Unit - 01

PAGE NO.
DATE. / /

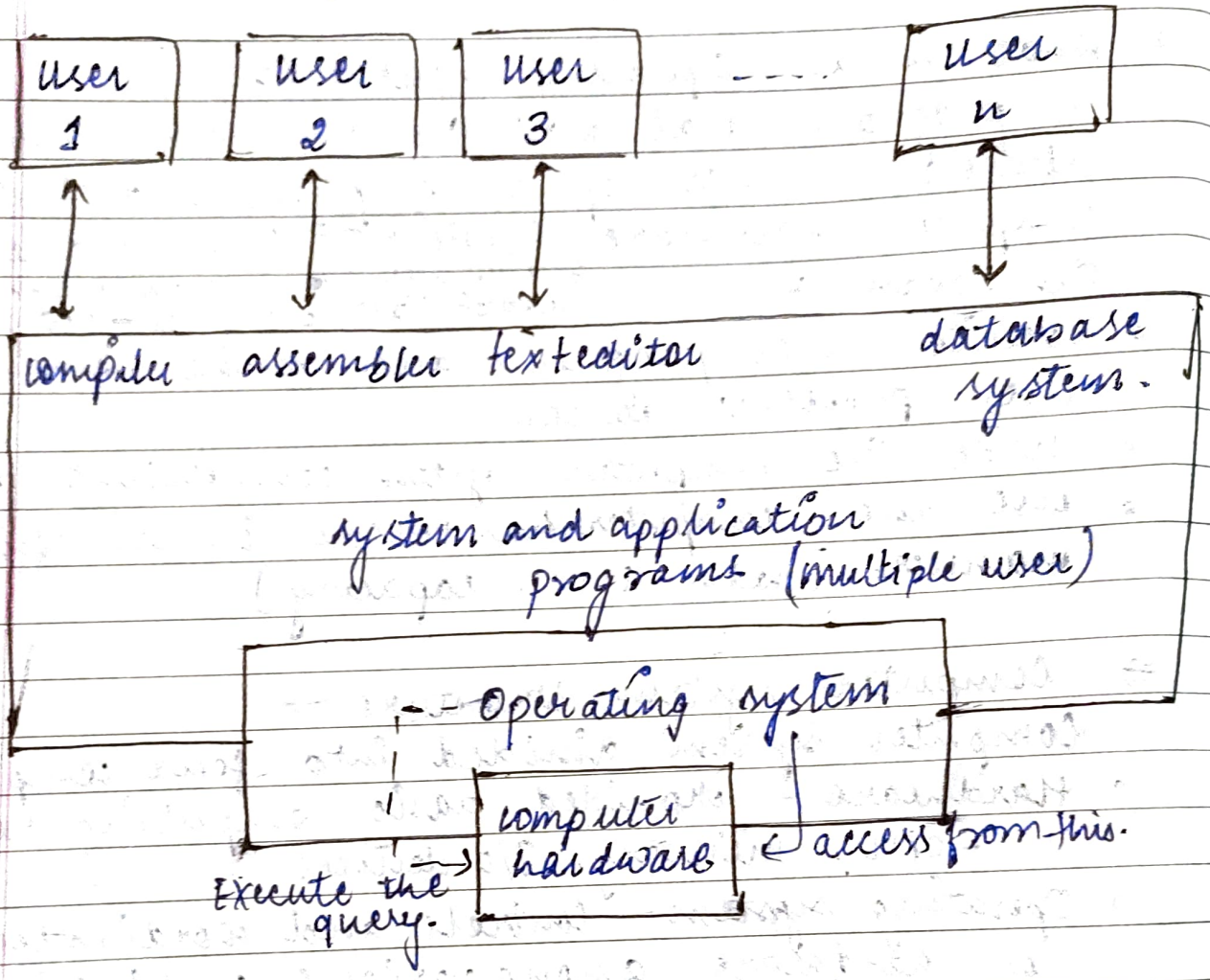
INTRODUCTION

* what is an Operating System (OS) ?
A program that acts as an intermediary between a user of a computer and the computer hardware
OS goals →
- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- use the computer hardware in an efficient manner. [with 100% capacity]

manage and allocate res. ← Resource allocater
exe. of user ← Control program
and op. Vodev. ← Kernel - 1 prog run at all times

