so the log-Structure system's heavy use of writes becomes a burden. The other system usually has to perform delete commands to make room for the new write which takes lot of time and memory to perform.

=> Journaling File System:

↳ ~~It is~~ Its purpose is to keep track of changes not yet committed to the file system

↳ Even after any crashes or unexpected shutdowns, you can still access the latest file version with a lower likelihood of it becoming corrupted.

↳ journaling allows all the updates to a file to be stored in a contiguous portion of disk. ii)

↳ The journal file entries are scattered all over the disk. But instead of accessing them randomly, they are available in a diary-like sequence which is much faster

↳ The information about the changes is recorded in a separate log (journal) before the indexes to the files are updated.

↳ File system adopting journaling:

i) NTFS

ii) EXT2 / EXT3 / EXT4

iii) XFS

=> Semaphore:

↳ Are integer variables that are used to solve the critical section problem by using two atomic operations:

→ wait and signal that are used for process synchronization.

i) Wait:

The wait operation decrements the value of its argument S, if it is positive. If 'S' is negative or zero, then no operation is performed.

```
wait(S)
{
  while (S<=0)
  S--
}
```

ii) Signal

The signal operation increments the value of its argument 'S'

```
Signal(S)
{
  S++;
}
```

⇒ Types of Semaphores

i) Counting Semaphores

↳ These are integer value semaphores and have an ~~int~~ unrestricted value domain.

↳ These semaphores are used to coordinate the resource access, where semaphore count is the no. of available resources.

↳ If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

ii) Binary Semaphores

↳ These are like counting semaphores but their value is restricted to 0 and 1.

↳ The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.

↳ It is sometimes easier to implement binary semaphores

⇒ Strict Alternation

↳ Is a busy waiting solution which can be implemented only for two processes.

↳ In this approach, A turn variable is used which is actually a lock.

↳ The actual problem of the lock variable approach was the fact that the process was entering in the critical section only when the lock variable is 1.

↳ More than one process could see the lock variable as 1 at the same time hence the mutual exclusion was not guaranteed there.

↳ This problem is addressed in the turn variable approach. Now, A process can enter in the critical section only in the case when the value of turn variable is equal to PID of the process.

↳ There are only two values possible for turn variable, i or j. If its value is not i then its definitely j and vice versa

↳ In the entry section, in general, the process Pi will not enter critical section until its value is j or Pj will not enter in the critical section until its value is i.

↳ Initially two process Pi and Pj are available and want to execute into critical section

→ Analysis of Strict alternation approach.

i) Mutual exclusion:

↳ The process will only enter when it sees that the turn variable is equal to its PID otherwise not. Hence no process can enter in the critical section regardless of its turn

ii) Progress:

↳ progress is not guaranteed in this mechanism.

↳ If P_i doesn't want to enter into the critical section on its turn then P_j got blocked for infinite time. P_j has to wait for so long until P_i assignes it to j.

iii) Portability:

↳ The solution provides portability.

↳ It is a pure software mechanism implemented at user mode and doesn't need any special instruction from OS.

Strict alternation does not preserve the progress of the mechanism.

⇒ Multimedia ~~file~~ system.

↳ Streaming:

→ is delivering a multimedia file from a server to a client over a network connection.

↳ There are two types of streaming:

i) Progressive download:

→ Client begins playback of the multimedia file as it is delivered.

→ The file is ultimately stored on the client's computer.

ii) Real time Streaming:

→ Multimedia file is delivered to but not stored on Client's computer.

=> Real-time Streaming:

i) Live Streaming:

→ Used to deliver live event while its occuring.

ii) On-demand Streaming:

→ used to deliver media streams that is already recorded ie movies, archived lectures. The events are not delivered in real time.

=> Characteristics of Multimedia System

↳ Multimedia files can be quite large.

↳ Continuous media data may require very high data rate

↳ Multimedia applications may be sensitive to timing dela during playback of the media.

↳ Must be computer controlled

↳ are integrated

↳ information they handle must be represented digitally.

⇒ OS Security (Common Program and System Threats):

i) Program threats:

↳ OS's processes and kernel do the designated task as instructed.

↳ If a user program made these process do malicious task then it is known as Program threats.

