No :

Date: / /

01. Introduction.

- **Architecture** is a art and a technique / science of design and building something.
  important bcz → support to understand how components should be interrelated.

- **Computer architecture** is a fundamental study of how to design part of computer inorder to acheive requirements.

tributes - Instruction set, no.of bits used to represent various data types, I/O mechanism, Techniques for addressing memory,

- **Computer organization** - provides information about the linking of different operational attributes of computer system.
  - important bcz → Understand interrelationship between the computer language, org^n of a computer & the design of a computer system.
  - Role → balancing performance, efficiency, cost.

tributes - Hardware details, control signals, interface between compute & peripherals, memory technology used

Computer System Structure (components)

- Abstract view of components of computer)

User
↕
Application programs
↕
Operating System
↕
Computer Hardware.

Operating System

- A program that act as an intermediary between a user of a computer and computer hardware.

Goals. →
- Execute user programs & make solving user problems ease
- Make the computer system convenient to use.
- Use the hardware in an efficient manner.

No: Date: / /

Terms OS covers many roles. ↓

- Because of myriad (no. of) design & uses of Oses.
- Present in toasters through ships, spacecraft, game machines, TVs & industrial control systems.
- Born when fixed use computers for military became more general purpose & needed resource management and program control.

* The 1 program running at all times on the computer is the kernal part of the OS.

History Of OS.

- ✓ First developes in - 1950s to manage tape storage.
- ✓ 1960s - starts to use disks.
  - 1st version of unix os developed.
- ✓ 1981 - 1st OS of Microsoft (DOS).
- ✓ 1981 - GUI was created paired with MS-DOS.

Types of OS.

1. Batch ~~Processing~~ OS
2. Multitasking / Time sharing OS
3. Multiprocessing OS
4. Real Time OS
5. Distributed OS
6. Network OS
7. Mobile OS

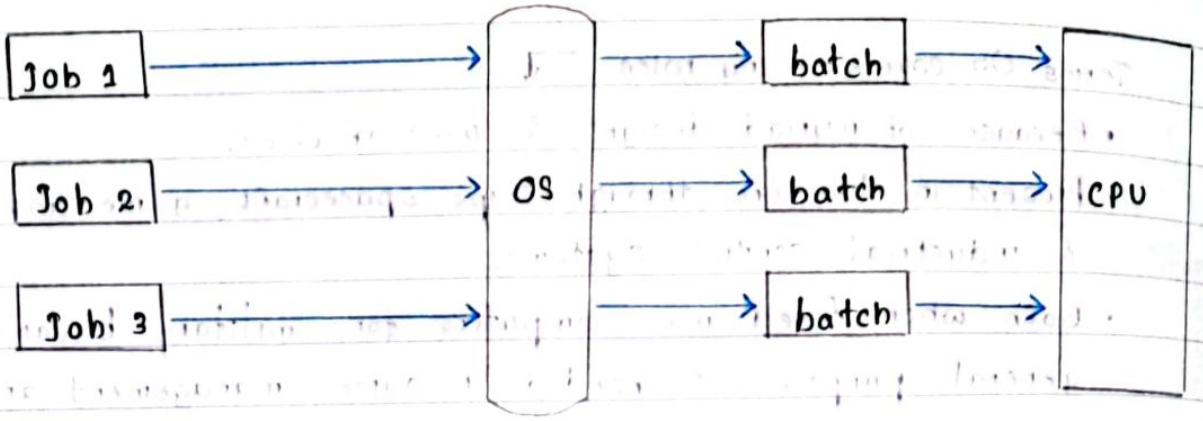
Batch OS

- ✓ Doesnot interact with the computer directly.
- ✓ There is an operator which takes similar jobs having the same requirement and group them into batches.
- ✓ It is a responsibility of operator to categorize jobs with similar needs.

ex: payrall systems, Bank statements.

No :

Date: __/__/__



Advantages.

- More efficient - in multiple jobs with similar needs.
- The idle time ofor the batch system is very less.
- Easy to manage large work repeatedly.
- It is very ~~differen~~ difficult to guess or know the time required for any job to complete. But batch system processors are aware of how long a job will take when it is in the queue.

Disadvantages.

- Expensive.
- Hard to debug.
- Other jobs will have to wait for an unknown time if any job fails.

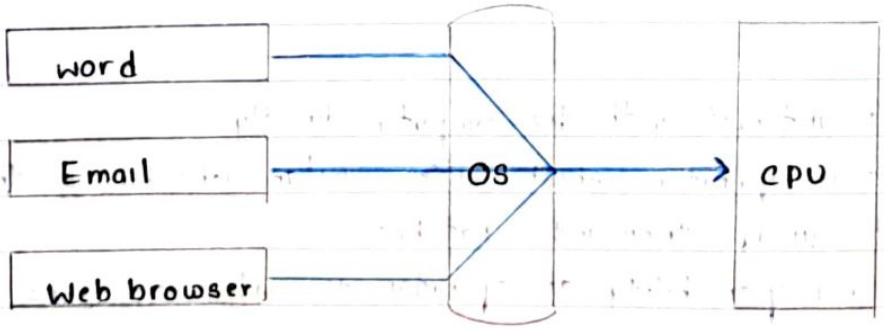
Time sharing OS

- ✓ Each task given some time to execute so that all the tasks work smoothly
- ✓ Each user gets the time of CPU as they use single system. (Multitasking)
- ✓ The task can be single user of different users also.
- ✓ Quantum - time that each task gets to execute.

After the time interval is over OS switches over to the next task.

ex: Multics, Unix.

↑ Good for researches



No: Date: / /

Advantages.

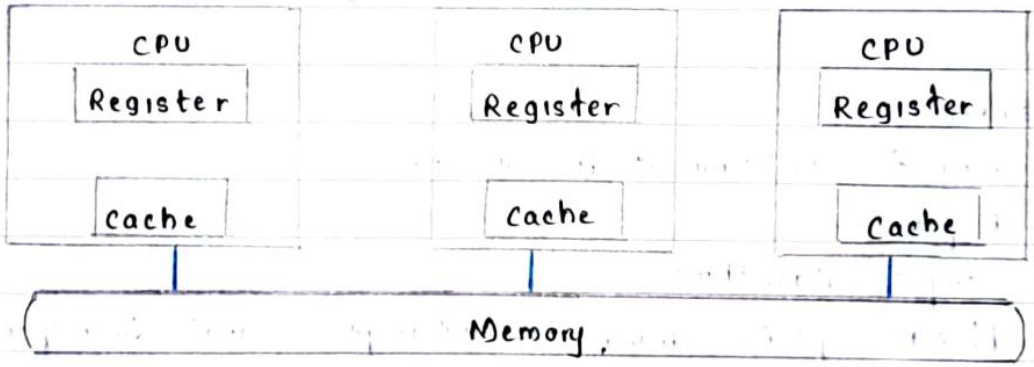
- Time saving - each task gets equal opportunity.
- Fewer chances to duplicate of software.
- CPU idle time can be reduced.
- Resource sharing - multiple users sharing CPU, memory and perpherals.
- Improves productivity - Time sharing allows users to work concurrently.

Disadvantages.

- Need expert programmers.
- Expensive.
- One must have to take care of the security & intergrity of user programs & data.
- Data communication problems.
- Complex to debugging.
- Security risks.

Multiprocessing OS.

- More than 1 CPU is used for the execution of resources.
  ex: Windows NT, 200, XP, Unix.



Advantages.

- Increase the throughput of the system.
- It has several processes, if 1 processor fails, we can proceed with another processor.

Disadvantages.

- Complex to understand.

No :

Date: __/__/____

Real-time OS

- Serve real time systems.
- Time interval required to process & respond to inputs is very small.
  Time interval ⟶ Response time.
- Used when there are time requirements that are very strick like missile systems, air traffic control systems & robots.
  ex: Scientific experiments.
  Medical imaging systems.
  Industrial control systems.
  Weapon systems.

Advantages.

- Maximum system and device usage results in increased output from all resources.
- Time assigned for shifting tasks is very less.
- Focus on running applications & less important on applications that are in the queue.
- Error free.
- Best memory allocation.

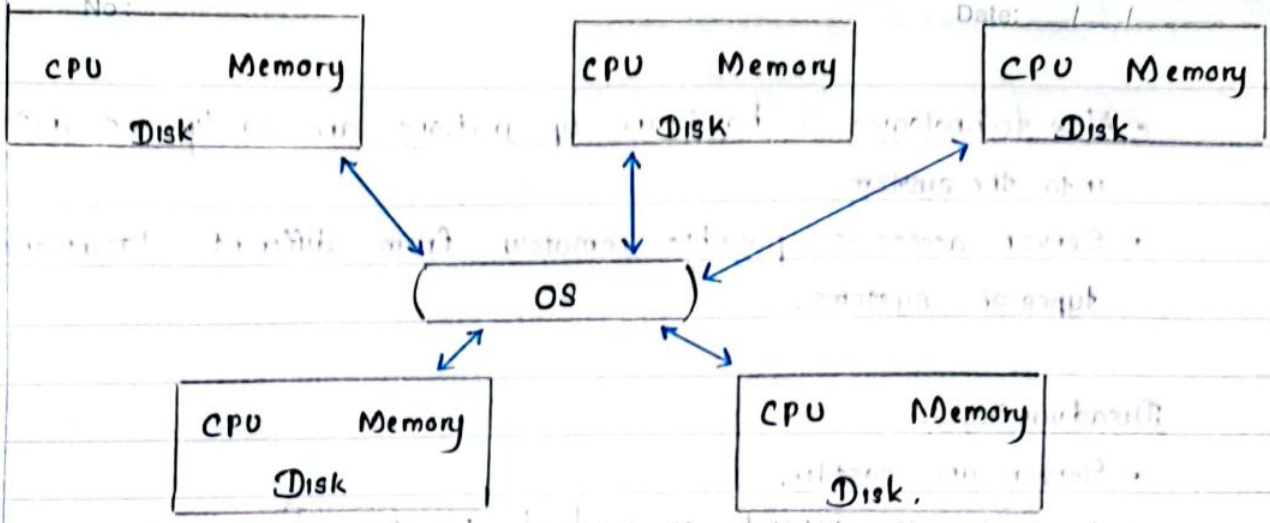
Disadvantages.