* Computer system structure →
Computer system divided into four components.
- Hardware — provided basic computing resources
  - cpu, memory, I/O devices
- Operating system — controls and coordinates use of hardware among various application and users.
- Application programs — define the ways in which the system resources are used to solve the computing problems of the users.
  - word processors, compilers, web browsers, database systems, video games.
- users — People, machine and other computers

Four components of a computer system →



* what OS do ?
  It depends on various factors →
  - users want convience, ease of use and good performance. Don't care about resource utilization.
  - But shared computer such as mainframe or minicomputer must keep all users happy.
  - users of dedicate systems such as workstation have dedicated resources but frequently use shared resources from servers.

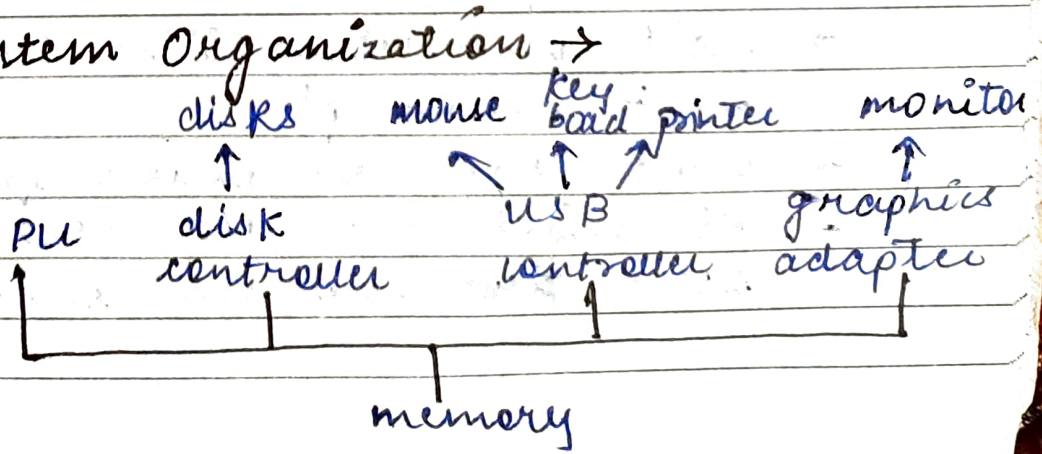
PAGE NO.
DATE. / / /

- Handheld computers are resource poor, optimised for usability and battery life.
- Some computers have little or no user interface, such as embedded computers in devices and automobiles.

* About Operating system →
  - OS is a resource allocator
    - Manages all resources.
    - Decides between conflicting requests for efficient and fair resource use.
  - OS is a control program
    - Controls execution of programs to prevent errors and improper use of the computer.

* Computer startup →
  - bootstrap program is loaded at power-up or reboot.
    - Typically stored in ROM and EPROM, generally known as firmware.
    - initializes all aspects of system.
    - loads OS kernel and starts execution.

* Computer system Organization →



§ • Computer system operation
- One or more CPUs, device controllers connect through common bus providing access to shared memory.
- Concurrent execution of CPUs and devices competing for memory cycles. §.

* Computer - system operations :→
- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer.
- CPU moves data from / to main memory to / from local buffers.
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.

