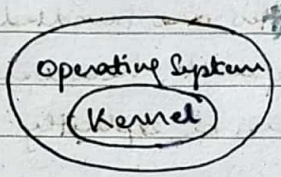
OS Internal (I).

1) Operating System is a system program that runs on computer so that an interface is provided to the computer and the user is able to operate on the computer.
(Manages the hardware) & manages application programs.

2) Kernel is a system program that controls all other programs running on the computer.



3)	Kernel	Operating System
i) Basic	Kernel is an imp part of the OS.	OS is a system program.
ii) Interface	Kernel is an interface b/w software and hardware of the computer.	Interface between the user and the hardware of the computer.
iii) Type	• Monolithic • Microkernel	• Single and multiprogramming batch sys. OS • Distributed OS • Realtime OS.
iv) Purpose	• Memory management • Process management • Task management • Disc management	In addition to the purposes of the kernel, the OS is responsible for the protection and security of the computer

Q1) How is the kernel different than the OS? Why do we say that an OS is more than a kernel?

Ans) The kernel is the core of the OS. The OS consists of the kernel, hardware management software (device drivers), various sys libraries and specialised utilities. Hence an OS is much more than a kernel.

PAGE:
DATE: / /

Operating System Utilities –

We know OS is a system software that manages the various system resources. There is, however, a system body of software, that while not strictly a part of the OS itself, cannot be termed as application s/w. This s/w is often bundled with the OS s/w and is called utility s/w.

Most of the utilities are bundled with a particular OS are installed by default, however a significant no. are optional and can be explicitly selected for installation

Eg- file reorganisation utilities
- backup programs
- communication services utilities.

Q2) Why do we say utilities are part of the OS but not the kernel?
How does the kernel differ from utilities?
Can we include all utilities in the kernel? justify.

Ans) Utilities come with the OS distribution. They collectively define the UI to the computer system. They provide a friendly and a convenient environment to the users.

Kernel is the core of the OS and it permanently resides in the main memory at runtime. Utilities are executed by the users and they can go on demand.

No we cannot include all utilities in the kernel because of memory space limitation and security.

5) Various kinds of packages are present in a computing system that help peoples feel comfortable in the environment. Application s/w and utility s/w are two such programs.

6) Differences between application and utility s/w.

	Application s/w	Utility s/w
i) Basic def.	A computer based prog. that is designed to perform some tasks grouped together	A program that aims to perform specific tasks that help in making the device work better.
ii) Access	Always downloaded from the internet	May also come preinstalled.
iii) Example	VLC Media Player	AVAST Antivirus.

Q3) In what ways are utilities similar to user applications? In what ways are they different?

Ans) Both of them run in the user space and use OS services to accomplish their tasks.
Diff. as above.

Q4) What are the 2 operating modes of a processor?
What is the advantage of having multiple operating modes?

Ans) User mode → The system is in user mode when it is running user application such as handling a text editor.
The transition from the user mode to the kernel mode occurs when the app. requests the help of the OS or an interrupt or a system call occurs.
Kernel mode → The system starts in kernel mode and when the OS is loaded, it executes apps. in user mode.
When the system is in user mode, the mode bit is set to 1; and when switching from user to kernel mode, it is changed from 1 to 0.

there are some privileged instructions that can only be executed in the kernel mode such as interrupts, input-output management, etc. If such instructions are executed in the user mode, it is illegal and a trap is generated.

Hence a dual mode is reqd as in its absence,

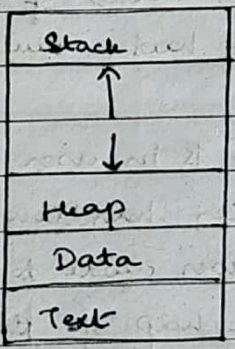
i) A running user program can wipe out the OS by overwriting it with user data.

ii) Multiple processes can write in the same system at the same time with disastrous results.

Therefore, a dual mode helps in protecting the OS & protecting the privileged resources from application processes.

Q5) Define a process & what is it used for? What is the diffrence between user and kernel processes?

Process - A process is basically a program in execution. In simple words, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.



Process life cycle :-

i) Start - Initial state when the process is started/created

ii) Ready - Process is waiting to have a processor allocated to it by the OS.

iii) Running - Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

iv) Waiting – Process moves into the waiting state if it needs to wait for a resource such as user input or a file to become available.

v) Terminated/ Exit – Once the process finishes its execution or is terminated by the OS, it is moved to the terminated state where it waits to be removed from the main memory.

User processes	Kernel processes
Also known as application processes, execute primarily applications and utilities.	Whenever they need services from the OS, they execute operating system programs. Kernel processes execute only OS programs.

Q6) Why do processes interact with one another? What are the 2 classes of process interactions?

Ans) Some processes work together to achieve application-specific tasks. From time to time they interact among themselves to know what they have collectively achieved. Some processes also compete for shared resources and they need to coordinate their use of those resources.

Two classes of process interactions :-

i) Inter-process communication – Mechanism for processes to communicate and synchronise their actions.

ii) Synchronisation – Sharing system resources in such a way that concurrent access to shared data is handled thereby minimising inconsistent data.

Q7) What is preemption? Why is processor preemption needed in an OS?

Ans) Preemption is the mechanism of forcefully taking away a resource from a process.

PTO.

In other words, it is the temporary interruption or suspension of a task without asking for its cooperation, with the intention to resume that task later. This act is called a context switch.

Processor preemption is required as it is used to multiplex a resource to various processes that are ready to execute their program.

Q8) What is the difference between preemption and interruption? In what way are they similar?

Ans) The both break the current program execution and start a new execution flow.

Interrupt breaks the current program execution and starts a new program execution in the same process context.

Preemption also breaks the current program execution, but starts a new program execution in a different process context.

Q9) Compare microkernel and monolithic kernel.

Ans) Monolithic – All the OS programs lie in the kernel space

Microkernel – Only a small fraction of the OS programs lie in the kernel space, the rest lie in the user space.

Q10) What happens if we execute kernel programs in the user mode and the apps. in the kernel mode?

Ans) The kernel will start malfunction as it would not be able to execute the privileged instructions to control the hardware.

Running the application in the kernel mode will throw the system in a state of chaos as then will be able to do anything in the system.

Q12) Why do we
Because it
the computer
and the hard

Q13) Why do we
the OS is not
starts with an
output and
Instead, it
interrupt and

Q14) Cooperating
in the system

Q15) A semaphore
below 0.

Q16) In UNIX, do
Ans) No, it only
When we
process cre
as the paren
replace all
process.

Two semap
critical se
in a mu
Two kinds
i) counti
ii) binar

Q(12) Why do we call the OS as resource manager?
Because it manages all (hardware and software) resources in the computer and acts as the interface between the user and the hardware.

Q(13) Why do we say that the OS is a reactive program?
The OS is not an active program in the generic sense that it starts with an input, crunches that input, produces some output and finally exits.
Instead, it reacts to three kinds of events - System call, interrupt and exception. Hence its name.

Q(14) Cooperating process can be affected by other processes running in the system.

Q(15) A semaphore is a shared integer variable that cannot drop below 0.

Q(16) In UNIX, does exec create a new process? Justify.
Ans) No, it only reuses the caller's pid and its process descriptor.
When we want to create a new process, we fork the current process creating a new child process which is exactly the same as the parent process; then we do an exec system call to replace all the data of the parent process with that of the new process.

Semaphore is simply a variable which is used to solve critical section problems and to achieve process synchronisation in a multiprocessor environment.
Two kinds of semaphores:-
i) Counting - can take non-negative integer values
ii) binary - can only take 0 or 1.

PAGE:
DATE: / /

Q17) Explain why UNIX-like systems use the zombie state.

Ans) It is a process state. A process which has finished its execution but still has its entry in the process table to report to its parent process is known as the zombie state. Every child process first becomes a zombie process before being removed from the table. It remains in this state until its parent obtains its status information.

Q18)

26/09/19

Self-Study

Module1

1) Introduction

2) OS functi

I) Introdu

① When a
its main

② It provid
the user.

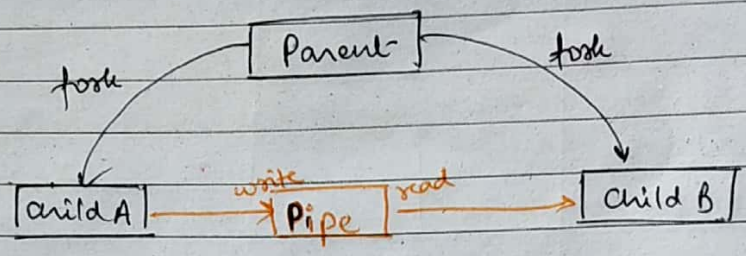
③ Basically
that acts

④ It mana
system, th

⑤ Examples

22/10/19.

PROCESSES.



1) Some special processes in Unix:-

i) PID 0 – Swapper (ie, the Scheduler) → It has nothing to do with swapping. It is the process created at start (the first process that gets created). It then creates init (PID 1) process. The swapper has nothing to do after initialisation and only gets run when there is no other process that can be/needs to be run on the system as it is given the least priority. Hence it is also called the idle process.
   - kernel process
   - No program on the disk corresponds to this process.

ii) PID 1 – init → system process responsible for bringing up a Unix system after the kernel has been bootstrapped.
   - user process with superuser privileges.

iii) PID 2 → pagedaemon responsible for paging
   - kernel process.

2) Process,

i) A basic unit of work from the viewpoint of the OS.
A process is basically a program in execution. In simple words, we write our program in a text file and when we execute it, it becomes a process that performs all the tasks mentioned in the program.

* A daemon is a comp. prog. that runs as a background process.

PAGE:
DATE: / /

ii) Types of processes:-
- Sequential → an activity resulted from the execution of a prog. by a processor.
- Multi-thread

iii) A process is an active entity.
- program code — passive entity
- stack and data segments.

iv) the current activity → PC, registers, contents in the stack and data segments.

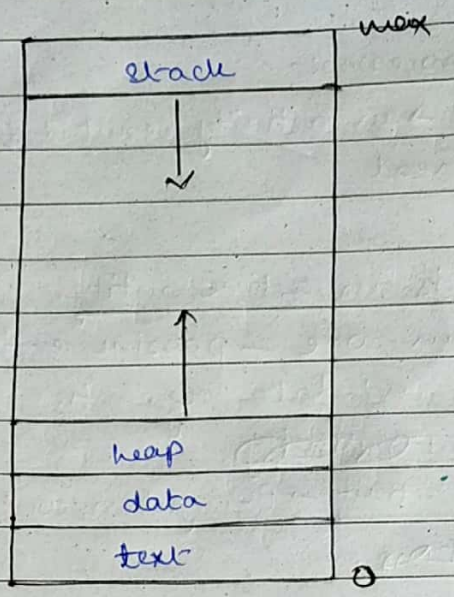
2) Process Structure.