- Very few tasks run at the same time.
- Expensive.
- Complex algorithms.
- Needs specific drivers & interrupts signal to respond earliest to interrupts.

Distributed OS.

- Various autonomous interconnected computers communicate with each other using a shared computer network.
- Independent systems posses their own memory unit & cpu. These are refered to as loosely coupled systems or distributed systems.
  ex: Locus.

(devide tasks to individual devices & perform a goal)



Advantages.

* • Always possible for a user to access files or software that are actualy pe present on another system connected to the network, or remote access.
  • Failure of 1 will not affects the other network communication.
  • Electronic mail increases the data exchange ~~rates~~. speed.
  • Computation is high fast & durable. (bc2 resources are shared)
  • Scalable (bc2 many systems can easily added to network.
  • Delay in data processing reduces.

Disadvantages.

  • Failure of the main network will stop the entire communication.
  • Expensive.
  • Complex & not understood well.

Network OS

- Provide a server based system.
- These types of Oses allow shared access to files, printers, security, applications & other networking functions over a small private network.
- All the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc.

Advantages.

  • High stable centralized servers.
  • Security concerns are handled through servers.

No : Date: __/__/__

- New technologies & hardware up-grations are easily integrated into the system.
- Server access is possible remotely from different locations & types of systems.

Disadvantages.

- Servers are costly.
- User has to depend on central location for most operations.
- Maintenance & updates are required regularly.

Ex : Microsoft window server 2003 / 2008, UNIX, Linux, Mac OS X, Novell NetWare.

Operating System Design & Implementation.

① Design Goals.

② Requirements → User
→ System

③ Mechanism & policies.

- Policies - what to be done / How to acheive requirements.
- Mechanism - How it is to be implemented.

* If properly seperated & implemented, policy changes can be easily adjusted without re-writting the code, just by adjusting parameters or possibly loading new data / configuration files.

④ Implementation.

- Earlier days OS are written in assembly language.
  - Provided direct control over hardware related issues, but tie to a particular hardware platform.
- Recent advanced in compiler efficiencies mean that most modern OSes are written in C, or more recently; C++ or high-level language.

A hardware or software that enables 1 computer system to behave like another computer system.

- OS may be developed using emulators of the target hardware, particular if the real hardware is unavailable or not a suitable platform for development.

Architecture & structure.

- The core software components of an OS are collectively known as the kernal.

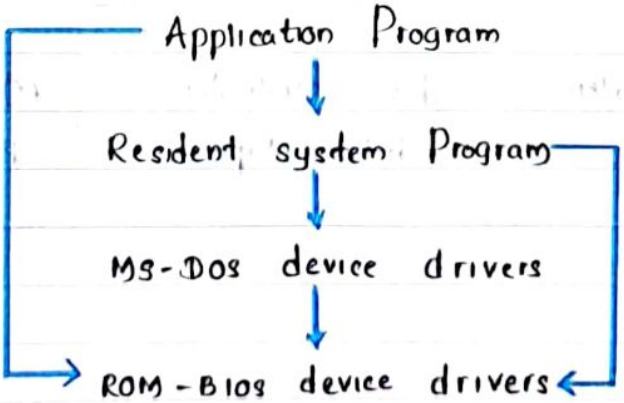
Kernal → Basic codes in the OS. (essencial part)

- kernal has unrestricted access to all the resources on the system.

Kernal.

→ Central component (co-component)
→ Manages operations of computer & hardware.
→ Basically manages operations of memory & cpu time.
→ It is core component of an OS.
→ Kernal act as a bridge between application & data processing performed at hardware level using inter-process communication & system calls.
→ Every OS has a kernal.

Simple structure

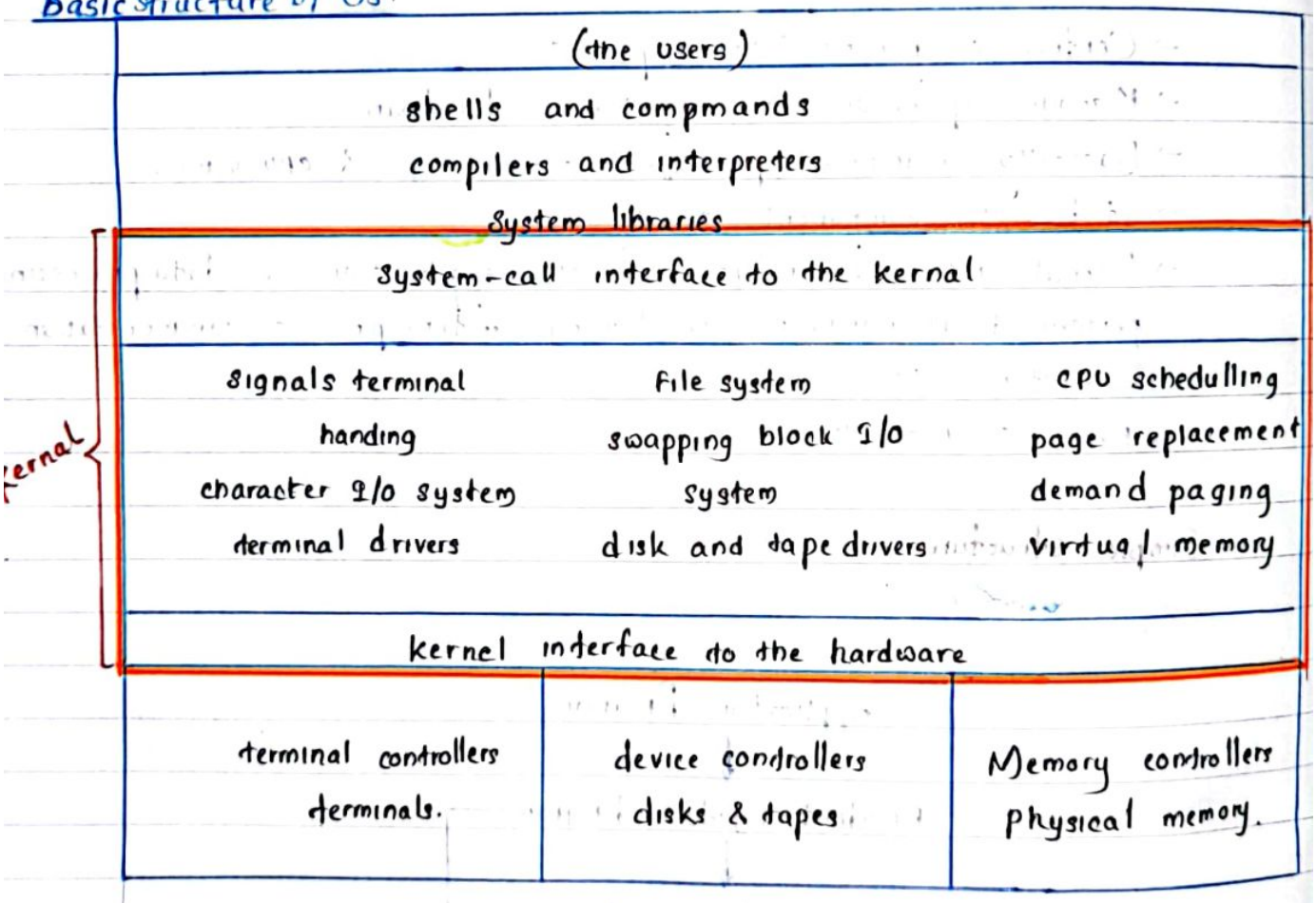


- When DOS was originally written - no idea how big and important it would eventually become.
- A few programmers - a relatively short amount of time - without the benefit of modern software engineering techniques.
- Not break the system into subsystems.
- No distinction between user & kernal modes.
- Allowing all programs direct access to the underlying hardware.

* Users are directly accessible to hardware or kernal level & also multiple access is not allowed. So this structure is not efficient or secured. *

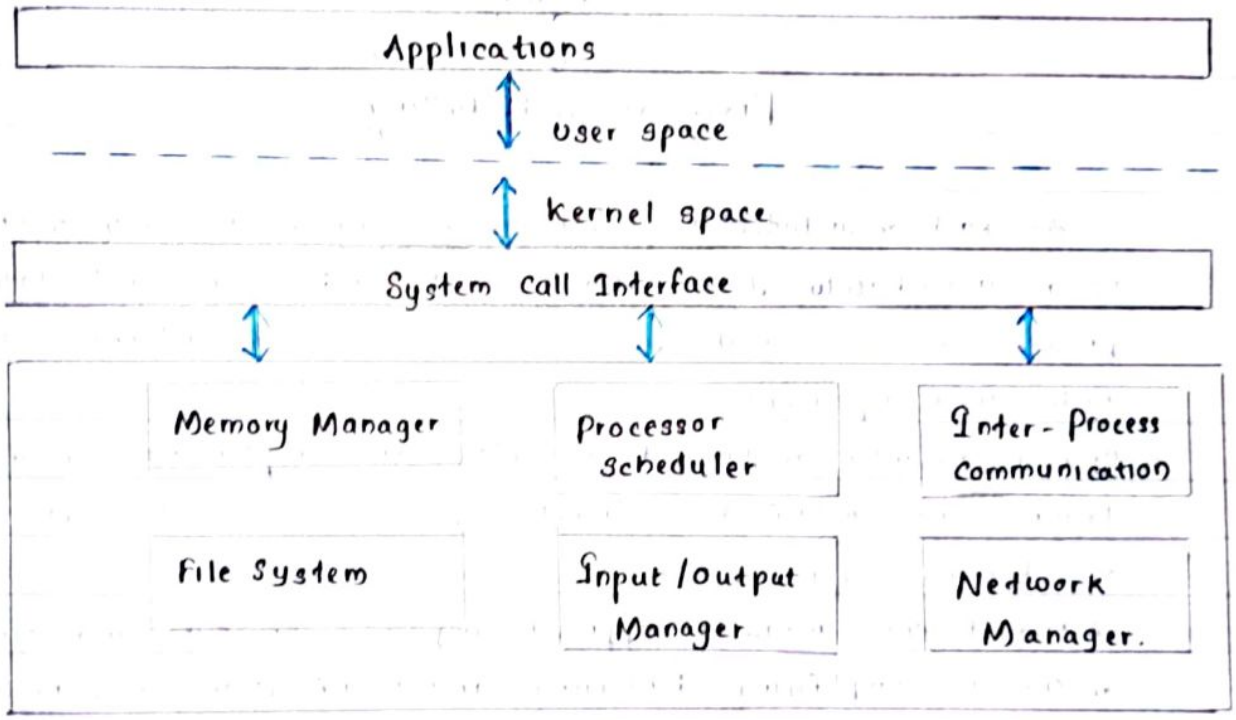
[no any distinction between user level or kernel level → no secured]

Basic structure of OS.