* I/O Structure →
- Control returns to user program only upon I/O completion.
- Wait instruction idles the CPU until the next interrupt
- Wait loop (contention for memory access)
- At most one I/O request is outstanding at a time, no simultaneous I/O processing

- control returns to user program without waiting for I/O completion.
  - System call → request to the OS to allow user to wait for I/O completion.
  - Device-status Table → contains entry for each I/O device indicating its type, address and state.
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

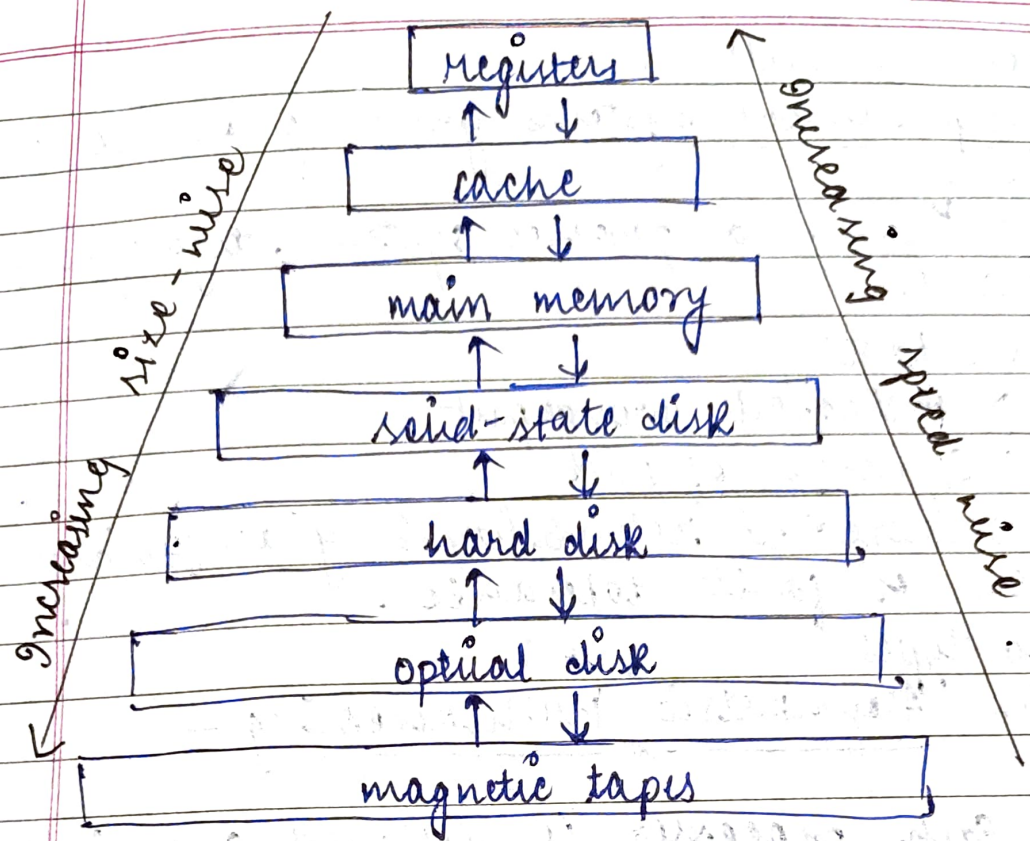
* Storage definitions & Notation Reviews. → The basic unit of computer storage is the bit. A bit can contain one of two values 0 and 1. All other storage in a computer is based on collections of bits. A byte is 8 bits and on most computers it is the smallest convenient chunk of storage.

  KB = 1024 B
  MB = $(1024)^2$ B
  GB = $(1024)^3$ B
  TB = $(1024)^4$ B
  PB = $(1024)^5$ B

* Storage Structure →
  - Main memory - only large storage media that the CPU can access directly.
  - Random access.
  - Typically volatile

- Secondary storage — extention of main memory that provides large non-volatile storage capacity.
- Hard disks — rigid metal or glass platters covered with magnetic recording material.
  - Disk surface is logically divided into tracks, which are subdivided into sectors.
  - The disk controller determines the logical interaction between the device and the computer.
- Solid-state disks — faster than hard disks, nonvolatile.
  - various technologies.
  - Becoming more popular

* Storage hierarchy →
  - storage systems organised in hierarchy.
    - speed (v. high)
    - cost (v. low)
    - volatility (v. high)
  - caching — copying information into faster storage system; main memory can be viewed as a cache for secondary storage.
  - Device Driver — for each device controller to manage I/O.
    - Provides uniform interface between controller and Kernel.