3) Process Structure
- A process is more than the program code, which is sometimes called the text section.
- It includes the current activity
  - the value of the PC
  - the contents of the processors registers.
- It also includes the process stack which contains temporary data (function parameters, local variables, return addresses).
- It also includes the data section which contains global variables.
- It may also include a heap, ie memory allocated dynamically at process runtime.

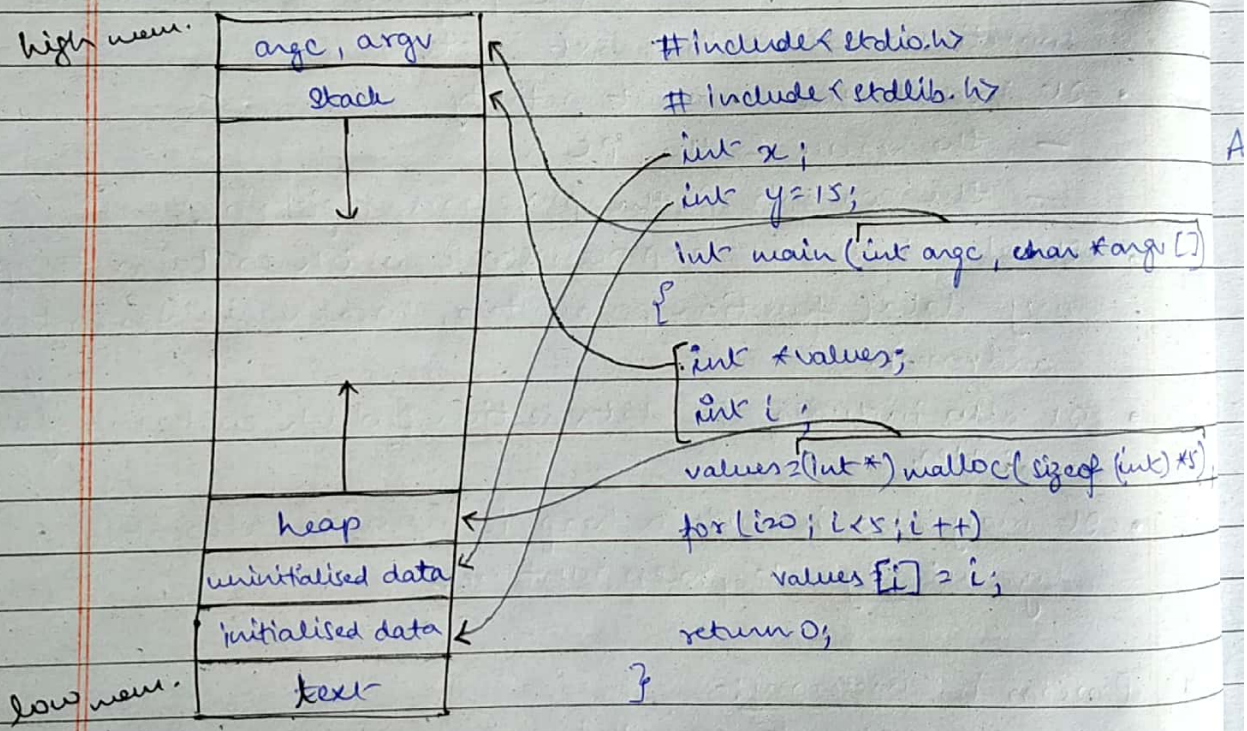
4) Process in memory:-
When we allocate something dynamically in memory using malloc, there are actually two pieces of data being stored. The dynamic memory is allocated on the heap, and the pointer itself is allocated on the stack.

```
int* j = malloc(sizeof(int));
```

So this is allocating space on the heap for an integer and on the stack for the pointer j. The variable j's value is set to the address returned by malloc.



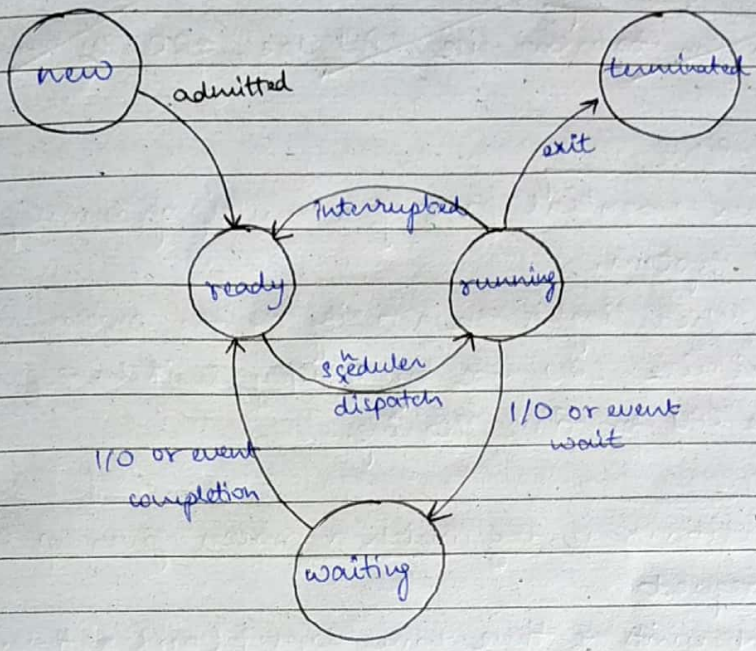
5) Memory Layout :-



6) Process States :-

As a process executes, it changes state

- new → process is being created.
- running → the instructions are being executed.
- waiting → process is waiting for some event to occur.
- ready → waiting to be assigned to a processor.
- terminated → the process has finished execution.



7) Process Control Block (PCB)

Also known as Task Control Block.

A DS that contains information associated with each process.

- process state – running, waiting, etc.
- program counter – loc. of the instruction to be executed next.
- CPU registers – contents of all process-centric registers.
- CPU scheduling information – priorities, scheduling queue pointers.
- Memory management info – mem. allocated to the process.
- Accounting info – CPU used, clock time elapsed, time limits.
- I/O status info – I/O devices allocated to process, list of open files.

Pointer
Process State
Process No.
Program Counter
Registers
Mem. Limits
Open File Lists
Misc. Accounting & Status data

PTO

PAGE:
DATE: / /

8) ~~Process Scheduling :-~~

- ~~In order to maximise CPU use, quickly the process~~

8) Process Scheduling

→ It is an essential part of a multiprogramming operating system.
Process scheduling is the activity of the process manager to maximise CPU use by quickly switching processes onto the CPU for time sharing.

→ Two conditions under which a process gives up CPU :-

- I/O request
- After N units of time have elapsed (need a timer).

→ Once a process gives up the CPU, it is added to the ready queue.

→ The next process to be executed on the CPU is selected by the process scheduler from the ready queue.

9) Scheduling queues :-

i) Job queue → set of all processes in the system.

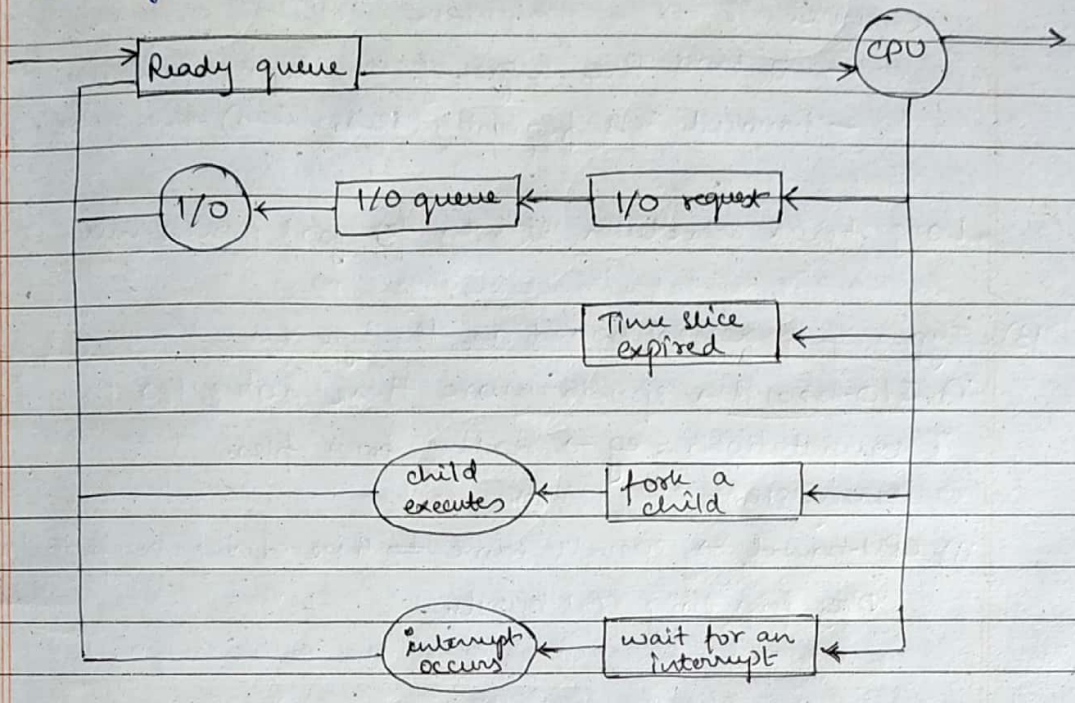
ii) Ready queue → set of all processes residing in the main memory, ready and waiting to be executed.

iii) Device queue → set of all processes waiting for an I/O device

Processes keep migrating among the various queues.

10) Representation of Process Scheduling:-

Queuing diagram represents queues, resources, flows.



11) Context Switch

- When CPU switches from one state to another, the system must save the state of the current process and reload the state of the new process via the context switch.
- Context of a process is represented in the PCB.
- Context-switch time is purely overhead, the system does no useful work while switching.
  - the more complex the OS and PCB, the longer the switch time.
- Time is also dependent on h/w support.
  - some h/w provides multiple sets of registers per CPU, hence multiple contexts can be loaded at once.

12) Schedulers.

1) Short-term Scheduler → Also called CPU scheduler, selects which process should be executed next and allocates CPU.
   - sometimes the only scheduler in the system
   - invoked frequently (ms) ⇒ hence must be fast.

ii) Long-term scheduler → Also called job scheduler, selects which processes should be brought into the ready queue.
- Controls the degree of multiprogramming.
- invoked less frequently (secs, mins) ⇒ may be slow

Long term scheduler strives for good process mix.