Monolithic OS - not good solution

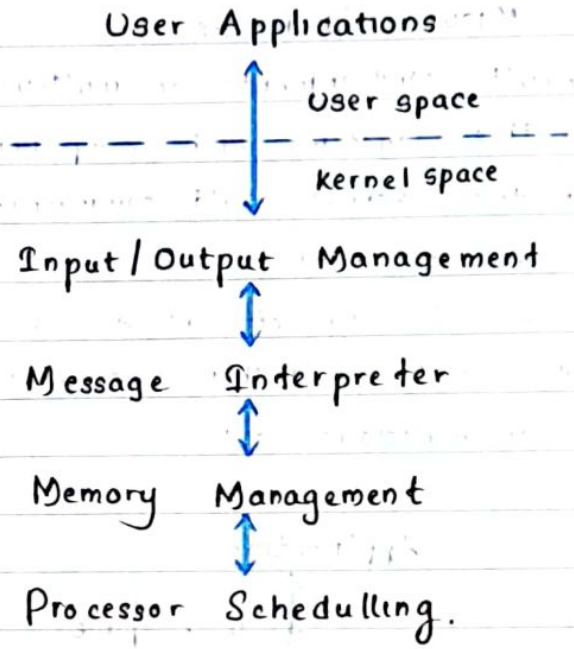
- One of the oldest kind of OS structure.
- Problem : applications can,
  - crash OS software,
  - crash another application,
  - long cpu time,
  - Misuse I/O devices.
- Not good for fault containments or multiusers.
- In early monolithic systems, each component of the OS was contained within the kernel, could communicate directly with any other component, and had unrestricted system access.
- While this made the OS very efficient, it also meant that errors were more difficult to isolate, and there was a high risk of damage due to erroneous or malicious code.



No: Date: / /

Layered OS.

- As OS become larger and more complex, this approach was largely abandoned in favor of a modular approach which grouped components with similar functionality into layers to help OS designers to manage the complexity of the system.



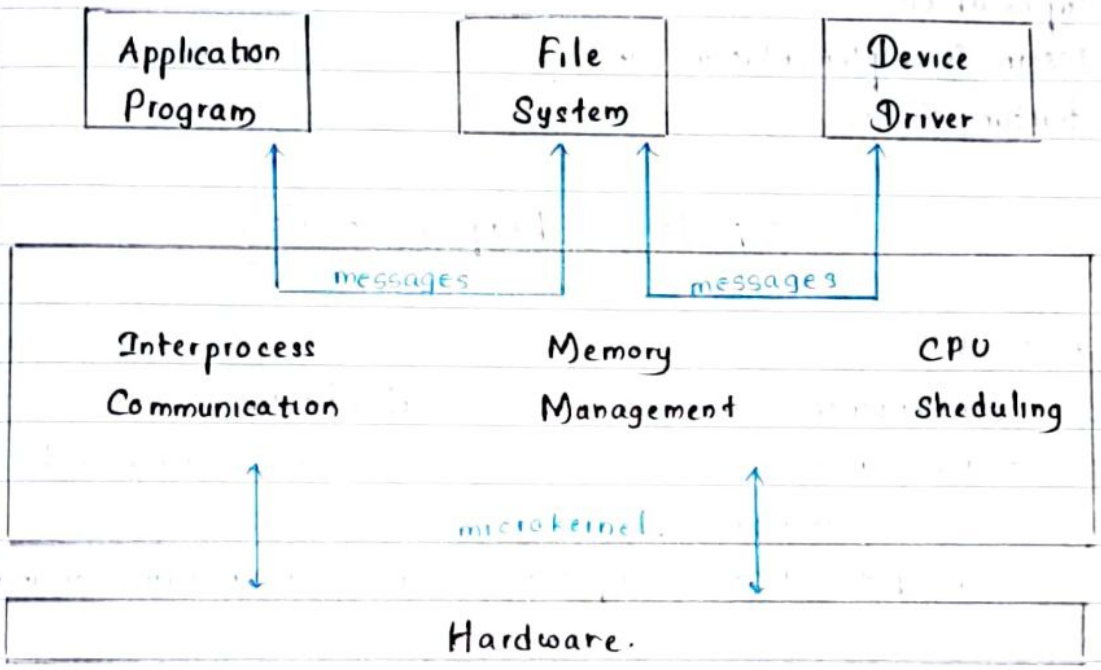
- In this kind of architecture, each layer communicates only with the layers immediately above and below it, and lower-level layers provide services to higher-level ones using an interface that hides their implementation.
- The modularity of layered OS allows the implementation of each layer to be modified without requiring any modification to adjacent layers.
- Although this modular approach imposes structure and consistency on the OS, simplifying debugging and modification, a service request from a user process may pass through many layers of system software before it is serviced and performance compares unfavorably to that of a monolithic kernel.

No:

Date: / /

Microkernel Architecture

- Include very small no.of services within the kernel in an attempt to keep it small & scalable.
- The service typically include low-level memory menagement, inter-process communication & basic process synchronization to enable processes to cooporate.
- In micro kernel designs, most OS components, such as process management & device management, execute outside the kernel with a lower level of system access.
- Avoid communication cost.
- Simplify the kernel without complicated.



Modules

- Modern OS development is object-oriented, small core kernel and a set of modules (linked in dynamically)
- Modules are similar to layers: has clearly defined tasks & interface.
- Modules are free to contact any other module.
- Eliminating the problems of going through intermediary layers.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly.

No:

OS - provide security

Date: / /

Core Solaris kernel

- Schedulling classes
- File systems
- Loadable system calls
- Executable formats.
- STREAMS modules
- Miscellaneous modules
- Device & bus drivers.

Summary:

① What is OS
② Types of OS
③ Design & implementation of OS
④ Structure of OS.

02 Role & Purpose of OS.

✓ What OS do?

- User convenience
  - Easy to use
  - Good performance
  - Help to share computers such as mainframe or minicompu
- Hardware
  - managing activities of

Purposes

OS provides an environment for execution of programs and services to programs and users.

- User Interface - CLI, GUI, touch-screen, Batch
- Program execution - The system must be able to load a program into memory and to run that program, end execution
- I/O operations - Support communication between internal & external env.
- File system manipulation - Read & write files and directories, create and delete them, search them, List file information, permission management.

- **Communication** - Processes may exchange information, on the same computer or between computers over a network.
- **Error detection** - OS needs to be constantly aware of possible errors.
- **Resource Allocation** - When multiple users or multiple jobs running concurrently resources must be allocated to each of them. (OS like a 24 hr convenience)
- **Logging** - Keep track of which users use how much and what kinds of computer resources.
- **Protection & Security** - The owners of information stored in a multiuser or network computer.

System Call.

- Passing the messages from user to OS

User Application (User level)
↕ system call
OS (Kernel) supported by API

- It is a method of interaction with the OS through the system programs
- It is a technique in which a computer system program requests a service from the OS kernel.
- The Application Program Interface (API) helps to connect the OS functions with user programs. It serves as a bridge between a process and the OS, enabling user-level programs to request OS accessed using the kernel system & any software that consumes resources must use system calls.

Types of system call

1. Process control
2. File management
3. Device management
4. Information Maintanance
5. Communication.

- Process Control - It is responsible for ~~file~~ manipulation of jobs, including creating ~~files~~, deleting ~~files~~, reading, opening, writing, closing ~~etc~~ of jobs.

- File management - It is responsible for file manipulation of jobs, including creating files, deleting, reading, opening, writing &

- Buffer - A small amount of memory / use to keep a small job queue memory
- device drivers - Basic essential instructions to particular device.

✓ Device Management - These are responsible for device manipulation including reading from device buffers, writing into device buffers, etc.

✓ Information Maintanance - These are use to manage the data and its share between the OS and the user program.
Some common instances of information maintanance are getting time, or date, getting system data, setting time or date, setting System etc.

✓ Communication - These are used for interprocess communication (IPC)
Some examples of IPC are creating, sending, receiving messages, deleting communication connection, etc.

System Program.

Language translators - compiler, interpreters, assemblers
device drivers
OS
utility programs - cleaner, antiviruses

✓ A system program offers an environment in which programs may be developed and run.

✓ In simple terms the system programs serve as a link between the user interface (UI) and system calls.

✓ Some system programs are only user interfaces and others are complex

Types of the System Program.

1. File management
2. Status information
3. File modification
4. Programming Language support
5. Program loading & execution
6. Communication.