- storage device hierarchy.

* Direct Memory Access Structure →
  → Used for high-speed I/O devices able to transmit information at close to memory speeds.
  → Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
  → Only one interrupt is generated per block rather than the one interrupt per byte.

* Computer System Architecture →
  - Most system use a single general-purpose processor.
  - Most systems have special-purpose proce-

-ssors as well.

- Multiprocessors systems growing in use and importance.
  - Also known as parallel systems, tightly-coupled systems.
  - Advantages
    - > Increased throughput
    - > Economy of scale
    - > Increased reliability: graceful degradation or fault tolerance.
- Two types
  1. Asymmetric Multiprocessing –

     Each processor is assigned a specie task.
  2. Symmetric Multiprocessing –

     Each processor performs all tasks

* OS structure →
  - Multiprogramming (Batch system) – needed for efficiency
    - Single user cannot keep cpu and I/o devices busy at all times.
    - Multiprogramming organizes jobs (code and data) so cpu always has one to execute.
    - A subset of total jobs in system is kept in memory

- One job selected and run via job scheduling.
- When it has to wait (for I/O fo ex.), OS switches to another job.

- Timesharing (multitasking) → is logical extention in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing.
- Response time should be < 1 sec.
- Each user has atleast one program executing in memory → process.
- If several jobs ready to run at the same time → CPU scheduling
- If processes don't fit in memory, swapping moves them in and out to run.
- Virtual memory allows execution of processes not completely in memory.

* OS Operations →
- Interrupt driven (hardware & software)
- Hardware interrupt by one of the devices.
- Software interrupt (exception or trap)
  > software error (eg division by 0)
  > Request for operating system service
  > Other process problems include infinite loop, processes modifying each other or the OS.

PAGE NO.
DATE. / /

- Dual mode operation allows OS to protect itself and other system components.
  - > User mode and Kernel mode
  - > Mode bit provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code.
    - Some instructions designated as privileged, only executable in Kernel mode.
    - System call changes mode to Kernel, return from call resets it to user.
- Increasingly CPUs support multi-mode operations
  - > ie virtual machine manager (VMM) mode for guests VMs.

* Process Management →
- A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.
- Process needs resources to accomplish its task.
  - > CPU, memory, I/O files.
  - > Initialization data.
- Process termination requires reclaim of any reusable resources

- Single threaded process has one program counter specifying location of next instruction to execute.
  > Process executes instructions sequentially one at a time, until completion.
- Multi-threaded process has one program counter per thread.
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  > Concurrency by multiplexing the CPU among the processes/threads.

* Process Management Activities →
- Creating and deleting both user and system process.
- Suspending and resuming processes.
- Providing mechanisms for process synchronization.
- Providing mechanisms for process communication.
- Providing mechanisms for deadlock handling

* Memory Management →
- To execute a program all of the instructions must be in memory.
- All of the data that is needed by the prog. must be in memory.
- MM determines what is in memory and when > optimizing CPU utilization and comp. response to users.

- Memory manag. activities.
  > Keeping track of which parts of memory are ~~to~~ currently being used & by whom.
  > Deciding which processes and data to move into and out of memory.
  > Allocating & deallocating memory space as needed.

{ L.A. - 2ⁿ ?.

to change logical Ad. into Phy. add. mm plays a vital role.

* Storage Management →
  - OS provides uniform, logical view of information storage.
    > Abstract phy. prop. to logical storage unit - file
    > Each medium is controlled by device
      - Varying prop. include access speed, capacity, data transfer rate; access method (sequential / random).