13) Types of processes (based on the type of events they perform):-
i) I/O-bound → spends more time doing I/O than computations. eg → reading from files.
Many short CPU bursts.
ii) CPU-bound → spends more ~~do~~ time doing computations
~~few~~ very few long CPU bursts.

Therefore, when put together....

* Long term/Job Scheduler
goal → select a good mix of I/O bound and CPU bound processes.

Remarks:-
- controls the degree of multiprogramming
- Can take more time in selecting processes because of a longer interval between executions.
- May not exist physically.

* Short term/CPU Scheduler
goal → Efficiently allocate the CPU to one of the ready processes accordily to some criteria.

* Mid-term Scheduler
Swap processes in and out of memory to control the level of multiprogramming.

14) Cooperating Process :-

Cooperating processes are those processes that can affect or are affected by other processes running on the system. Cooperating processes may share data with each other.

Reasons for needing cooperating processes:-

1. Modularity → Modularity involves dividing complicated tasks into smaller subtasks each of which can be completed by different cooperating processes. This leads to faster and more efficient completion of the required tasks.

2. Convenience → There are many tasks that a user needs to do such as compiling, printing, editing, etc. It is much convenient if cooperating processes manage these tasks.

3. Information Sharing → Sharing of info among multiple processes can be accomplished using cooperating processes. This may include access to the same files. A mechanism is reqd. by which the multiple processes can access the same files in parallel to each other.

4. Computation Speedup → Subtasks of a single task can be performed parallely using cooperating processes. This increases the computation speedup as the task can be executed faster. However, this can only be done in systems that have multiple processing elements.

15) Independent Processes :-

They are those processes that operate concurrently on a system and that can neither affect nor be affected by other processes running on the system.

PTO

16) Inter-process Communication :-

It is a mechanism which allows processes to communicate with each other and synchronise their actions. The communication between these processes can be seen as a method of cooperation between them.

Why is it required?

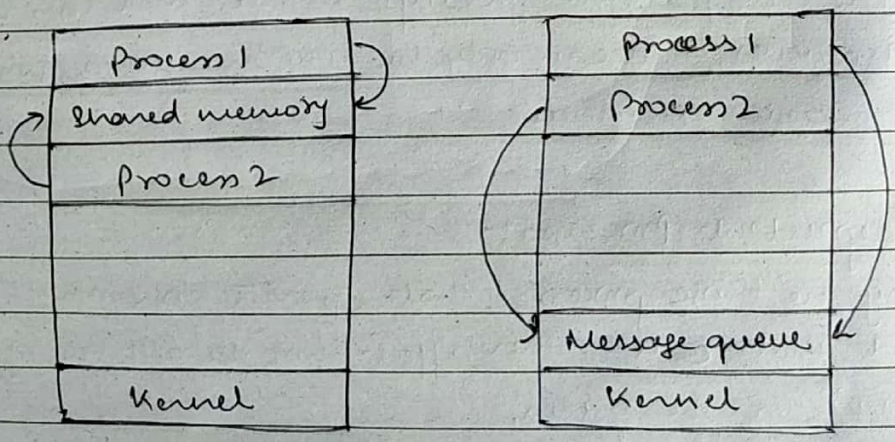
Exchanging of data and control info

Communication Models.

Two models :-

① Shared memory → It is the memory that can be accessed by multiple processes at the same time. Processes can use shared memory for extracting info as a record from other processes as well as for delivering any specific info. to other processes.

② Message passing → Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite imp. for IPC and are used by most OSs.



(a) Shared Memory (b) Message passing (queue

17) Process Synchronisation :-

Sharing system resources in such a way that concurrent access to shared data is handled thereby minimising inconsistent data.

Needs :-

① The need for synchronisation originates when processes need to execute concurrently. The main purpose of synchronisation is the sharing of resources without interference using mutual exclusion. → ensure the order of execution

② Protect critical sections.

(Critical section is the part of a program that tries to access shared resources. The critical section cannot be accessed by more than one process at the same time.)

Why do processes interact with one another?

a) Some processes work together to achieve application-specific tasks. From time to time they need to interact among themselves to know what they have collectively achieved so far.

b) Some processes compete for shared resources and they hence they need to communicate to coordinate their use of those resources.

18) Examples of IPC systems :-

There are 4 types of IPC systems :-

① POSIX API for shared memory

② Mach operating system, which uses message passing.

③ Windows IPC, which uses shared memory as a mechanism for providing certain types of message passing.

④ Pipes, one of the earliest IPC mechanisms on UNix systems.

PTO.

POSIX → Stands for Portable Operating System Interface, and is IEEE standard designed to facilitate application portability. POSIX is an attempt by a consortium to create a single standard version of Unix.

19) Ordinary Pipes.

- On Unix systems, ordinary pipes are constructed using the function,

  pipe(int fd[]) ← pipe system call.

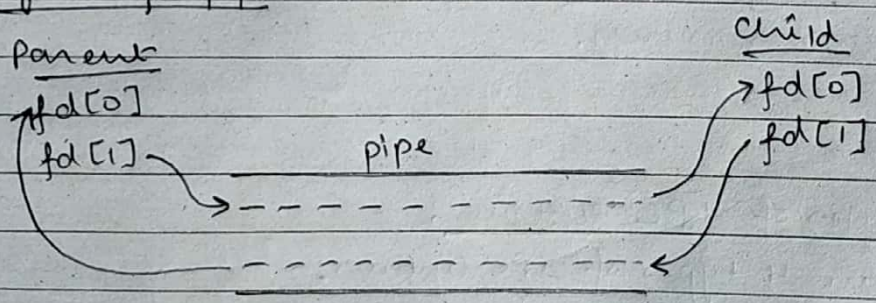
- This fn creates a pipe that is accessed through int fd[] file descriptors:-

  fd[0] ← read-end of the pipe
  fd[1] ← write-end of the pipe.

- Unix treats a pipe as a special type of file. Thus, pipes can be accessed by using ordinary read() and write() system calls.

- Windows calls these anonymous pipes.

Figure of pipe.



PAGE:
DATE: / /

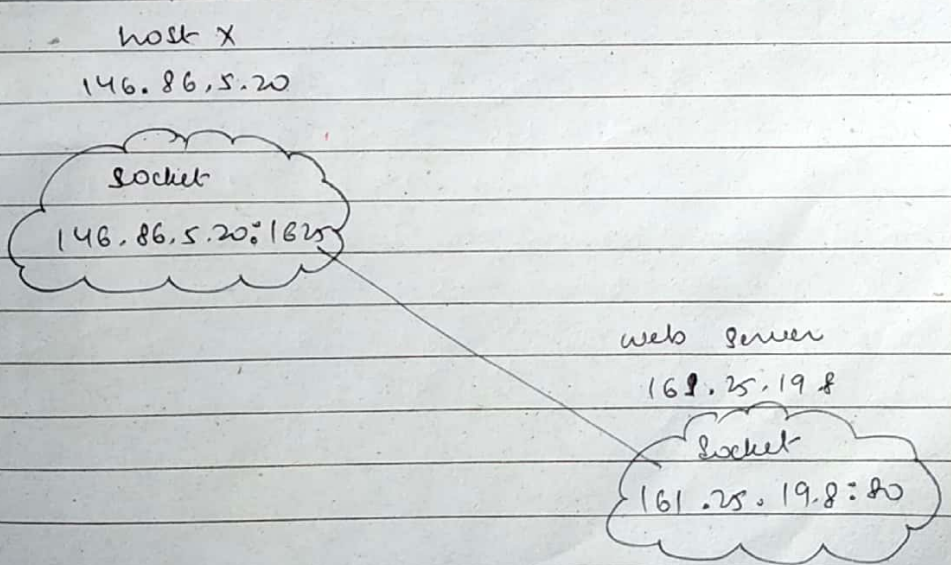
20) Communication in Client-Server Systems:-

1. Sockets
2. Remote Procedure Calls
3. Remote Method Invocation (JAVA)

1. Sockets

- A socket is defined as an endpoint for communication.
- Concatenation of the IP address and the port - a no. included in the message packet to differentiate network services on a host
  (A host is a computer that is accessible over a network. It can be a server, a client or any other type of computer.)
- The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
- Communication consists between a pair of sockets.
- All ports below 1024 are well known, used for standard services.
- Special IP address 127.0.0.1 (loopback) is used to refer to the system on which the process is running. That is, when a computer refers to the address 127.0.0.1, it is referring to itself.

Socket communication



There are 3 types of sockets:- → connection-oriented

1. TCP (Transmission Control Protocol) → It is a standard that defines how to establish and maintain a network conversation. ~~via which application progs. can exchange data.~~ TCP is a connection-oriented protocol, which means a connection is established and maintained until the app. progs. at each end have finished exchanging messages

2. UDP (User Datagram Protocol) → connectionless

3. Multicast-Socket class → data can be sent to multiple recipients.

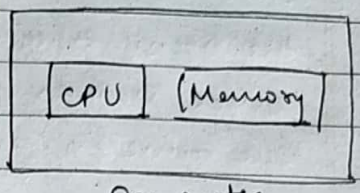
What is a procedure?

A procedure is an independent code module that fulfills some concrete task and is referenced within a larger body of source code. It can also be called a function or subroutine. The fundamental role of a procedure is to offer a single point of reference for some small goal or task that the developer or programmer can trigger by invoking the procedure itself.

PAGE: DATE: / /

Basics of Memory Management:-

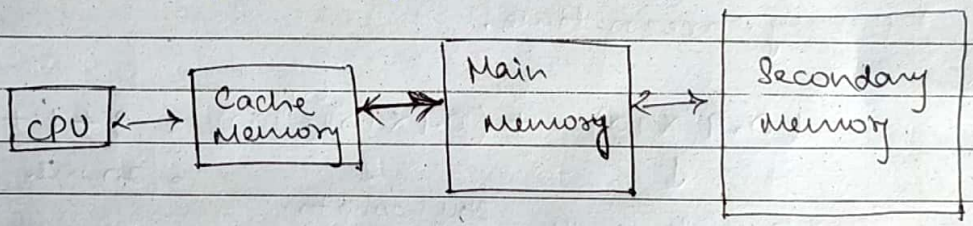
1) The functionality of a computer depends on 2 criterias:-
- CPU → how fast
- Mem → how much.



Computer.