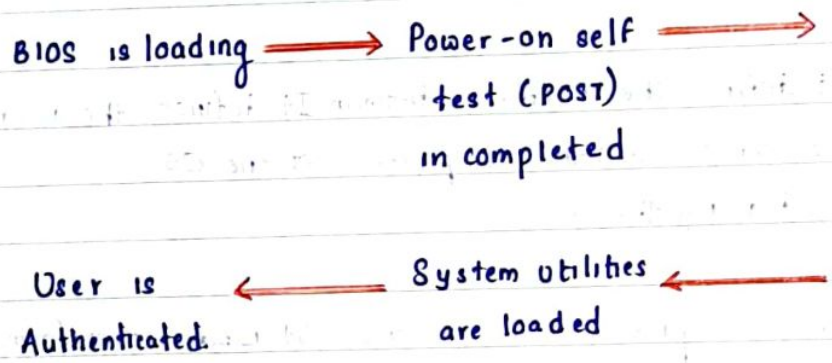
Features	System call	System Program.
Definition	It is a technique in which a computer system program request a service from the OS kernel.	It offers an environment for a program to create and run.
Request	It fulfils the low-level requests of the user program.	It fulfils the high-level request or requirement of the user program.
Programming Languages	It is usually written in C & C++ programming languages. Assembler level language is used in system calls where direct hardware access is required.	It is commonly written in high-level programming languages only.
User view	It defines the interface between the services and the user process provided by the OS.	It defines the user interface of the OS.
Action	The user process requests an OS service using a system call.	It ~~random~~ transforms the user ~~requiremen~~ request into a set of system calls needed to fulfil the requirement.
Classification	It may be categorized into file manipulation, device manipulation, communication, process control, information maintenance, and protection.	It may be categorized into file management, program loading and execution, programming-Language support, status information, file modification and communication.

Firmware - contains in ROM

System Boot

- Booting is the process of starting a computer.
- It can be initiated by hardware such as a button press or by a software command.
- After it is switch on, a CPU has no software in its main memory, so some processes must load software into memory before execution. This may be done by hardware or firmware in the CPU or by a seperate processor in the computer system.
- Restarting a computer also called rebooting, which can be "hard", eg: after electrical power to the CPU is switched from off to on or "soft": where the power is not cut.

Booting Process.



Network OS Vs Distributed OS.

- Both Network OS & Distributed OS works on multiple systems/nodes. The main difference between a network OS and a distributed OS is the way they handle resources and communication between devices.
- A network OS is primarily concerned with managing resources and communication within a single network, while a distributed OS is designed to manage resources & communication across multiple networks.

No: Date: / /

Edge
Grid
cluster computing

Network OS

A type of OS that is used to run a system software on a server & allow the server to manage the users, groups, data, security applications & other networking operations.

A network OS permits resources sharing among 2 or more computers that are operating under their own OS.

Distributed OS

A distributed OS is an Operating system that run on multiple machines and provides the appearances of single system to the users.

It is designed to allow multiple computers to work together as a single system with each machine in the system running its own instance of the OS & contributing its own resources to the system.

Client - Server OS.

The term "client/server" describes a type of distributed processing in which an application is divided into 2 parts, each possibly residing on separate OS, but working together to provide a service to the end user.

Tutorial 01

① Differences between Network OS & distributed OS.
② Explan how networked, client-server distributed OS are different from single node OS.

Summary:

① Role and purpose of OS
② System call / System Program / System Boot.
③ Network OS vs Distributed OS
④ Client server OS.

Program - Just a instructions store the memory
process - Active / running instructions.

SCI - Hub
Springer / Academia

03. Process Management

An OS executes a variety of programs:

- Batch system - job
- Time shared systems - user programs or tasks.

Process - A program is execution; process execution must progress in sequential fashion. (active)

Program - Just an instruction that stored in the memory. (passive)
(Executable files but not executing.)

* Program becomes process when executable file loaded into memory.

When program converts to a process there are multiple parts.

stack (dynamic) ↓
↑
heap (dynamic)
data (variables/constances)
text (instructions)

- The program code : text section.
- Current ~~action~~ activity including program counter, processor registers.
- stack ~~coun~~ containing temporary data (function parameters return addresses, local variables)
- Data section containing global variable.
- Heap containing memory dynamically allocated during run time

* One program can be several processes.
- Multiple users executing the same program

Process state

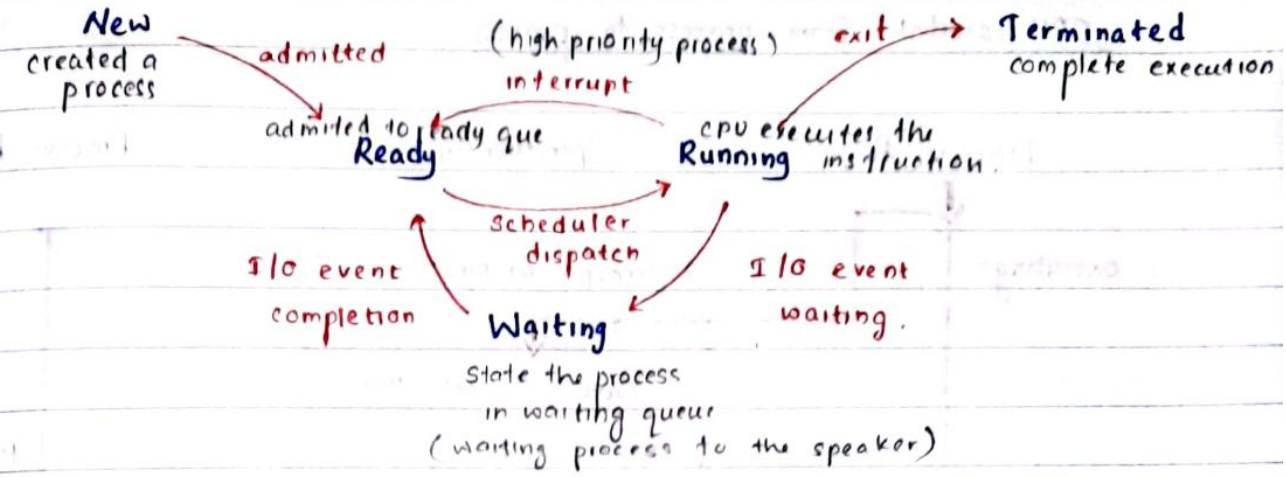
- New : The process is being created.
- Running : Instructions are being executed.
- Waiting : The process is waiting for some event to occur.
- Ready : The process is waiting to be assigned to a Processor
- Terminated : The process has finished execution.

- I/O request → I/O queue → I/O → ready queue
- time slice expired (process එකේ time එක expire වුණාම) → ready queue
- fork a child (ඒ process එක ඇතුළෙම වෙන්න වෙන process එකක් හදා ගැනීම) → execute the child → ready queue
- wait for an interrupt → interrupt occurs → ready queue.

No: state

Date: __/__/____

Process transition diagram.



When a new process created it will be in the new state. Then it will admitted to the ready queue Then ~~it~~ after perform a scheduler dispatching the process will change to running state, which is ~~the~~ to the process of CPU execution of the instructions. When this is happen, if the interrupt may occur it will go back to the ready state. Also when I/O or event occur. ~~placed~~ the programe state changes to waiting que, after it complete it will go back to ready queue. ~~to~~ Finaly when the execution has completed the program or the process will change to terminated state. and exit

Process Control Block (PCB)

(A block of memory where we keep information related to a process)

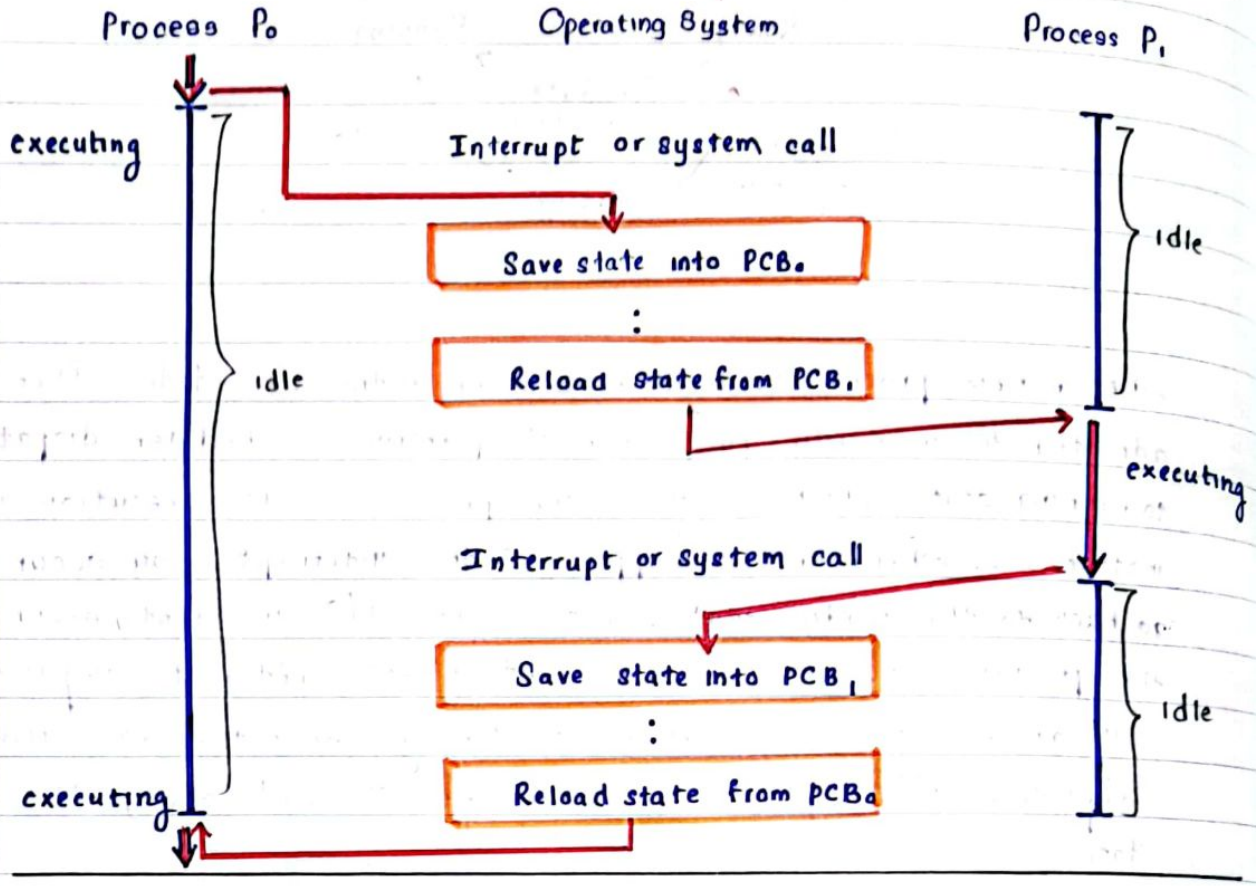
- A block of memory where we keep information related to a process.
- Each an every process has a PCB.
- What includes in PCB,

Process state	• Process state	- running, waiting etc.
Process num	• Process counter	- location of instruction to next execute.
Program counter	• CPU register	- Contents of all process-centric registers.
Registers	• CPU shedulling information	- Priorities, schedulling queue pointers.
Memory limits	• Memory-management infor.	- memory allocated to the process.
List of open files	• Accounting Information	- CPU used, clock time elapsed since start, time limits.
...	• I/O, startup Information	- I/O devices allocated to process ~~Hm~~ list of open files.