  - file system management.
    > files usually organised into directories.
    > Access control on most systems to determine who can access what.
    > OS activities include
      - creating & deleting files / directories
      - Primitives to manipulate files / directories.
      - Mapping file onto secondary storage.
      - Backup files onto stable storage media.

PAGE NO.
DATE. / /

* Mass - Storage Management →
- Disks used to store data that does not fit in main ~~main~~ memory or data that must be kept for a "long" period of time.
- Proper management is of central imp.
- Entire speed of computer operation hinges on disk subsystem and its algo.
- OS activities.
  - > free space manage.
  - > storage allocation
  - > disk scheduling.
- Some storage need not be fast.
  - > Tertiary storage (optical stor, mag. tape)
  - > Still must be managed - by OS app.
  - > Varies b/w WORM (Write once, read many times) and RW (Read write).

* Performance of various levels of storage →

Level	1	2	3	4	5
Name	registers	cache	main memory	SSD	mag. disk
Typical size	< 1 KB	< 16 MB	< 64 GB.	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk.
Access time	0.25 - 0.5	0.5 - 25	80 - 250	25000 - 50,000	5,000,000
Bandwidth	20,000 - 100,000	5,000 - 10,000	1000 - 5,000	500	20 - 150
Managed by	compiler	hardware	OS	OS	OS
Backed by	cache	main memory	disk	disk	disk or tape.

PAGE NO.
DATE. / /

* I/O subsystem →
- One purpose of OS is to hide peculiarities of hardware devices from the user.
- I/O subsystem responsible for
  > memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
- General device-driver interface.
- Drivers for specific hardware devices.

* Protection & Security →
- Protection → any mechanism for controlling access of processes or users to resource defined by the OS.

- Security → defense of the system against internal and external attacks.
  > Huge range, including denial-of-service worms, viruses, identity theft, theft of service.
- Systems generally first distinguish among users, to determine who can do what
  > User identities (user IDs, security IDs) including name and associated no., one per user.
  > User ID then associated with all files processes of that user to determine

access control.

> Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file

> privilege escalation allows user to change to effective ID with more rights.

⇒ Process Management :

- Program & Process

Program →

- A program is a piece of code.
- A program is a passive entity as it resides in the secondary memory, such as the contents of a file stored on disk.
- When we execute a program that was just compiled, the OS will generate a process to execute the program. Execution of the program starts via GUI mouse clicks, command line entry of its name, etc. One program can have several proces.

Process →

- The term process (Job) refers to the program code that has been loaded into a computer's memory so that it can be executed by the central processing unit (CPU).

PAGE NO.
DATE. / /

- A process can be described as an instance of a program running on a computer or as an entity that can be assigned to and executed on a processor.
- A program becomes a process when loaded into memory and this is an active entity.

Eg.

```
main()
{
int a, x=1;
for(i=0; i<100; i++)
  x = x*i;
}
```

Whole program is a single line

Whole process is a 100 line statement execution

⇒ Differentiation b/w Process & Program

Parameter	Process	Program
Definition	executing part of the prog.	Group of ordered operations to achieve a programming goal.
Nature	An instance of the program being executing.	passive, so it's unlikely to do to anything until it gets executed.
Resource Management	Resource Requirement is quite high.	Only needs memory for storage.

Overheads	Co
Lifespan	Has li ge co
Creation	Ne cr pr
Required Process	Pr i cl
Entity type	
Contain	• M • M

⇒ Type

- Uni
  - Uni
  - Mult
  - Mult
- Multi

Overheads	Considerable overhead.	No significant overhead cost.
Lifespan	Has a shorter and very limited lifespan as it gets terminated after the completion of the task.	Has a longer lifespan as it is stored in the memory until it is not manually deleted.
Creation	New processes require duplication of the parent process	No such duplication is needed.
Required Process	Process holds resources like CPU, memory addr., disk, I/O etc.	stored on disk in some file and doesn't require any other resources.
Entity type	Dynamic or active type	⇒ passive or static entity.
Contain	many resources like a memory address, disk, printer, etc.	needs memory space on disk to store all instructions

⇒ Types of OS

- Uni processing
  - Uni prog.
  - Multiprogramming
  - Multitasking / Time sharing
- Multiprocessing

PAGE NO.
DATE. / /

- Uniprocessing (single processor)

Eg.