Well known threats:

↳ Trojan Horse : traps user login credentials and stores them to send to malicious user.

↳ Trap door : which is designed to work as required, have a security hole in its code & perform illegal action without knowledge of user.

↳ Logic Bomb : is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program.

↳ Virus : Can replicate themselves on computer system. Can modify and delete user files, crash systems Are generally small codes embedded in a program.

ii) System threats:

↳ Refers to misuse of system services and network connections to put user in trouble.

↳ System threats can be used to launch program threats on a complete network.

Well-known System threats:

i) Worms : Is a process which can choke down a Sys performance by using system resources to extreme levels. A worm generates multip copies where each copy uses system reso prevents all other process to get required resources.

ii) Port Scanning : Is a mechanism or means by which a hack can detect system ~~vulll~~ vulnerabilities to make attack on the system.

iii) Denial of Service : prevents user to make legitimate use of the system.

=> Morris Worm:

↳ Was a self-replicating computer program written by Robert Tappan Morris.

↳ the purpose of the worm was to gauge the size of Internet of that time, ARPANET

↳ Although it unintentionally caused (DOS) for around 10% of the 60000 machines connected to ARPANET in 1988.

↳ The worm spread by exploiting vulnerabilities in UNIX Send mail, finger and rsh as well as by guessing weak passwords.

↳ Before spreading to a new machine, the Morris worm checked if the machine had already been infected

and was running a Morris worm process.

↳ If the target machine had already been infected, the Morris worm would re-infect it 1 in 7 times.

↳ This practice of "1-in-7 re-infection" ensured that a user could not completely avoid a Morris worm infection by creating a fake Morris worm process pretending that the machine is already infected.

↳ The United States v. Morris (1991) court case resulted in the first conviction under Computer Fraud and Abuse Act, 1986, with Morris receiving a sentence of 3 years in prison, 400 hours of community service and a $10,000 fine.

⇒ Firewall (System and Network)

↳ Can be defined as a special type of network security device or a software program that monitors and filters incoming and outgoing network traffics based on a defined set of rules.

↳ Acts as a barrier between internal private networks and external sources.

↳ primary purpose of a firewall is to allow non-threatening traff and prevent malicious or unwanted data traffic the computer from viruses and attacks.

↳ Is a cyber security tool, Can be both hardware & softwar

↳ Hardware: Broadband router, Software: Installed program.

⇒ Distributed System: Cache update policy in distributed system.

↳ Use many central processors to serve multiple real-time applications and users. As a result data processing jobs are distributed bet^n the processors.

↳ It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory.

↳ They are also known as loosely coupled systems.

↳ The users of a true distributed system should not know on which machine their programs are running.

Cache Update Policy:

↳ Write-through:

→ Write data through to the master disk as soon as they are placed on any cache.

→ Reliable, but poor performance.

↳ Delayed-write:

→ Modifications written to the cache and then written through to the server later.

→ Some data may be overwritten before they are written back.

→ ~~User~~ Unwritten data will be lost whenever a user machine crashes.

→ write on close, update on regular intervals.

Himal
Date
Page

⇒ Real-Time kernel:

↳ Not necessarily superior or better than a standard kernel. Instead, it meets different system requirements.

↳ It is an optimized kernel designed to maintain low latency and consistent response time.

⇒ File System implementation:

↳ Defines how files and directories are stored, how disk space is managed and How to make everything work efficiently and reliably

↳ File system is organized in many layers:

Application programs

↓

Local File System

↓

File system organization module

↓

Basic File System

↓

I/O Control

↓

Devices

i) Devices;

↳ Device driver manages I/O devices at the I/O control layer. Device driver controls the physical devices.

ii) I/O control level;

↳ Device driver acts as interface between devices and OS, they help to transfer betⁿ disk and main memory.

↳ It takes block number an input and output.

↳ Gives low level hardware specific instruction.

iii) Basic file system:

↳ It issues general commands to device drivers to read and write physical blocks on disk.

↳ It manages memory buffers and caches.

↳ Buffer stores content of a disk block.

↳ Caches stores frequently used file system metadata.

iv) File Organization Module:

→ It has information about files, location of files and their logical and physical blocks.

→ Physical blocks do not match with logical block numbered from 0 to N.