No :

Date : / /

CPU switch from process to process



- This diagram shows how the OS will switching among processes
- When P_0 is executing in the CPU currently & meanwhile if P_1 arrived the process of P_0 will be stepping to the waiting stage in process state transition diagram. (running → waiting)
- This may be due to a interupped occured to execute P_1.
- So to do this first PCB of P_0 will save the state of the P_0 process
- And then the new processes PCB will reload and from that cpu will understand what stage did P_1 would start to execute.
- Till the P_1 executes for allocated time the P_0 will stay idle.
- Then, if again P_0 wants to execute, the OS will step P_1 to waiting stage by occuring an interrupt.
- Then again it will save the state of P_1 to its PCB & recall the PCB of P_0.

Select next process.

Process Scheduling - multiple processes scheduling to the cpu that it don't idle (cpu busy)

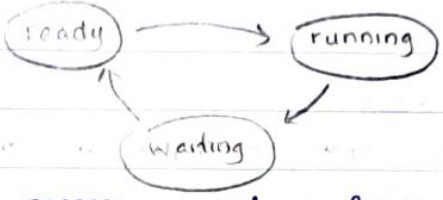
- Maximize the cpu use, quickly switch processes onto cpu for time sharing.
- The process of, scheduling multiple processes to the cpu, that the cpu want be in idle state at any time.
- Selecting the next process imediately to make cpu as bussy to make the processer efficient.
- Use multiprogramming to improve maximum cpu utilization.
- Process scheduler, selects among available processes for next execution on cpu.

Process scheduling queues.

new → ready

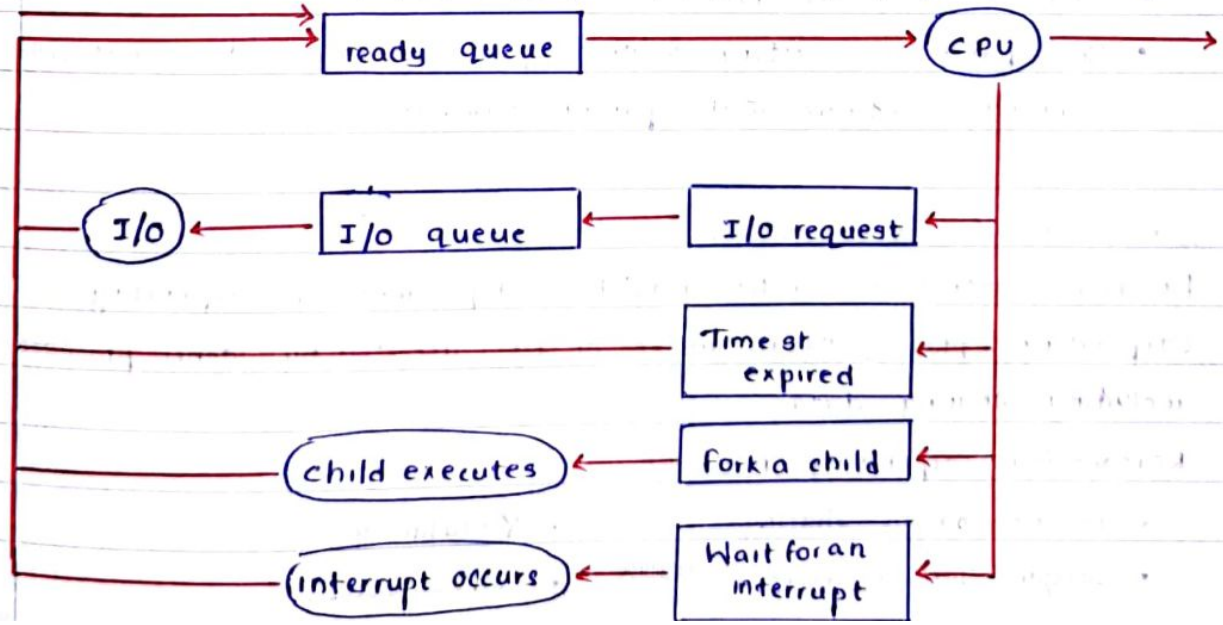
① Job queue - set of all processes in the system.

② Ready queue - set of all processes residing in main memory, ready & waiting to execute.



③ Device queue - set of processes waiting for an I/O device.

Representation of process scheduling.



No: _____ Date: __/__/__

Process Creation

- Parent process create children processes, which in turn create other processes, forming a tree of processes.
- Generally, process identified & managed via a process identifier (pid)
- Resource sharing options.
  1. Parents and childs share all resources.
  2. Children share subset of parent's resources.
  3. Parent and child share no resources.

- Execution options.
  1. Parent & children execute concurrently.
  2. Parent waits until children terminate.

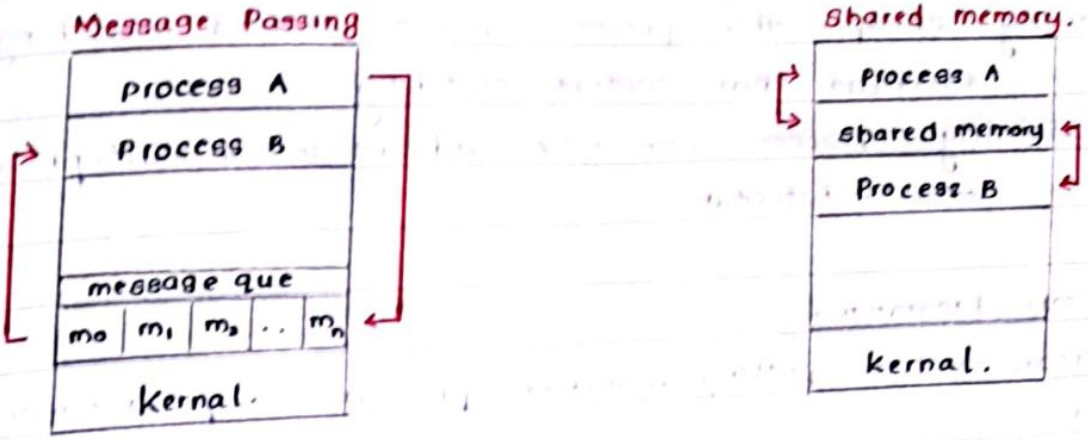
Process Termination

- Process executes last statement and then ask the OS to delete it using the exit() system call.
- Parent may terminate the execution of children processes using the abort() system call. Some reasons are,
  - child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - The parent is exiting & the operating system does not allow a child to continue if its parent terminats.

Inter-Process Communication

- Processes within a system may be independent or corperating.
- Corperating process - can affect or be affected by other processes, including sharing data.
- Reasons for corperating processes,
  - To information sharing
  - Computation speedup.
  - Modularity
  - Convenjences.

- Corporating processes need inter-process communication (IPC).
- 2 models of IPC
  1. Shared memory - putting the work into a common memory pool
  2. Message passing - direct communication. (1 to 1)



Synchronization

- Message passing may be either blocking or non-blocking.
  1. Blocking - Synchronous. - Waiting for the response & going to the other.
  2. Non-blocking - Asynchronous. - speed of communication ⇑ not reliable. (no waiting until the reply comes)

Threads (Thread vs process)

- The basic unit of execution in the CPU.
- The data taken by a thread will be executed by CPU at a time.
- Multithreading : Can work multiple threats at once.
- Multiple tasks with the application can be implemented by separate threads.
  - update display.
  - fetch data.
  - Spell checking.
  - Answer a network request.
- A bulk (heavy-weight) process is, devideded into multiple tasks/threads, so these threads are light-weighted.

threads do

be.

No:

Date: / /

Benefits of threds.

1. Responsiveness - May allow continued execution if part of process is blocked, especially important for user interface.
2. Resource sharing - threads share resources of process, easier than shared memory or message passing.
3. Economy - cheaper than process creation, thread switching lower overhead than context switching.
4. Scalability - Process can take advantage of multiprocessor ~~artic~~ architecture.

Multicore Programming.

Multicore systems putting ~~resources~~ pressure on programmers, challenges include,

- Devide activities
- Balance
- Data spliting
- Data dependency
- Testing & debugging.

This will done by,

① Parallelism - implies a system can perform more than 1 task simultaneously.

② Concurrency - When 2 or more tasks can start, run & complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant.

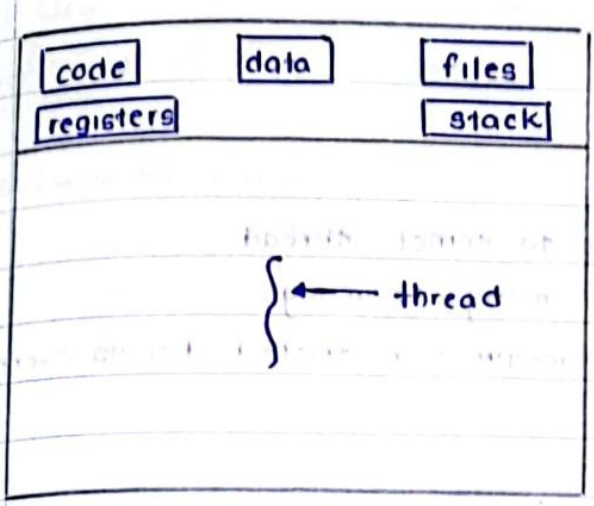
Types of parallelism

- Data parallelism - distributes subsets of the same data ~~acc~~ across multiple cores, same operations on each.
- Task parallelism - distributes threads across cores, each thread performing unique operation.