Suppose → 3 processes

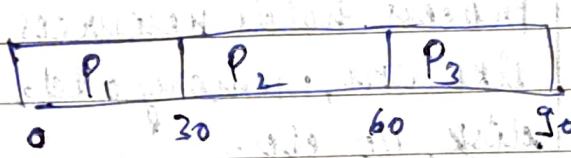
Programmer/user exe called process

P_1	P_2	P_3
30 unit time	30 UT	30 UT

	fetch (read)	decode (ins.)	Exe (execut.)
P_1	10 t	10 t	10 t
P_2	10 t	10 t	10 t
P_3	10 t	10 t	10 t

F	D	E
10	10	10

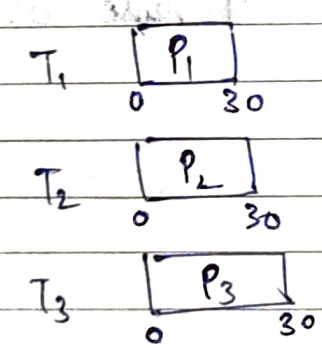
1 Processor :



Total time for Exe all 3 processes = 90.

- Multiprocessing (2 or more processors)

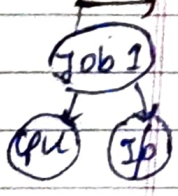
Suppose 3 processor

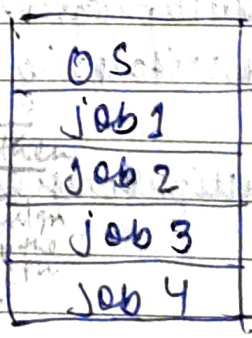


Parallel mode

Total time for exe all 3P. = 30

→ Multiprogramming:





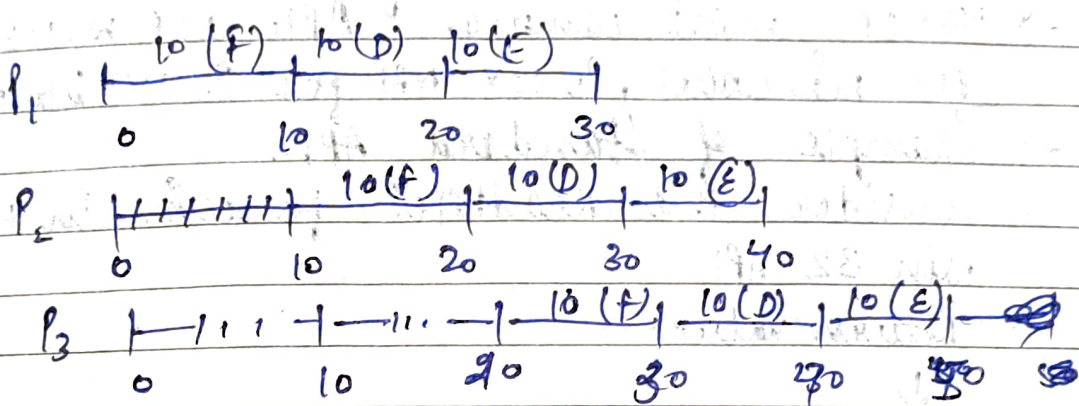
PAGE NO.
DATE. / /

→ ~~multiprocessing~~

Suppose 3 Processes

P_1	P_2	P_3
30	30	30

10 → fetch
10 → Decode
10 → Execute.