2) 3 imp. criterias governing memory:-
1) Size → Larger memory is more desirable
2) Access time → less access time
3) Per unit cost → less

Contradicting criterias ~~to be balanced to find a balance~~
If the memory is of large size, the access time will naturally increase.
If the per unit cost needs to be less and the size large, again the access time will have to increase.
Therefore we do not have a single memory, we have a hierarchy of memory.



3) Why this hierarchy?
With a level of memories, ~~it beco~~ the possibility of fulfilling all the desirable criterias increases.

This hierarchy works based on a concept called Locality of reference.
Secondary memory → large size, less cost, access time more
Main memory → small size, hence access time less.
When an instruction is being executed, we keep the data that needs to be available readily in the main memory for reduced access time (Not always true, but hit ratio is extremely high)

→ Imp. data/ currently using data

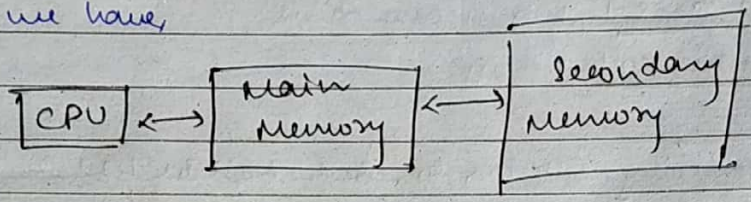
PTO.

PAGE:
DATE: / /

This is the locality of reference concept. In this manner the access time is less while we have the support of the layer secondary memory for storing everything else. The same concept can be applied to cache memory & main memory and so on.

Cache memory is a more imp. part of CA, so we'll exclude that for now.

So we have,



Secondary mem → basically permanent memory.
Main memory → Run time memory.

Q) If access time of sec. mem. is 100 ms and of main mem. is 10 ms and the hit percentage is 90%, ~~the~~ calculate the avg. access time.

$$(0.9 \times 10) + (0.1 \times (10 + 100)) = 9 + 1 + 10$$

$$= 20 \text{ ms.}$$

0.9 → 90% of the time we'll get the data in MM.

0.1 → Still having to look into MM before going to SM.

Hence we see the locality of reference has helped us reduce the access time from 100 to 20 ms.

4) 2 imp. things :-

1) How we bring data from SM to MM (contigious and non-contigious allocation)
2) How the logical address (~~used in CPU~~ generated by CPU used to access SM) is translated to physical address (in MM).

5) • Contigious memory allocation

when we bring a process from SM to MM, the memory allocated in the MM needs to be contigious, meaning one after the other, together.

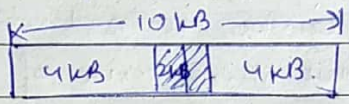
• Non-contigious memory allocation

while allocating the process space in the MM, we can break the process into ~~segments~~ pieces and arrange those pieces in different areas in the MM.

6) Problems of contigious memory allocation

① Always suffers from external fragmentation.

eg:) We have a contigious memory of 10 kB out of which only 2 kB is occupied at a certain place.



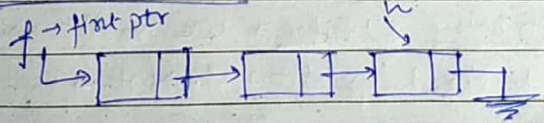
We need to allocate a process of size 5 kB contigiously, this is not possible as 5 kB contigiously is not available even though 8 kB space is available in total. This is known as the problem of external fragmentation.

Advantage

Access time is very fast as we only need the base address of the memory to look for any data in the memory, eg:- as in the case of an array.

7) Non-contigious policy

eg:- linked list.



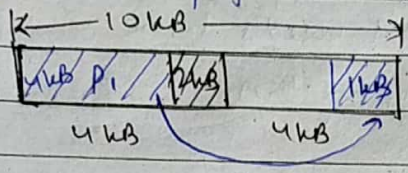
Can we go directly to nth location? No. In a linked list, we must go through all the preceeding ptrs to reach nth location.

So access time is more.

PAGE:
DATE: / /

Advantage

Problem of external fragmentation is solved.



Now when we want to allocate space to a process of 5 kB, it is possible.

- First, 4 kB will be allocated, then it can point to another location which has free space to allocate the space to the remaining 1 kB of the process non-contiguously.

~~So one~~

when there is space available anywhere in the system, we can allocate that space to a process, then the remaining part can be accessed with the help of a ptr placed in some other area of the memory.

---

(1)

Contigious memory allocation

- fixed size partitioning
- variable size partitioning.

* ~~the~~ CPU can access only those processes which have been brought ~~from the see~~ SM to the MM.

(2) Two approaches to contigious MA.

(1) Fixed sized partitioning schemes: we partition the memory into fixed sized partitions. Note → it is not at all necessary that all the partitions will be of equal sizes, only the sizes should be fixed (we cannot change them after making the partitions).

PAGE:
DATE: / /

Some partitions should be small to accomodate small processes, some should be large to accomodate large processes.

Problem → i) we cannot reuse a partition for another process
2) If the process does not occupy the entire space, there will be internal fragmentation and space will be wasted. (leftover)

2) Variable size partitioning scheme → We'll allocate space to the processes in the memory as and when they come according to their requirements. We don't have predefined partitions. Hence, the problem of internal fragmentation is solved.

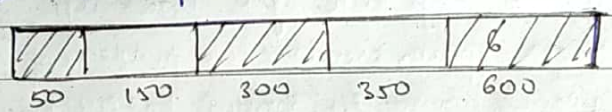
3) External fragmentation → The total space that is required is available, but it is not available in a contigious fashion. This is a problem of contigious MA.

Fixed size partitioning policy has internal fragmentation in addition to this.

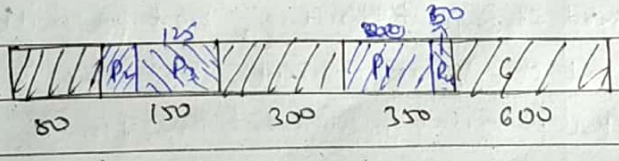
Variable size partitioning.
↳ First fit, Best fit, Worst fit.

Q) $P_1 = 300$
$P_2 = 25$
$P_3 = 125$
$P_4 = 50$.

Present state of the memory.



i) First

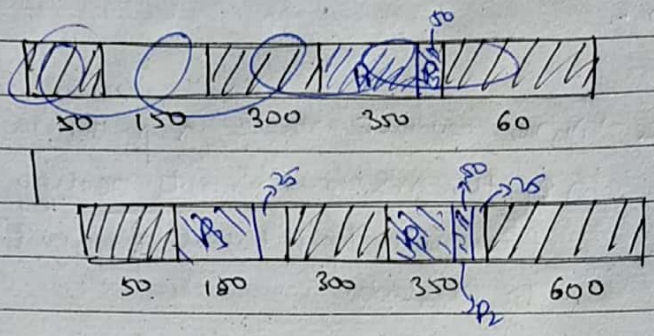


The first block (partition) capable enough for the process is used.

ii) Best Fit

The smallest ~~to~~ partition capable for the process is used, not the first.

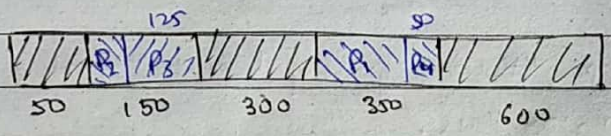
$P_1 = 300$
$P_2 = 25$
$P_3 = 125$
$P_4 = 50$



Unable to allocate space to P_4 even though we have 50 kB in total (external fragmentation).
Hence we see, the best fit is not best after all, although this is only true for variable size ~~partition~~ partitioning. In fixed size, best fit is actually best.
In variable size, worst fit is best.

iii) Worst Fit

The largest partition is used for the process.



In worst fit, since we use the largest block possible, the remaining space will also be of large size (in variable size). The probability of this block getting used will be high, whereas best fit leaves out smallest space everywhere which will not get used up for process that require larger spaces, creating the problem of ext. fragmentation.
Hence worst fit works better in variable size partitioning

In fixed size partitioning, best fit will perform best as in fixed size, we cannot use the leftover space in a block (internal fragmentation). Hence best fit will lead to smaller spaces getting wasted.

PAGE:
DATE: / /

Variable size partitioning :-

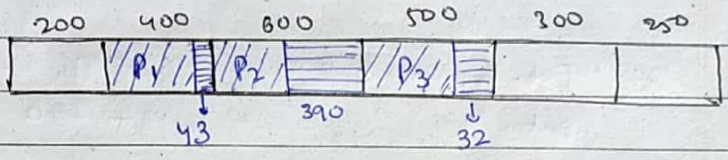
$P_1 = 357$

$P_2 = 210$

$P_3 = 468$

$P_4 = 491$

i) First Fit



Can't fit 491 (P_4).

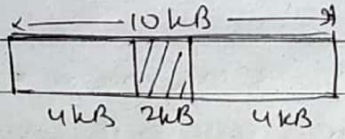
↳ because 491 kB is not available in contiguous fashion (ext. fragmentation)

~~even though we have more than 491~~

↳ space wasted on int. fragmentation → 43 + 390 + 32 = 465 kB.

↳ space wasted on ext. " → 491 kB since that is the space that was reqd. If the space reqd. was less than 300, then there would have been no ext. frag.

eg)



P = 9 kB.

No ext. fragmentation, since the tot. space available is only less than 9 kB (not our fault).

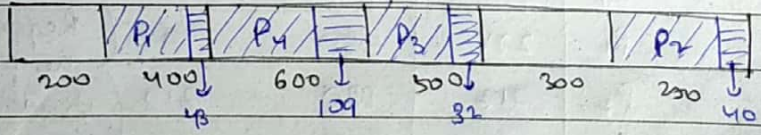
P = 7 kB.

Ext. frag. of 7 kB.

P = 3 kB.

No ext. frag.

ii) Best Fit

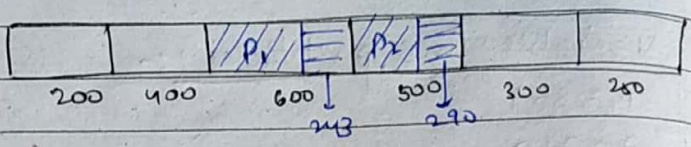


No ext. fragmentation

Int. frag = 43 + 109 + 32 + 40

= 224 kB

iii) Worst fit



$P_3 = 468$ X

$P_4 = 491$ X

Ext. fragmentation = 468+491

Int. " = 243+290

Address Translation

CPU always generates address for the LM

CPU works directly with the MM.

LM is at the back to support MM.