User No:

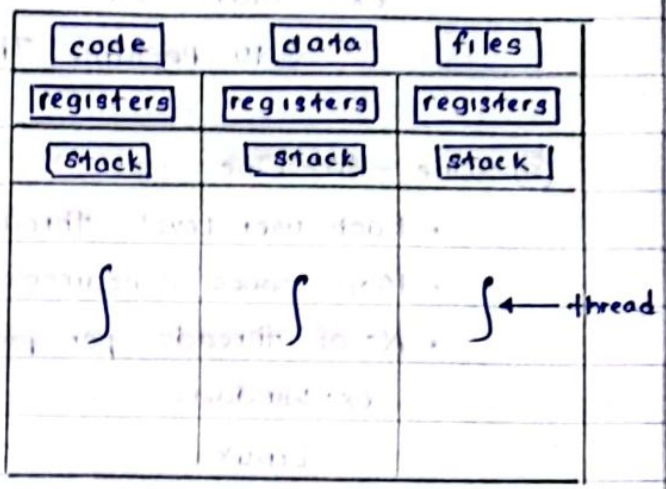
* {concept} of threads

Single - threaded process



- These processes are having individual code, data, file as well as registers and stack.

Multithreaded Process



- These processes are having common code, data, files & registers and stacks are assign * individualy to each thread.

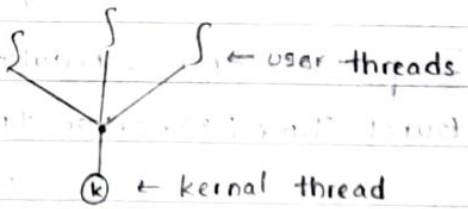
Types of threads.

① User threads - management done by user level threads libraries.
- 3 primary thread libraries.
  - POSIX Pthreads
  - Windows threads
  - Java threads.

② Kernel threads - Supported by the kernel.
- All general purpose os including; windows, Solaris, Linux, Mac os x

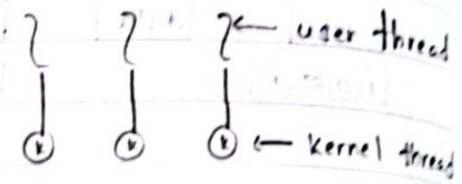
* Usually user threads must be mapped to the kernel threads.

Thread mapping techniques.



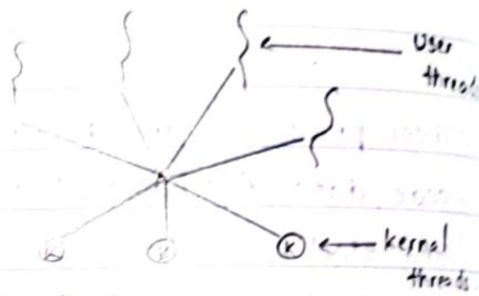
① Many - to - One.
- Many user-level threads mapped to single kernel thread.
- Multiple threads may not run in parallel on multicore system because only 1 may be in kernel at a time.

- Few systems currently use this model.
  ex: Solaris Green Threads.
  GNU Portable Threads.



② **One - to - One**

- Each user-level thread maps to kernel thread.
- More ~~conca~~ concurrency than many-to-many.
- No. of threads per process sometimes restricted due to overhead
  ex: Windows
  Linux
  Solaris 9 & later



③ **Many - to - many** model

- Allows many ~~kern~~ user level threads to be mapped to many kernel threads.
- Allows the OS to create a sufficient number of kernel threads
  ex: Solaris prior to version 9
  Windows with the ThreadFiber package.

Thread cancellation

- ✓ Terminating a thread before it has finished.
- ✓ Thread to be cancelled is target thread.
- ✓ 2 general approaches
  1. Asynchronous cancellation - terminates the target thread immediatly
  2. Deferred cancellation - allows the target thread to periodically check if it should be cancelled. ([illegible])

04 Process Management

Important CPU scheduling terminologies

1. **Burst Time / Execution time** : Time required by a process to complete execution. (running time).
2. **Arrival Time** : When a process enters in a ready state.
3. **Finish Time** : When a process complete & exit from a system.
4. Multiprogramming : A no. of programs which can be present in memory at the same time.

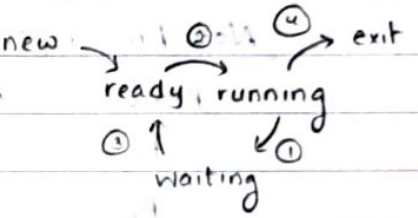
No :

Date: __/__/____

5. Jobs : Type of program without any kind of user interaction.
6. User : kind of program having user interaction.
7. Process : The reference that is used for both job & user.
8. CPU I/O burst cycle : Characterizes process execution, which alternates between CPU & I/O activity. CPU times are usually shorter than the time of I/O.
(CPU burst - must be in a process.
I/O burst - may or may not be in a process.)
Also this is a process of process execution consists of a ~~cycle~~ cycle of CPU execution & I/O wait.

CPU Scheduler

- The CPU scheduler selects from among the processes in ready queue, & allocates a CPU core to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. switches from ready to running state.
  3. switches from waiting to ready.
  4. Terminates.

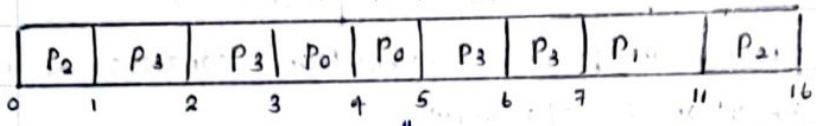


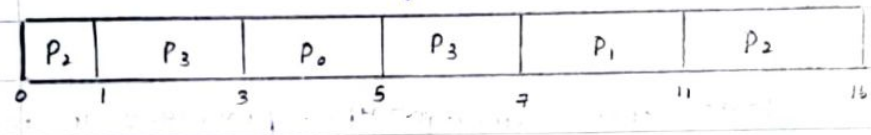
Types of scheduling.

Pre-emptive	Non-Pre emptive.
- Stopping a process when it is in the running state, to access the new process.	- Whole process which is currently in running state will ~~be~~ need to be turminated to start the next.
- It gives priority to the smallest burst timed process to be execute ~~depen~~	

Pre-emptive

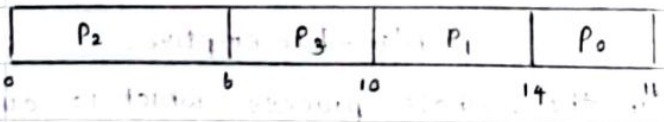
process	Arival Time	Burst Time
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4





Non Pre-emptive

Process	Arival time	Burst time
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4



Scheduling criteria

- **CPU utilization** - keep the cpu as bussy as possible.
- **Throughput.** - No. of processes that complete their execution per time unit.
- **Turnaround time** - Amount of time to execute a particular process.
- **Waiting time** - Amount of time a process has been waiting in the ready queue.
- **Response time** - Amount of time takes from when a request was

submitted until the first response is produced.

if,

CPU utilization ↑ good
Throughput ↑ good
Turnaround time ↓ good
Waiting time ↓ good
Response time ↓ good.

Scheduling algorithm optimization criteria.

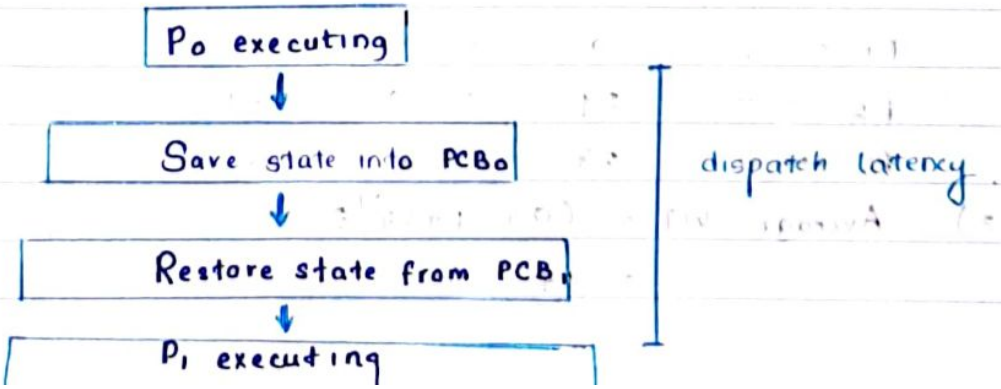
- Max CPU utilization
- Max throughput
- M turnaround time
- Min waiting time
- Min response time.

TT = CT - AT	AT - Arival Time
WT = TT - BT	CT - Completion time
TT = WT + BT	TT - Turnaround time
CT = AT + TT	BT - Burst Time
	WT - Waiting time

Dispatch

- Dispatcher module gives control of the CPU to the process selected by the cpu scheduler;
- This involves
  1. context switching
  2. switch user modes
  3. Jumping to the proper location in the user program to restart that program.

* Dispatch latency – time it takes for the dispatcher to stop 1 process & start another running.



No :

Date: __/__/__

Types of cpu scheduling algorithm.

1. First come first serve (FCFS)
2. Shortest - Job first (SJF)
3. Shortest remaining time.
4. Round robin scheduling.
5. Priority schedulling.
6. Multilevel Queue.

dis→ coinvoy effect

FCFS

- Non - preemptive
- When a process exits it's all threads & giving chance to another process.

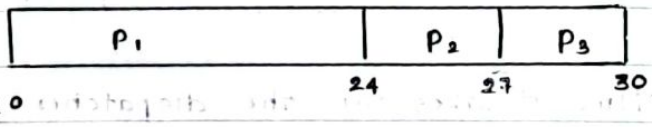
Ex:

Process	Bust Time
P_1	24
P_2	3
P_3	3

Suppose that process arive in the ~~older~~ order P_1, P_2, P_3

1) Create the grantt chart
2) Get waiting time.
3) Average waiting time.