→ It has the free space which tracks unallocated blocks.

v) Logical File System:

→ It manages metadata information about a file i.e. includes all details about a file except that the actual contents of file.

→ It also maintains via file control blocks.

→ File control block (FCB) has information about a file - owner, size, permissions, locations of file contents.

vi) Application programs

→ Logical layers can be implemented by any coding method according to OS designer

⇒ Caching and Remote Service

↳ Caching can be served locally so that access can be faster
↳ Network traffic, server load is reduced. Improved scalability.
↳ Network overhead is less when transmission of big amount of data.
↳ for maintaining consistency, if less writes then better performance in maintaining consistency. if more frequent writes then poor performance..
↳ Caching is better for machines with disk or large main memory
↳ Lower level machine interface is different from upper level user interface.

Remote Service:

↳ Remote access is handled across the network so it is slower than caching.
↳ Increased server load and network traffic, performance degrade
↳ Transmission of series of response to specific request leads to higher network over head in comparison to caching.
↳ for maintaining consistency, communication between client and server is there to have a master copy consistent with client's cached data
↳ Remote service is better when main memory is small.
↳ just an extension of local file system interface across network.

=> Compare MPEG-1, MPEG-2 and MPEG-3

→ MPEG-1 ;

↳ Moving Picture Experts Group
↳ The first standard launched by MPEG was MPEG-1.
↳ Provides the video resolution of 352 x 240 at 30 fps.
↳ It provides lower quality than VCR
↳ MPEG-1 was used for transmission of non-interlaced videos

→ MPEG-2

↳ Provides the resolution of 720*480 and 1280*720 at 60fps and is used by DVDs.
↳ Uses interframe compression and Intra frame compression.
↳ Inter frame compression
  → it compares the current frame with last sent frame.
  → Only different frames are sent & repeating frames are not sent.
  → It avoids the repetition of frames.

↳ Intra frame compression
  → Each selected frames is divided into the block of 8*8 pixels.
  → These pixels are compared with nearest pixels.
  → Only different pixels are sent.

↳ MPEG-2 compression reduces 90 mbps bandwidth video into less than 1.5 mbps video, saving a huge bandwidth without loss in quality of picture.

→ MPEG-3

↳ Design for HD Television. But it was abounded and incorporated into MPEG-2

→ MPEG-4

↳ Higher quality compression technique than the previous one.

↳ Avoids the limitation of MPEG-2, it was standarized in Oct. 1998

↳ It is applicable for:

- → Web pages
- → Digital TV
- → Animated graphics

↳ MPEG-4 compression utilities the Media Block concepts.

↳ Media Block Contains basically 3 parts;

- → Video object
- → Audio object
- → Still images

⇒ Requirement of Multimedia Kernel.

↳ Rate requirements and deadlines are known as Quality of Service (QOS)

↳ There are 3 QOS levels;

i) Best Effort Service:

↳ The system makes a best-effort attempt to satisfy the requirement. However, no guarantees are made.

ii) Soft QoS:

↳ This level treats different types of traffic in different ways, giving certain traffic streams higher priority than other streams. However, no guarantees are made.

iii) Hard QoS:

↳ The quality-of-service requirements are guaranteed.

Parameters defining QoS:

→ Throughput:

↳ Is the total amount of work done during a certain interval.

↳

→ Delay:

↳ Refers to the elapsed time from when a request is first submitted to when the desired result is produced.

→ Jitter:

↳ Refers to delay that occur during playback of the stream

→ Reliability:
↳ Refers to how errors are handled during transmission and processing of continuous media.

=> Difference between distributed OS and Network OS.

i) Distributed OS
↳ Main objective is to manage the hardware resources.
↳ Communication takes place on the basis of messages and shared memory.
↳ less scalable than Network OS.
↳ While in DOS, fault tolerance is High.
↳ Rate of autonomy is less
↳ Ease of implementation is less
↳ All nodes have same OS.

ii) Network OS.
↳ Main objective is to provide the local services to remote client.
↳ Communication takes place on the basis of files.
↳ More scalable than DOS.
↳ Fault tolerance is less
↳ Rate of autonomy is high
↳ Ease of implementation is high
↳ All nodes can have different OS.