But CPU doesn't know that the MM exists, hence it always generates logical address for the LM

We change the LA to physical address (PA) which can be used to access MM.

If we know the base address, we can access the entire process in a continuous fashion.

Limit Register (LR) → the size of the process in units.

(Eg → if the size of one instruction is 1 unit, then a limit register of 500 units would mean there are 500 instructions)

Relocation Register (RR) → In MM, the base address of the process.

	LR	RR	LA
P_0	500	1200	450
P_1	275	550	300
P_2	212	880	210
P_3	420	1400	450
P_4	118	200	80

Requests made by the processes, we have to find if these requests are legal or illegal.

Request to access that particular process no.

PAGE:
DATE: / /

To get the final physical address:-

$P_0 \rightarrow$ Size is 500, request is for 450, so yes its allowed,
we add 450 to the base address to get the physical address.
So physical address = 450 + 1200 = 1650.

$P_1 \rightarrow$ not allowed since size itself is 275, how can we access instruction no. 300.

$P_2 \rightarrow$ Physical address = 210 + 880
= 1090

$P_3 \Rightarrow$ not allowed.

$P_4 \rightarrow$ 80 + 200 = 280.

Two major issues in contiguous memory allocation:-

① Internal Fragmentation → not as serious as ext. fragmentation.

② Ext. fragmentation → ~~two ways to solve the issue:-~~
we allocate the process in non-contiguous fashion to solve this issue.
① Paging
② Segmentation.

PAGE:
DATE: / /

Non-Contiguous Memory Allocation.

Paging

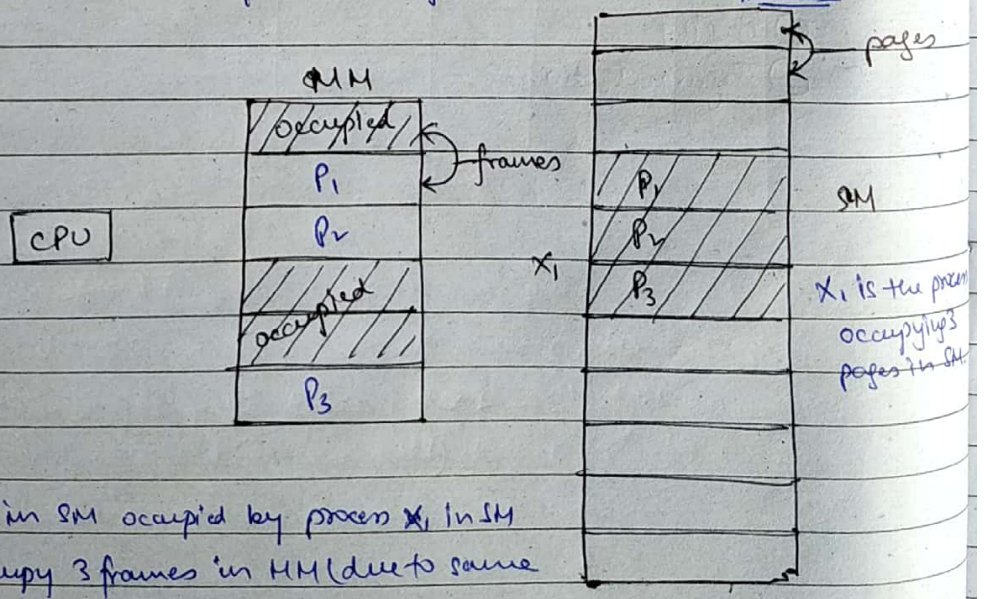
For non-contiguous ~~to~~ ma, we partition the SM and the MM into some blocks.
It will be a fixed-sized partitioning scheme. So it will result in Int. fragmentation.

We divide the SM into equal partitions (so that the management of memory becomes easier)
these partitions (fixed size blocks that are also equal in size) are called pages.

Q) what is a page?
A) the SM is divided into equal sized partitions, and these partitions are called pages.

Q) Why call them pages?
Analogous to a book. If in a book, the page sizes are different, then it will be very difficult to maintain that book.

It is simple to understand that we should divide the MM also in same sized units as the SM, so if the size of each page in SM is 1 KB, then the size of each partition in MM will also be 1 KB. The partitions of MM are called frames.



3 pages in SM occupied by process X_1 in SM will occupy 3 frames in MM (due to same size).

In the above scenario, the process X_1 can still be allocated space in the MM (as paging follows the non-contiguous scheme)

* Ordering of pages is not a must.

Translation of LA to PA in Paging.

Translation in non-contiguous scheme becomes difficult as the concept of base address doesn't hold value here.

CPU generates the LA for SM, in non-contiguous scheme we say that this LA is divided into 2 parts – the page no. and the instruction offset.

LA | p | d → offset.

~~If we want to say that~~

In contiguous scheme we would say go to instruction no. 325, here we can say go to (page 4, instruction 25)

p = 4, d = 25.

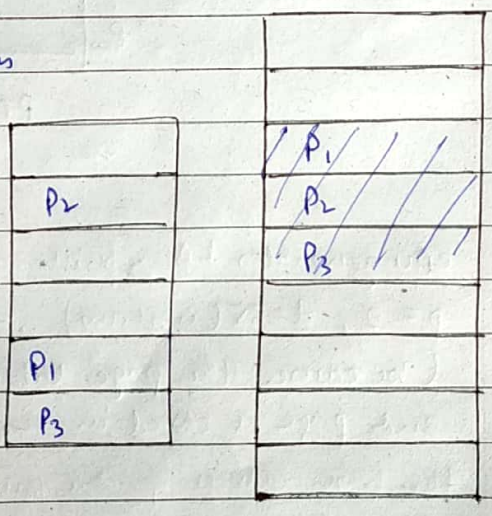
Much more organised.

(given that we are dividing the memory into 100 int.)

Task → To find out the address of the pages of SM stored in MM (since they can be arranged in any fashion)

Basic approach of non-contiguous scheme is to use the linked-list approach, ie, here last line of every page will contain the address of the next page. But this is not very feasible since worst case is O(n) address for LL.

So we use Indexing.

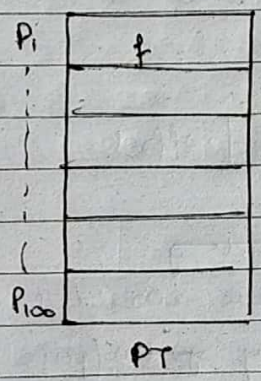


For indexing, we will use a new DS called a page table. No. of entries in our page table will be the no. of pages the process occupies in the SM.

Eg → if the process contains 100 pages in SM, then the PT will contain 100 slots as well.

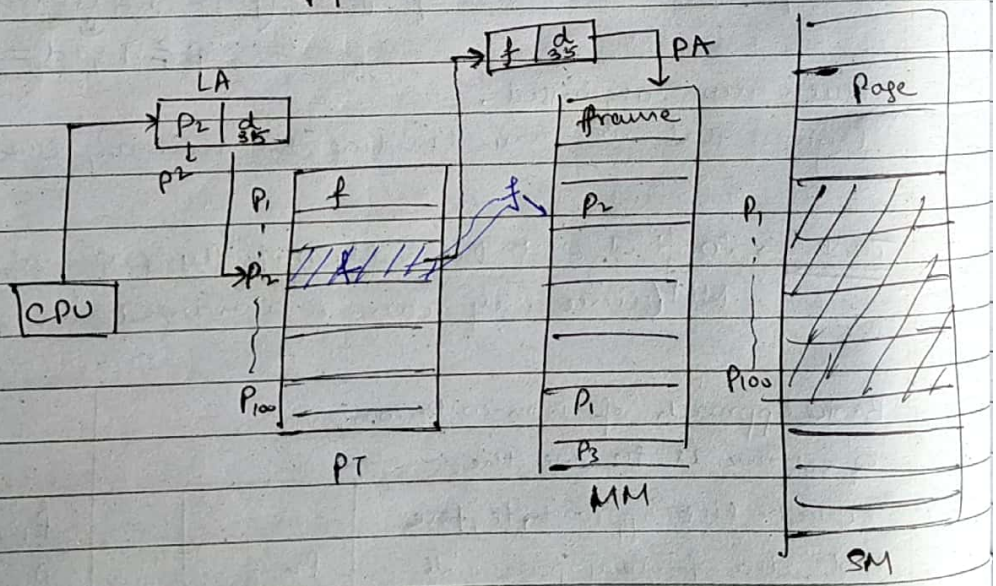
* Page table is for a specific process ⇒ Every process will have its independent page table.

The page table is nothing but the index.



Page no. holds the frame no.

Frame no → the base address of the frame in the MM in which that particular page is currently stored.



CPU generates LA which contains page no and offset.

p = 2, d = 35 (inst. no.) ⇒ we then go to the page table (because the page table index 2 will tell us where the 2nd page is stored in the MM) (this is not the frame, this is the base address of the frame)-

PAGE
DATE: / /

PTBR → Page Table Base Register

Special purpose register that holds the value of the page table. It is stored in PCB (Process Control Block)

PTBR will give us the base address of the PT.

Advantages of this scheme :-

(1) we get to access our process in a very convenient fashion.

(2) No Ext. fragmentation

Disadvantages :-

(1) We are paying heavy penalty in maintaining a separate data structure that is the PT.

(2) To make the page table useful, we are also having to remember the base address of all the frames individually since we do not want the linked-list approach either.

(3) Every process has to have an independent page table.
It is not stored in PCB (∵ very big in size)
(we access it through PTBR)

(4) ∵ Paging is FSP scheme, it suffers from int. fragmentation.

(5) The access time is doubled since PT being a ds is also stored in MM. So first we access the PT to find the frame no and then we again access the MM to get the inst. no. from the relevent frame no.

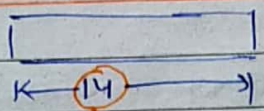
Addr Translation of Address to Space.

If an address is of n bits, then tot. no. of addresses (ie, memory locations) possible in the mem. is 2^n.
Size of each mem. loc. is usually given as byte addressible (usually 1 Byte) but may be different.
∴ Size of the memory (space) will be no. of addresses × size of each add.

$= 2^n \times 1B$ (usually)

PAGE:
DATE: / /

Eg) i)

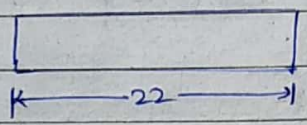


no. of address bits → 14 → No. of locations.