1)



2) WT = Execution time - Arival time

$P_1 \longrightarrow 0 - 0 = 0$

$P_2 \longrightarrow 24 - 0 = 24$

$P_3 \longrightarrow 27 - 0 = 27$

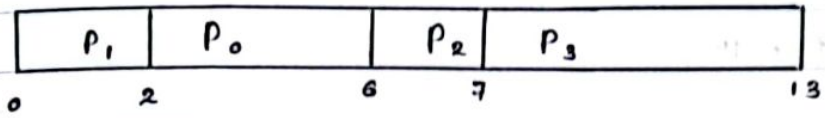
3) Average WT = $(0 + 24 + 27)/3$

$= 17$

No: _____ Date: __/__/____

① 1) Avg WT
2) Avg TT

Process	AT	BT
P_0	1	4
P_1	0	2
P_2	2	1
P_3	3	6



WT = Execution Time - AT

TT = CT - AT

$P_0 \longrightarrow 2 - 1 = 1$
$P_1 \longrightarrow 0 - 0 = 0$
$P_2 \longrightarrow 6 - 2 = 4$
$P_3 \longrightarrow 7 - 3 = 4$

Avg WT = $(1 + 0 + 4 + 4)/4$
= $9/4$

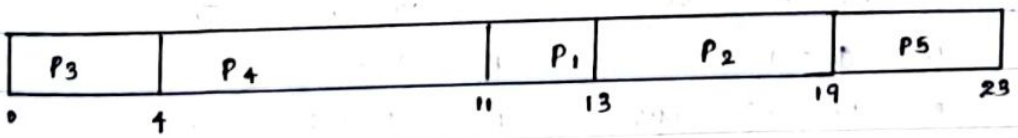
$P_0 \longrightarrow 6 - 1 = 5$
$P_1 \longrightarrow 2 - 0 = 2$
$P_2 \longrightarrow 7 - 2 = 5$
$P_3 \longrightarrow 13 - 3 = 10$

Avg TT = $(5 + 2 + 5 + 10)/4$
= $22/4$

②

Process ID	Arival Time	Burst Time
P1	2	2
P2	5	6
P3	0	4
P4	0	7
P5	7	4

e.g. shortest job 1st



WT = Execution Time - AT

$P_0 = 11 - 2 = 9$
$P_2 = 13 - 5 = 8$
$P_3 = 0 - 0 = 0$
$P_4 = 4 - 0 = 4$

Avg WT = $(9 + 8 + 0 + 4 + 12)/5$
= $33/5$

TT = CT - AT

Avg TT = (11 + 14 + 4 + 11 + 15)/5
= 55/5
= 11

P1 = 13 - 2 = 11
P2 = 19 - 5 = 14
P3 = 4 - 0 = 4
P4 = 11 - 0 = 11
P5 = 23 - 7 = 15

Shortest - Job - First (SJF) Scheduling.

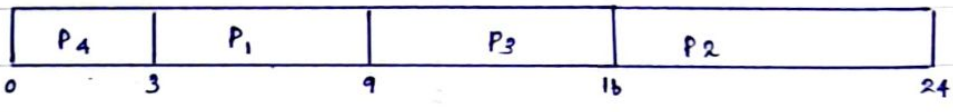
Priemtive version
non-premtive version

- Associate with each process the length of its next CPU burst.
- SJF is optimal. - gives minimum average waiting time for a given set of processes.
  - ✓ Non - premtive → shortest - job - first
  - ✓ Priemtive → shortest - remaining - time - first

Non- Priemtive.

Ex:

Process	AT	Burst Time
P1	0	6
P2	0	8
P3	0	7
P4	0	3



Avg waiting time = (3 + 16 + 9 + 0)/4 = 7

	ET - AT = WT		CT - AT = TT
P1	9 - 0 = 3	P1	9 - 0 = 9
P2	16 - 0 = 16	P2	24 - 0 = 24
P3	9 - 0 = 9	P3	16 - 0 = 16
P4	0 - 0 = 0	P4	3 - 0 = 3

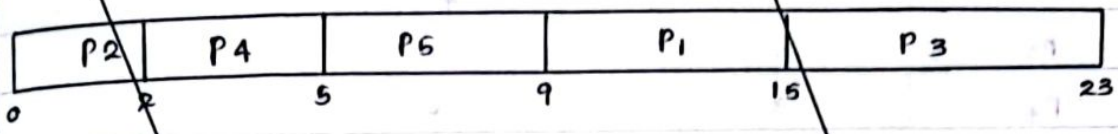
Avg Turnoround time = (9 + 24 + 16 + 3)/4 = 13

No: _____ Date: __/__/____

①

Process Queue	Burst Time	Arrival time
P_1	6	2
P_2	2	5
P_3	8	1
P_4	3	0
P_5	4	4

Time should consider



Waiting time = ET - AT

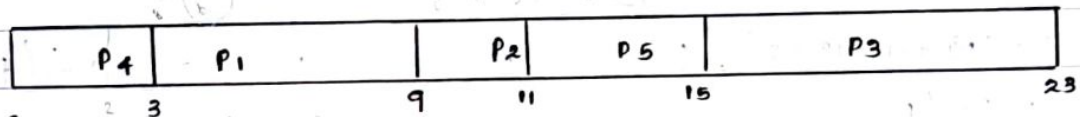
P_1 9 - 2 = 7
P_2 0 - 5 = -5
P_3 15 - 1 = 14
P_4 2 - 0 = 2
P_5 5 - 4 = 1

Avg WT = (7 + (-5) + 14 + 2 + 1)/5
= 19/5

Turnaround Time = CT - AT

P_1 = 15 - 2 = 13
P_2 = 2 - 5 = -3
P_3 = 23 - 1 = 22
P_4 = 5 - 0 = 5
P_5 = 9 - 4 = 5

Avg TT = (13 + (-3) + 22 + 5 + 5)/5
= 32/5



P4 → 3

Waiting time = ET - AT

P_1 3 - 2 = 1
P_2 9 - 5 = 4
P_3 15 - 1 = 14
P_4 0 - 0 = 0
P_5 11 - 4 = 7

Avg WT = (1 + 4 + 14 + 0 + 7)/5
= 26/5

Turnaround Time = CT - AT

P_1 9 - 2 = 7
P_2 11 - 5 = 6
P_3 23 - 1 = 22
P_4 3 - 0 = 3
P_5 15 - 4 = 11

Avg TT = (7 + 6 + 22 + 3 + 9)/5
= 49/5

Preemtive

Shortest - Remaining - Time - First

- Now we add the concept of varying arrival times & preemption to the analysis.

Ex:

Process	Arrival Time	Burst Time
P_1	0	~~8~~ 7
P_2	1	~~4~~
P_3	2	9
P_4	3	~~5~~

P_1	P2	P_4	P_1	P3
0 – 1	1 – 5	5 – 10	10 – 17	17 – 26

(P_1 2nd WT)

* 1st Arrive උන එක execute වෙනවා. ඊට පස්සේ time එක ඉවර වුනාම next process එක 1st එකට වඩා short burst time නම් ඒක execute වෙනවා.

Waiting Time = ET - AT

P_1	$(0-0)+(10-1) = 9$
P_2	$1-1 = 0$
P_3	$17-2 = 15$
P_4	$5-3 = 2$

Turnaround Time = CT - AT

P_1	$17-0 = 17$
P_2	$5-1 = 4$
P_3	$26-2 = 24$
P_4	$10-3 = 7$

Avg WT $= (9+0+15+2)/4$

$= 26/4$

Avg WT $= (17+4+24+7)/4$

$= 52/4$

$= 13$

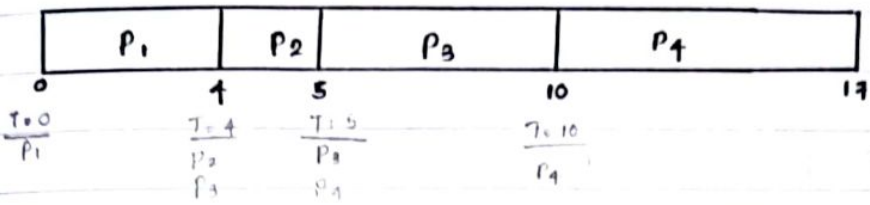
Preemtive → ...

Exercise

Process	AT	BT
P_1	0	4
P_2	2	1
P_3	3	5
P_4	5	7

1) Draw gant chart,
   - SJF
   - SRTF
2) Find AWT & ATT for both.

SJF



WT = ET - AT

$P_1 = 0 - 0 = 0$

$P_2 = 4 - 2 = 2$

$P_3 = 5 - 3 = 2$

$P_4 = 10 - 5 = 5$

9

AWT = 9/4

TT = CT - AT

$P_1 = 4 - 0 = 4$

$P_2 = 5 - 2 = 3$

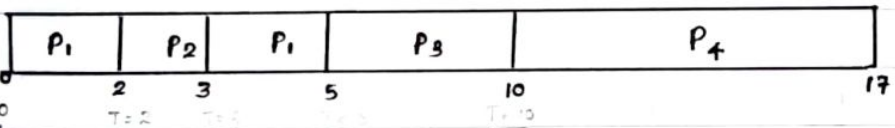
$P_3 = 10 - 3 = 7$

$P_4 = 17 - 5 = 12$

26

ATT = 26/4

SRTF



WT = ET - AT

$P_1 = (0 - 0) + (3 - 2) = 1$

$P_2 = 2 - 2 = 0$

$P_3 = 5 - 3 = 2$

$P_4 = 10 - 5 = 5$

8

AWT = 8/4 = 2

TT = CT - AT

$P_1 = 5 - 0 = 5$

$P_2 = 3 - 2 = 1$

$P_3 = 10 - 3 = 7$

$P_4 = 17 - 5 = 12$

25

ATT = 25/4

Turnaround time එක අඩුයි but its fair for all.

Arrival time නෙමෙන් නෑ.
Process ඔක්කොම එක queue එකේ ඉන්නවා.

Round Robin (RR)

time quantum (q)

- Each process gets a small unit of CPU time (usually 10 - 100 ms)
- After this time has ended the existing process will also added to the end of the queue.