∴ Space = $2^{14} \times 1B = 2^{4} \times 2^{10} \times 1B$ ($2^{16} \rightarrow 1K$)

= 16kB

ii)



∴ Space = $2^{22} \times 2B$

$= 2^{23} \times 1B$

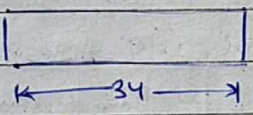
$= 2^{20} \times 2^{3} \times 1B$ ($2^{20} \rightarrow 1M$)

$= 8 \times 1M \times 1B$

= 8 MB.

$2^{10} \rightarrow 1K$
$2^{20} \rightarrow 1M$
$2^{30} \rightarrow 1G$
$2^{40} \rightarrow 1T$
$2^{50} \rightarrow 1P$

iii)



∴ Space = $2^{34} \times 1B$

$= 2^{30} \times 2^{4} \times 1B$ ($2^{30} \rightarrow 1G$)

= 16 GB

Space to Address Translation

If we have a memory of size 64 KB (assuming the mem is byte addressible)

Space = n × 1B.

∴ n = Space / 1B

n ⇒ $\lceil \log_2 n \rceil$

Memory of 64KB. ∴ No. of locations = 64K

∴ 64K = $2^{6} \times 2^{10} = 2^{16}$,

∴ No. of bits required = 16.

PAGE
DATE: / /

eg) i) | 32kB (1B) |

Size = 32kB
Size of each location = 1B.

∴ no. of locations (n) = 32k.

e.g. $32k = 2^5 \times 2^{10}$
$= 2^{15}$.

∴ No. of bit reqd. for address = 15.

ii) | 256 MB (1B) |

MS = 256 MB.
Size of each location = 1B.
∴ n = 256 M.
$= 2^8 \times 2^{20}$.
$= 2^{28}$.

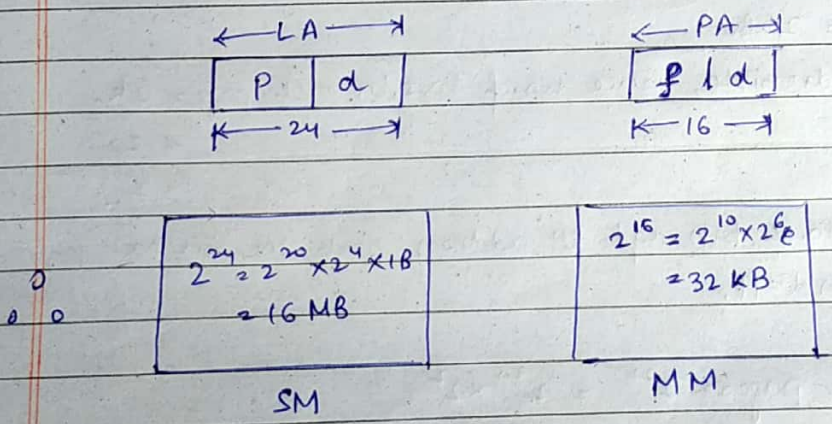
∴ No. of bit = 28.

iii) | 16 GB 4B |

MS = 16 GB.
∴ n = $\frac{16\ GB}{4B}$ = 4G.
$= 2^2 \times 2^{30}$.
$= 2^{34}$.

∴ No. of bit = 34

Paging



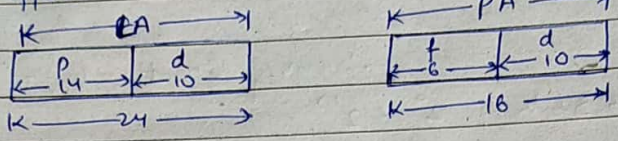
g) LA = 24b
PA = 16b
PS = 1 KB.
= FS

Now, page size = 1 kB. (given) and we know that every loc. is of 1B, so no. of locations in a page = $\frac{1\ kB}{1B}$ = 1k. locations.

∴ no. of bit reqd. to address 1024 locs. = 10 bits (2^{10})

∴ offset = 10 bits.

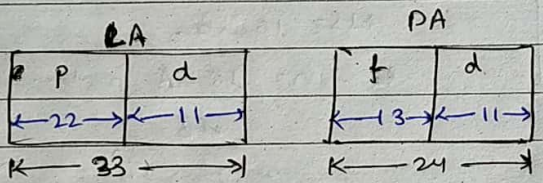
∴



∴ No. of pages = 2^{14}
$= 2^4 \times 2^{10}$
$= 16$ K.

No. of frames = $2^6 = 64$

(g) LA = 33 bits
PA = 24 b
PS = 2 kB.



VM	MM
$2^{33} = 2^{30} \times 2^3$	$2^{24} = 2^4 \times 2^{20}$
$= 8$ GB	$= 16$ MB

Page size = 2 kB.
Byte addressible, hence no. of locs. in each page = 2k.
$= 2 \times 2^{10}$
$= 2^{11}$.

∴ No. of bits reqd. to address each loc. in each page = 11.
⇒ d = 11.

∴ No. of pages = $2^{22} = 2^{20} \times 2^2$
$= 4$ MB.

No. of frames = $2^{13} = 2^3 \times 2^{10}$
$= 8$ K

Disadvantage
be accesed

Way out
A HW co
removes the
when refer

The first
have to a
the correspon
we need it
the S/W stor

The look
have to
TLB is of

CPU

Context
∵ TL
will have
to clear t
Hence t
switches,
using TLB

PAGE:
DATE: / /

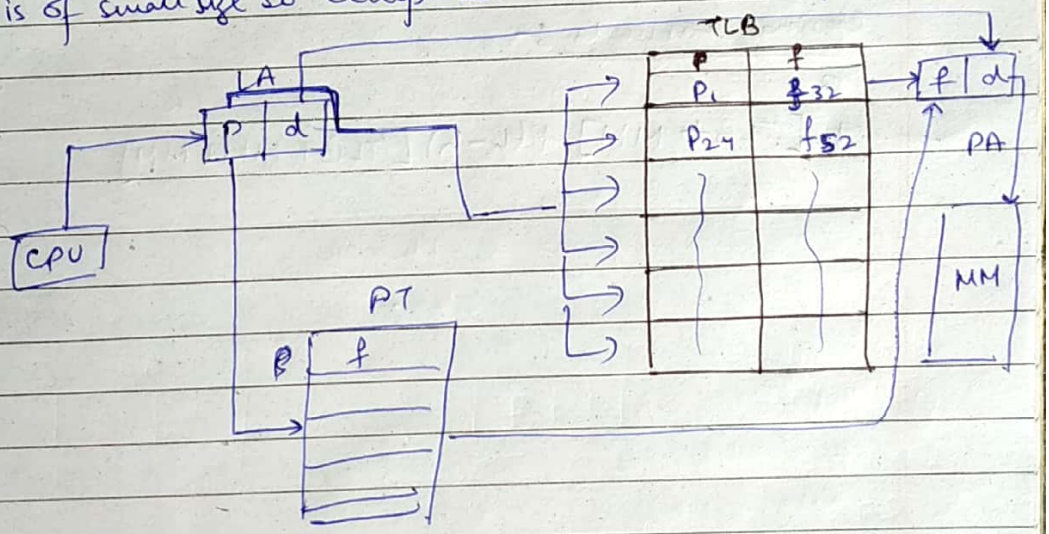
Disadvantage of paging as discussed is that MM needs to be accessed twice, thus increasing the access time.

Way out-

A H/W called TLB (Translation Lookaside Buffer) that removes the need to mention the frame no every time when referencing an inst.

The first time that a page needs to be referenced, we'll have to access the PT, but then we store the page no. and the corresponding frame no. in the TLB so that the next time we need it, we go to the H/W (less access time) and not the S/W stored in MM.

The looking up is done in a set associative form, meaning we have to look up all entries (costly affair) → but the the TLB is of small size so compared to PT, its still feasible.



Context Switch

∵ TLB is a H/W component, we say that every process will have its own TLB. So in case of a context switch, we need to clear the TLB to make space for the new process.

Hence the disadvantage of TLB is that with multiple context switches, we are not able to harness the main advantage of using TLB, i.e, increased speed.

PAGE:
DATE: / /

TLB Numerical

Q) MM access time = 400 μs.
TLB access time = 50 μs.
h = 90% (hit ratio)

Calculate the avg. memory access time.

A) If we do not use TLB, the MM will have to be accessed twice so the avg. access time will simply be (2×400) = 800 μs.

~~0.9 × 50~~

When using TLB, (50 → to get PA, 50 → for TLB miss)

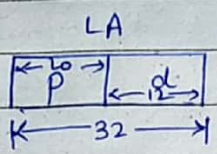
$$0.9\,(50+400) + 0.1\,(50+400+400)$$
$$= (0.9 \times 450) + (0.1 \times 850)$$
$$= 405 + 85$$
$$= 490\ \mu s.\ (\text{instead of } 800\ \mu s\ !!)$$

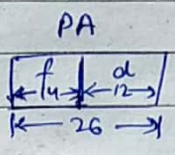
General Formula :-

$$h[TLB + MM] + (1-h)[TLB + MM + MM]$$

Paging Numericals

Q) MM = 64 MB
LA = 32 b
PS = 4 kB
Calculate the tot. space wasted in maintaining the PT?

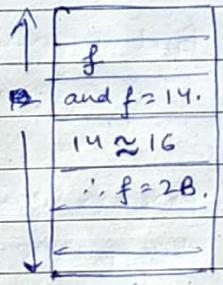




MM (64 MB)
n = 64 M
= $2^6 \times 2^{20}$
= 26.

2^6

Page size = 4 kB.
No. of locs. = 4k = $4 \times 2^{10} = 2^{12}$.
∴ d = 12.
No. of pages in the PT = p = 2^{14}
∴ Size of the PT = $2^{14} \times 2$ B
= 2^{15} B = 2 MB.



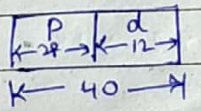
∴ Wastage due to page table = 2 MB.

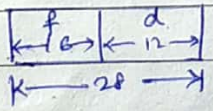
Q) MM = 256 MB
PS = 4 kB.
LA = 40 b
Process size = 4 MB.
Find the space wasted for the PT due to this process.

$\frac{64}{128}$

MM = 256 MB.
n = 256×2^{20}
= $2^8 \times 2^{20}$
= 2^{28}.

∴





Page size = 4kB.

∴ no. of locs. in each page = 4k
= $2^2 \times 2^{10}$
= 2^{12}

∴ d = 12.

∴ p = 28.
f = 16

∴ no. of pages in the page table = 2^{28}
size of the page table = $2^{28} \times 2B \rightarrow f$
= $2^{20} \times 2^9$
= 512 MB

But we need to find wastage due to only one process which is of size 4MB.

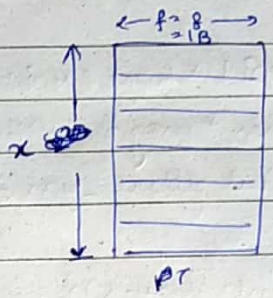
Process size = 4MB.
Size of each page = 4 kB.
∴ No. of pages = $\frac{4MB}{4kB} = \frac{2^{22}}{2^{12}} = 2^{10} = 1k$

∴ Tot. wastage = $1k \times 2B$
= 2kB.

Q) SM = 256 GB
PS = 2kB
MM = 512 kB
PT = 8 kB.
Size of the process?

SM size = 256 GB	MM size = 512 KB
$= 2^8 \times 2^{30} \times 1B$.	$n = 2^9 \times 2^{10}$
∴ $n = 2^{38}$	$= 2^{19}$.
∴ LA = 38 b.	∴ PA = 19 b.

Page size = 2 kB.
$= 2^1 \times 2^{10} \times 1B.$
$\therefore d = 11$ b.

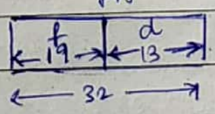


Size of PT = 8 kB.
~~one x^3 Q~~
$\Rightarrow 8\,kB = x \times 1B.$
$\Rightarrow x = 8k.$
~~$2^3 \times 2^{10}$~~
~~2^{13}~~
~~no. of bits required address~~

no. of pages × page size = size of the process page table.
$\Rightarrow 8k \times 2kB = 16\,MB.$

Q) PS = 8 kB
PA = 32b
PT = 24 MB
LA = ?

Page size = 8 kB
No. of pages = 8k
$= 2^3 \times 2^{10}$
$= 2^{13}.$
~~Q pg~~ $\therefore$ tot. no. of locs in a page = 13.
$\therefore d = 13.$



Size of PT = 24 MB. $\Rightarrow$ $24\,MB = n_p \times 19$
$= n_p \times$ pf $\Rightarrow$

PAGE:
DATE: / /

CPU Scheduling

→ A process execution consists of a cycle of CPU execution and I/O execution.

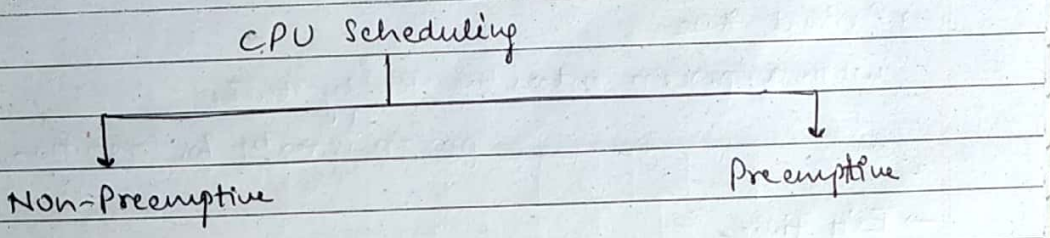
→ Normally every process begins with CPU burst that may be followed by I/O burst, then another CPU burst and then I/O burst and so on. Eventually at the last, will end up on CPU burst.

CPU bound process	I/O bound process
Those processes which require most of their time on the CPU.	These are those processes that requir most of the time on I/O devices or peripherals.
eg→ heavy calculations that req. very less I/O interaction.	eg→ printing of nos, etc

Conclusion

A good CPU scheduling idea should choose the mixture of both so that both the devices and CPU can be utilised efficiently.

~~Approaches towards selecting the suitable algorithm for scheduling~~



Non-Preemptive Approach

When a process has the CPU to itself, and if it is in the running state, we cannot take the CPU from it under any circumstances whatsoever (till it releases the processor (CPU) by itself).

Preemptive Approach

If a process is running but a higher priority process comes in, we will give preference to it.

OS Lab → 10 → 20
MP Lab → 9 → 18
CA Lab → 7 → 14
DBMS Lab → 6 → 12

OS → 10 → 40
ACT → 9 → 36
MP → 9 → 36
CA → 9 → 36
Eco → 9 → 27
DBMS → 7 → 28

Non-preemptive

→ when a process completes its execution.
→ when the process leaves CPU voluntarily to perform some I/O operation or to wait for an event.

Preemptive

→ if a process enters in the ready state either from new or waiting state and it is a high priority process.
→ if a process switches from running state to ready state because the quantum expires. (eg → RR approach → we do not wait for the process to complete if it hasn't completed its op in the specified time)

CPU Terminology

→ Burst time/ Execution time/ Running time.
It is the time the process requires for running on the CPU.

→ Waiting time.
Time spent by a process in ready state waiting for CPU.

→ Arrival time
when a process enters the Ready state.
process is ready for execution.

→ Exit time
when a process completes execution and exits from system.

→ Turnaround time
Total time spent by the process in the system.
TAT = Exit Time (ET) − Arrival Time (AT)
= Burst Time (BT) + Waiting Time (WT)

→ Response time
Time between a process enters the ready queue and get scheduled on the CPU for the first time

PAGE
DATE: / /

Criteria for CPU Scheduling:-

To judge the performance of the various algorithms.

→ **Avg. waiting time**

less avg. waiting time is desirable.

(we cannot change the arrival time, nor the burst time. Through scheduling, we can play around with the waiting time to get best results)

→ **Avg. response time**

less is desirable.

Even if the avg. waiting time remain the same, a lesser response time will give the user the satisfaction of getting a faster response (the page can load slowly etc eg).

→ **CPU utilisation**

More CPU utilisation is desirable ← max. resource utilisation.

→ **Throughput**

No. of processes executed per unit time.

(How much work got done).

PAGE:
DATE: / /

First Come First Serve (FCFS) :-

→ Simplest Scheduling algorithm - ~~it~~ assigns the CPU to the process which comes first.

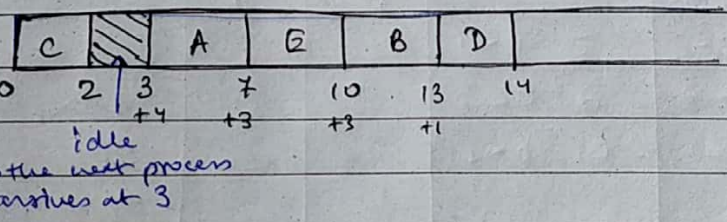
→ Easy to understand and can easily be implemented using Queue data structure.

→ Always non-preemptive in nature.
(we cannot preempt forcefully once a process acquires the processor for itself. The process has to release the CPU voluntarily - no matter the execution time of the running process or the priority of the waiting process)

eg)

P_id	A.T.	B.T	TAT = ET - AT	WT = TAT - BT
A	3	4	7 - 3 = (4)	4 - 4 = 0
B	5	3	13 - 5 = (8)	8 - 3 = 5
C	0	2	2 - 0 = (2)	2 - 2 = 0
D	5	1	14 - 5 = (9)	9 - 1 = 8
E	4	3	10 - 4 = (6)	6 - 3 = 3.

we will first make the gantt-chart based on the AT and the BT provided.
From there, we will go on to complete the table using the gantt-chart.



Now we calculate the Turn around Time (TAT) based on the exit time (getting from the ~~table~~ chart) and the arrival time (from table).

$$\text{Now, avg. TAT} = \frac{4+8+2+9+6}{5} = \frac{29}{5},$$

$$\text{avg. WT} = \frac{5+8+3}{5} = \frac{16}{5},$$

** Starvation

When a process has to wait due to the CPU being biased (happens in case of SJF as processor is given to that process which has the lowest BT and not in a fair way (i.e, FCFS basis)

Convoy Effect

Smaller processes have to wait for a longer time for bigger processes to release CPU.

eg) Scenario - 1

P_id	AT	BT	WT
P_1	0	100	0
P_2	1	1	99

Avg. WT = 49.5

Scenario - 2

P_id	AT	BT	WT
P_1	1	100	0
P_2	0	1	0

Avg. WT = 0.

It is clear that if bigger processes come later on, the avg. waiting time will be much lesser (desirable). But if it is the other way round, i.e, when smaller processes come later, then the efficiency of this algorithm is too low.

Advantages of FCFS:-

→ easy to use, understand and implement.
→ must be used for background processes where execution is not urgent.

Disadvantages:-

→ Suffers from convoy effect
→ Normally higher WT
→ No consideration to priority or BT, so should not be used for interactive system.

Q) Does FCFS suffer from starvation?

A) NO. this is because although the processes has to wait (maybe even for a long time) for the first process to complete its execution, but it is to be kept in mind that P_2 came late (so it is fair). This kind of waiting comes under convoy effect and not starvation.

non-preemptive ↑ Preemptive ↑

Shortest Job First (SJF) / Shortest Remaining Time First (SRTF)

→ Out of all available processes, CPU is assigned to the process having the smallest burst time requirement (no priority, no ~~seniority~~ seniority).
→ If there is a tie, FCFS is used to break the tie.
→ Can be used both with non-preemptive and preemptive approach.
→ Preemptive version (SRTF) is also called as optimal as it guarantees minimal average waiting time.

~~Shortes~~ Preemptive vs Non-Preemptive approach.

If we have assigned the CPU to a process based on the smallest BT, and while it is running, another one comes in that has an even smaller BT. In non-preemptive approach, we cannot take away the CPU from the first process. this is SJF.

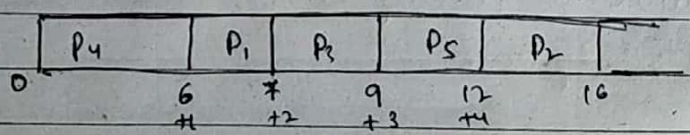
In the preemptive approach, when the 2nd process comes in, if its BT is smaller than the remaining BT of the running process, we preempt it and perform a context switch, giving the CPU to the 2nd process.

Non-Preemptive

TAT = BT + WT.

eg)

P_id	AT	BT	TAT = CT − AT	WT = TAT − BT
✓ P_1	3	1	7 − 3 = 4	4 − 1 = 3
✓ P_2	1	4	16 − 1 = 15	15 − 4 = 11
✓ P_3	4	2	9 − 4 = 5	5 − 2 = 3
✓ P_4	0	6	6 − 0 = 6	6 − 6 = 0
✓ P_5	2	3	12 − 2 = 10	10 − 3 = 7

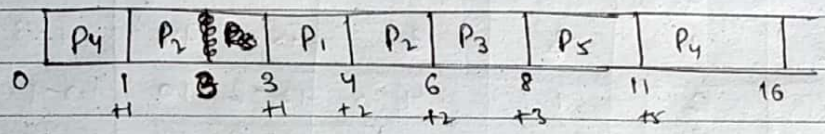


The 1st process is P_4 as we start from time 0. After this we continue according to the smallest BT.