EX: ①

Process	Burst Time
P_1	24
P_2	3
P_3	3

$q = 4$ (quantum time)

P_1 24 — P_2 3 — P_3 3 — P_1 20
P_2 3 — P_3 3 — P_1 20
P_3 3 — P_1 20

WT = ET - AT

$P_1 = (0-0) + (10-4) = 6$
$P_2 = 4-0 = 4$
$P_3 = 7-0 = 7$
17

AWT = 17/3

TT = CT - AT

$P_1 = 30-0 = 30$
$P_2 = 7-0 = 7$
$P_3 = 10-0 = 10$
47

ATT = 47/3

② AT

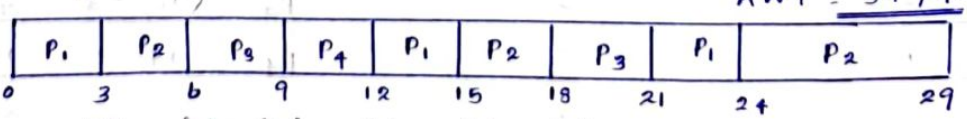
AT	Process	BT	Quantum Time
0	P_1	9	3
3	P_2	11	
1	P_3	6	
5	P_4	3	

P_1 | P_3 | P_2 |
0 3 6 9

WT = ET - AT

$P_1 = (0-0) + (12-3) + (21-15) = 15$
$P_2 = (3-0) + (15-6) + (24-18) = 18$
$P_3 = (6-0) + (18-9) = 15$
$P_4 = (9-0) = 9$
57

AWT = 57/4



P_1 9 — P_2 11 — P_3 6 — P_4 3 — P_1 6 — P_2 8 — P_3 3 — P_1 3 — P_2 5
P_2 11 — P_3 6 — P_4 3 — P_1 6 — P_2 8 — P_3 3 — P_1 3 — P_2 5
P_3 6 — P_4 3 — P_1 6 — P_2 8 — P_3 3
P_4 3 — P_1 6 — P_2 8 — P_3 3

No :

Date :

FCFS : non P
SJF : 2
Round : non
Priority : non

Priority Scheduling.

- A priority number is associated with each process.
- The CPU is allocated to the process with the highest priority.
  - Preemtive
  - Non-Preemtive.

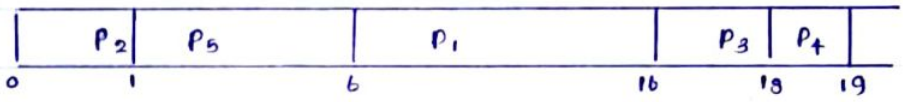
* SJF is priority scheduling where priority is the inverse of predicted next CPU burst time.

Problem → Starvation - low priority processes may never execute. (low priority process එක execute වෙන්න ඉස්සර high priority process එන්න පුළුවන්)

solution → Aging - as time progresses increase the priority of the process. (time limit එකෙන් පස්සෙ low priority execute කරන්න ඉඩ දෙනවා (waiting time).

Ex: ①

Process	BT	Priority.
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



WT = ET - AT

P_1 = 6 - 0 = 6

P_2 = 0 - 0 = 0

P_3 = 16 - 0 = 16

P_4 = 18 - 0 = 18

P_5 = 1 - 0 = 1

41

AWT = 41 / 5

TT = CT - AT

P_1 = 16 - 0 = 16

P_2 = 1 - 0 = 1

P_3 = 18 - 0 = 18

P_4 = 19 - 0 = 19

P_5 = 6 - 0 = 6

60

ATT = 60 / 5 = 12

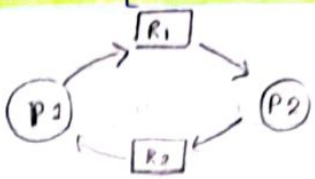
prevention & avoidance

05. Process Management.

A-situation where

Deadlock (A situation where a process is requesting a resource that assigned by another process & the other process also requesting the resource that holds by the previous process)

A deadlock in OS is a situation in which more than 1 process is blocked because it is holding a resource & also requires some resource that is aquired by some other process



characteristics. -> for occur deadlock it

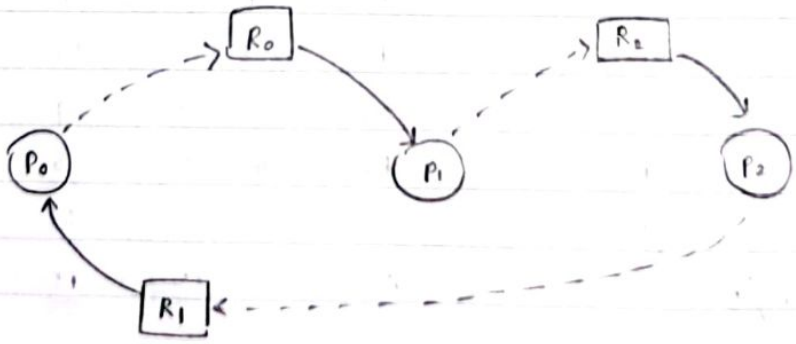
Deadlock can arise if 4 conditions hold simultaniously.

1) **Mutual exclusion** - no sharing ~~only 1~~ resources only 1 process at a time can use a resource.

2) **Hold & wait** - a process holding at least 1 resource is waiting to acquire, additional resources held by other processes.

3) **No preemption** - a resource can be released only voluntarily by the process holding it, after that process has complete its task.

4) **Circular wait** - there exists a set $\{P_0, P_1, P_2 \ldots P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, ..., P_{n-1} is waiting for a resource that held by P_n & P_n is waiting for a resource that is held by P_0.



Exam- (through resource allocation graph we can identify a deadlock)

Resource Allocation graph.

- A set of vertices (V) & set of edges (E).
- V is partitioned into 2 types.
  - P → processes.
  - R → Resources

Circular but not deadlock

- Request edge - directed edge ($P_i \rightarrow R_i$)
- Assignment edge - directed edge ($P_i \leftarrow R_i$)

Example:

1 instance of R1

2 instance of R2

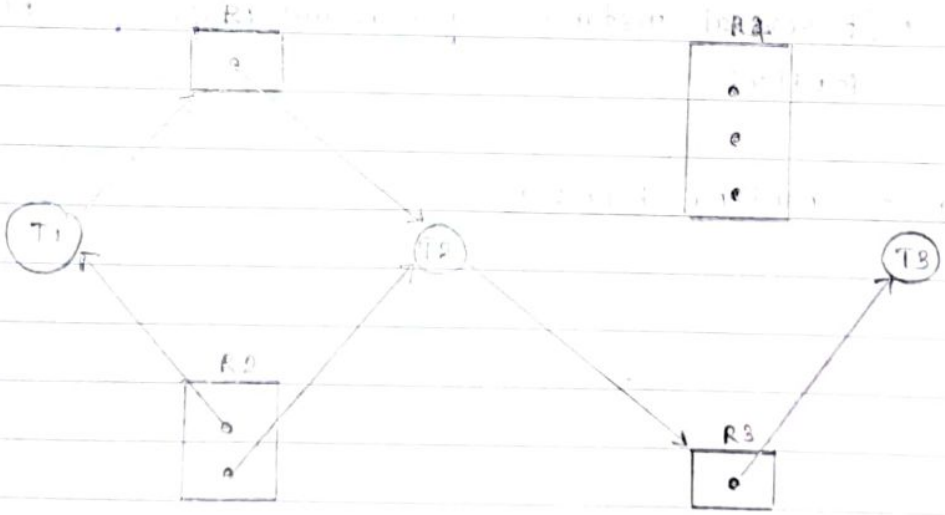
1 instance of R3

3 instance of R4

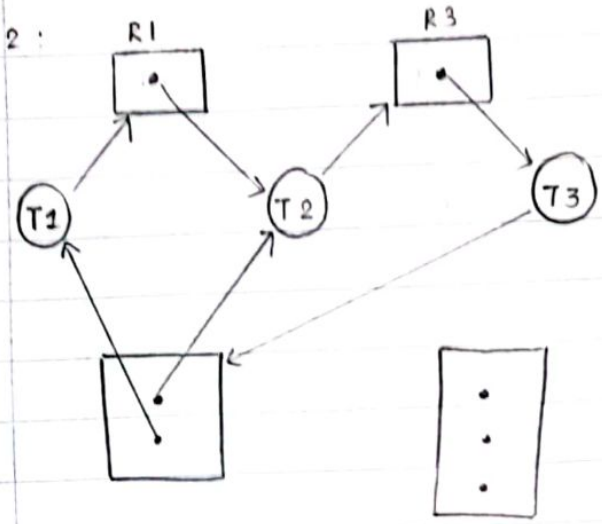
T1 holds 1 instance of R2 & is waiting for an instance of R1.

T2 holds 1 instance of R1, & 1 instance of R2, and is waiting for an instance of R3

T3 is holds 1 instance of R3.

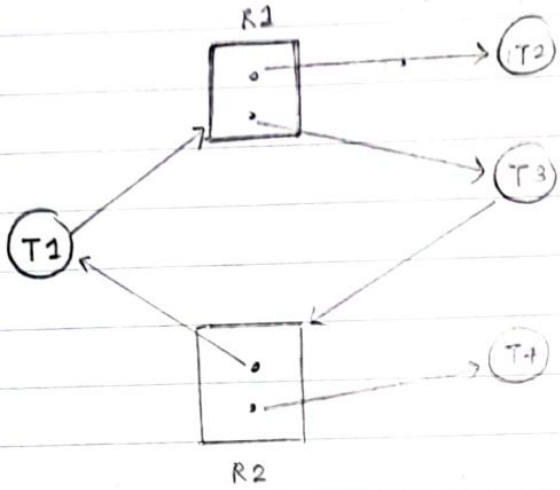


ex 2 :



T1 cannot process without completing T2, T2 cannot process without completing T3. So T3 will locking all the processes.

ex 3 :



T2 and T4 executing without any issue, so that T1 and T3 will also executes smothly. So this graph is with a cycle but no deadlock

* If a graph contains no cycles → No deadlock.
* If a graph contains a cycle →
  - If only 1 instance per resource type, then deadlock.
  - If several instances per resource type, possibility of deadlock.

Methods for handling deadlocks.