PAGE:
DATE: / /

Preemptive

P_id	AT	BT	TAT	WT
P_1	3	1	4-3=1	1-1=0
P_2	1	~~1~~ 2	6-1=5	5-4=1
P_3	4	2	8-4=4	4-2=2
P_4	0	6	16-0=16	16-6=10
P_5	2	3	11-2=9	9-3=6.



At the 2nd unit, remaining BT of both P_2 and P_5 is same, ie 3 units, so we let P_2 finish.

SRTF is optimal as avg. WT = $\frac{0+1+2+10+6}{5} = \frac{19}{5}$.

SJF WT = $\frac{3+11+3+7}{5} = \frac{24}{5}$.

Advantages :-

→ SRTF (preemptive) guarantees minimal avg. WT.
→ Provides a standard for other algos. in terms of avg. WT.
→ Better avg. response time (RT) as compared to FCFS.

Disadvantage :-

→ Algo cannot be implemented as there is no way to know the BT of a process.
Hence it ~~cannot be implemented~~ is not implementable. It has only been developed to serve as a standard to judge the efficacy of other algorithms.
→ Processes with longer CPU BT requirement will go into starvation.
→ No idea of priority, process with large BT will have poor response time.

PAGE:
DATE: / /

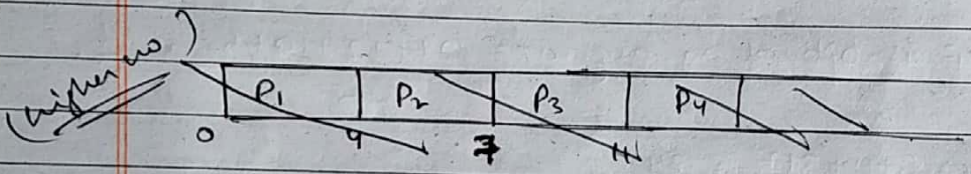
Priority Algorithm

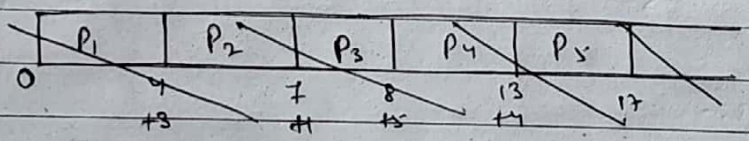
→ Here, a priority is associated with each process.
→ At any instant, out of all available processes, CPU is allocated to the process which possesses the highest priority (number may be higher or lower)
→ Tie is broken using FCFS.
→ No importance is given to AT or BT.
→ Supports both preemptive and non-preemptive version.

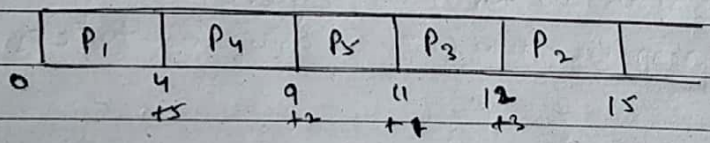
eg)

P_id	AT	BT	Priority	TAT	WT
✓ P_1	0	4	2	4-0=4	4-4=0
✓ P_2	1	3	3	15-1=14	14-3=11
✓ P_3	2	1	4	12-2=10	10-1=9
✓ P_4	3	5	5	9-3=6	6-5=1
✓ P_5	4	2	5	11-4=7	7-2=5.

(higher no)





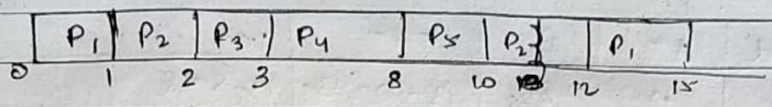


Preemptive.

P_id	AT	BT	Priority	TAT	WT
P_1	0	~~4~~ 3	~~2~~	15-0=15	15-4=11
~~P_2~~	1	~~3~~ 2	3	12-1=11	11-2=9
~~P_3~~	2	1	4	3-2=1	1-1=0
~~P_4~~	3	5	5	8-3=5	5-5=0
~~P_5~~	4	2	5	10-4=6	6-2=4

The basic idea to draw gantt diagram is that whenever a new process enters the system (=AT), we stop and observe the priority no.

PAGE
DATE: / /



Advantages:-

→ Provides a facility of priority specially for system process.
→ Allows to run important process first even if it is a user process.

Disadvantage:-

→ Here process with smaller priority may starve for the CPU.
→ No idea of RT or WT.

Note→ Ageing is a technique of gradually increasing the priority of a process that waits in the system for a long time.

Round Robin Scheduling Algorithm:-

→ This algo is designed for time sharing system, where it is not necessary to complete one process then start another, but to be responsive and divide time of the CPU among the processes in ready state.
→ Here, the Ready queue is termed as Circular queue.
→ We fix a time quantum upto which a process can hold the CPU in one go, within which either the process terminates or must release the CPU and reenter the circular queue and wait for the next chance.
→ RR is always preemptive by nature.

eg)

P_id	AT	BT
P_0	0	5
P_1	1	3
P_2	2	1
P_3	3	2
P_4	4	3

PTO.

PAGE:
DATE: / /

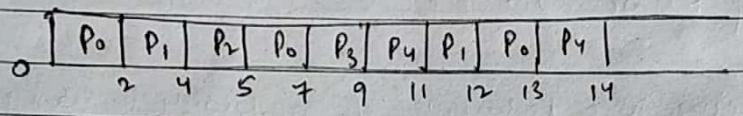
TQ = 2 (given)

eg)

P_id	AT	BT	TAT	WT
~~P_0~~	0	~~5~~ 3	13 - 0 = 13	13 - 5 = 8
P_1	1	~~5~~ 1	12 - 1 = 11	11 - 1 = 10
~~P_2~~	2	1	5 - 2 = 3	3 - 1 = 2
~~P_3~~	3	2	9 - 3 = 6	6 - 2 = 4
P_4	4	~~5~~ 1	14 - 4 = 10	10 - 1 = 9

We maintain a queue

~~P_0~~ P_1 ~~P_2~~ P_0 ~~P_3~~ ~~P_4~~ P_1 P_0 P_4



Advantages:-

→ Performs best in terms of avg. RT.

→ Works best in terms of time sharing system, client server architecture and interactive system.

→ Kind of like SJF implementation. (ultimately the process with the smallest BT will terminate at the earliest).

Disadvantages:-

→ longer processes may starve.

→ Performance depends heavily on time quantum.

(If we keep the TQ low, the no. of context switches will be more. Context switching also takes some time of its own. So a large no. of context switches would lead to the CPU suffering a longer period of idle time. On the contrary, if we choose larger TQs, that would defeat the purpose of RR) ~~add to~~

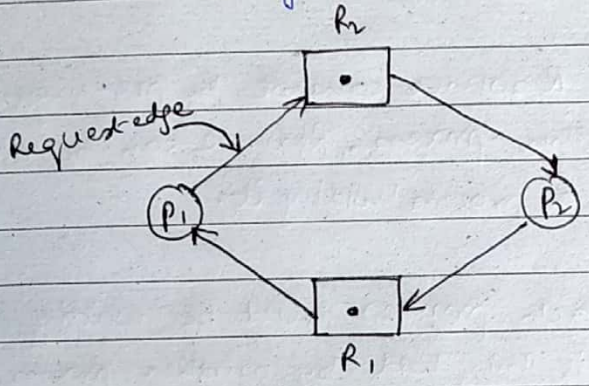
∴ TQ should be chosen very wisely.

→ No idea of priority.

Deadlock

→ In a multiprogramming system, a no. of processes compete for limited no. of resources and if a resource is not available at that instant, then the process enters into Ready state.

→ If a process is unable to change its WS indefinitely because the resources requested by it are held by another waiting process, then the system is said to be in deadlock.



Resource allocation graph.

Square → Resource
Circle → Process
When an arrow goes from a process to a resource, it implies that the process wants it, and if an edge goes from a resource to a process the resource belongs to that process.

This is clearly a deadlock as P_1 needs R_2 but R_2 belongs to P_2, as well as R_1 belongs to P_1 but but P_2 needs R_1 to complete its execution. Both keep waiting for their respective reqd. resources and a deadlock is encountered. (Implying that a cycle is vicious).

System Model (that every process needs to follow → rules)

→ Every process will request for the resource
→ If entertained, then process will use the resource.
→ Process must release the resource after use.

Necessary Conditions for Deadlock:-

1) Mutual Exclusion → Atleast one resource type in the system which is non-shareable, that is, in mutual exclusion mode (one at a time / one by one). eg. printer.

2) Hold and wait → Process is currently holding atleast one resource and requesting additional resources which are being held by other processes.

3) NO pre-emption → Resource cannot be pre-empted from a process by any other process. Resource can be released only voluntarily by a process holding it. **

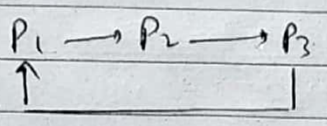
4) Circular wait → Each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

eg) $P_1 \longrightarrow P_2 \longrightarrow P_3$

Here, is mutual exclusion present? Yes, because if P_1 is waiting for P_2 to release a resource that it needs, it definitely means that that resource is non-shareable.
Is hold and wait present? Yes its possible as P_1 and P_2 may be holding some resources and waiting for the additionally reqd. resources in sequence.
Is non-preemption present? Yes definitely. If it wasn't, then the reqd. resources would be automatically allocated to the different processes based on different CPU scheduling algorithms. If forceful preemption was allowed, the P_1 for P_2 and P_2 for P_3 would have snatched the reqd. resources.

Even after these 3 conditions have been satisfied, is deadlock present? No. Because P_3 is not waiting for any resource, it will complete execution and give the reqd. resource to P_2 and so on, and this would lead to overall completion.

But what if. . . .



Now there is circular wait and the system is now in deadlock (most impt: condition).

** Deadlock is encountered only if all 4 conditions are satisfied.

Deadlock Handling Methods

① Prevention → means design such a system which violate atleast one of the four necessary conditions of deadlock and ensures independence from deadlock.

② Avoidance → System maintains a set of data using which it takes a decision whether to entertain a new request or not, to be in safe state.

③ Detection & Recovery → Here we wait for a deadlock to occur and once we detect it, we recover from it.

④ Ignorance → we ignore the problem as if it does not exists.

Deadlock Prevention :-

Mutual Exclusion