- Total time to complete all process. = 50

→ Multitasking. One of the process will be scheduled to CPU.
CPU (Job 1) (It is assigned will be time slot)

- There is not starvation occur.

⇒ Real time OS - Systems which are (strict deadly time bound).
→ Hard RTOS
→ Soft RTOS (minute delay is accepted).

- System Calls:
} Request made by user program (whatever program we have written) to the OS in order to get any kind of service. {

API used as system call

DATE. / /

→ Programming interface to the services provided by the OS.
→ Typically written in a high-level language (C or C++)
→ Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use.
→ Three most common APIs are WIN32 API for Windows, POSIX API for POSIX based sy
→ Example of Standard API.
- Consider the ReadFile() function in the win 32 API
- Return value

API or system cal
OS uses these code for lib

```
BOOL   ReadFile (  HANDLE       file,
                   LPVOID       buffer,
                   DWORD        byte to Read
                   LPDWORD      byte Read
                   LPOVERLAPPED ovl );
```

function name

Parameters

- A description of the Parameters passed to ReadFile()

* Handle file - the file to be Read
* LPVOID buffer - a buffer where the data will be read into and written form.
* DWORD bytes to read - the no. of bytes read during the last read.
* LPDWORD bytesRead - the no. of bytes read during the last read.
* LPOVERLAPPED ovl - Indicates if overlapped I/O is being used.

— System call Implementation

→ Typically, a no associated with each system call.

• System call interface maintains a table indexed according to these no.

→ The system call interface invokes intended system call in OS kernel and return status of the system call and any return values.

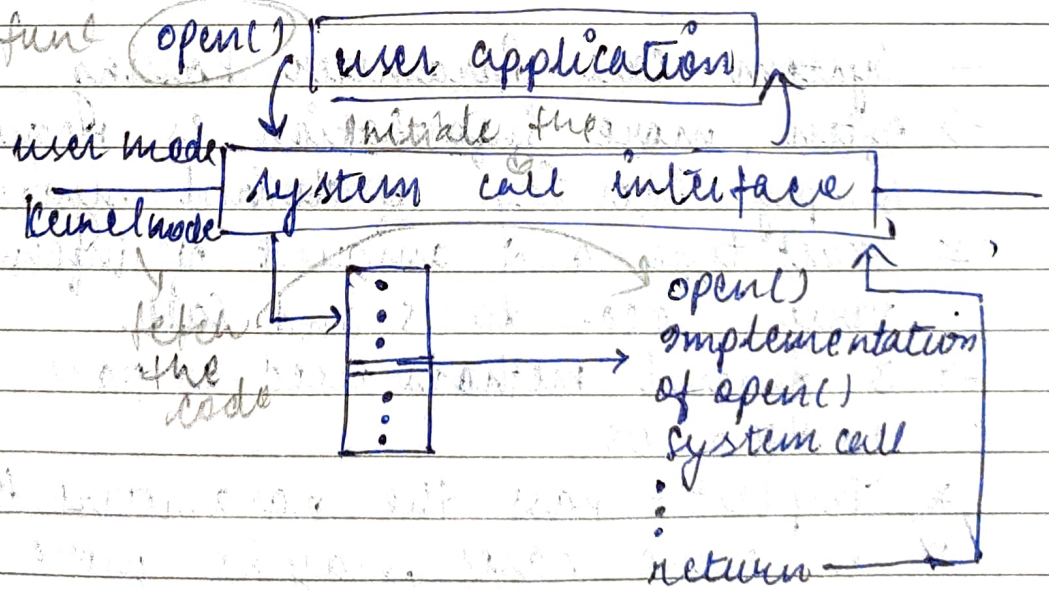
→ The caller need know nothing about how the system call is implemented.

* Just needs to obey API and understand what OS will do as a result call.

* Most details of OS interface hidden from programmer by API.

• Managed by run-time support library (set of functions built into libraries included with compiler).

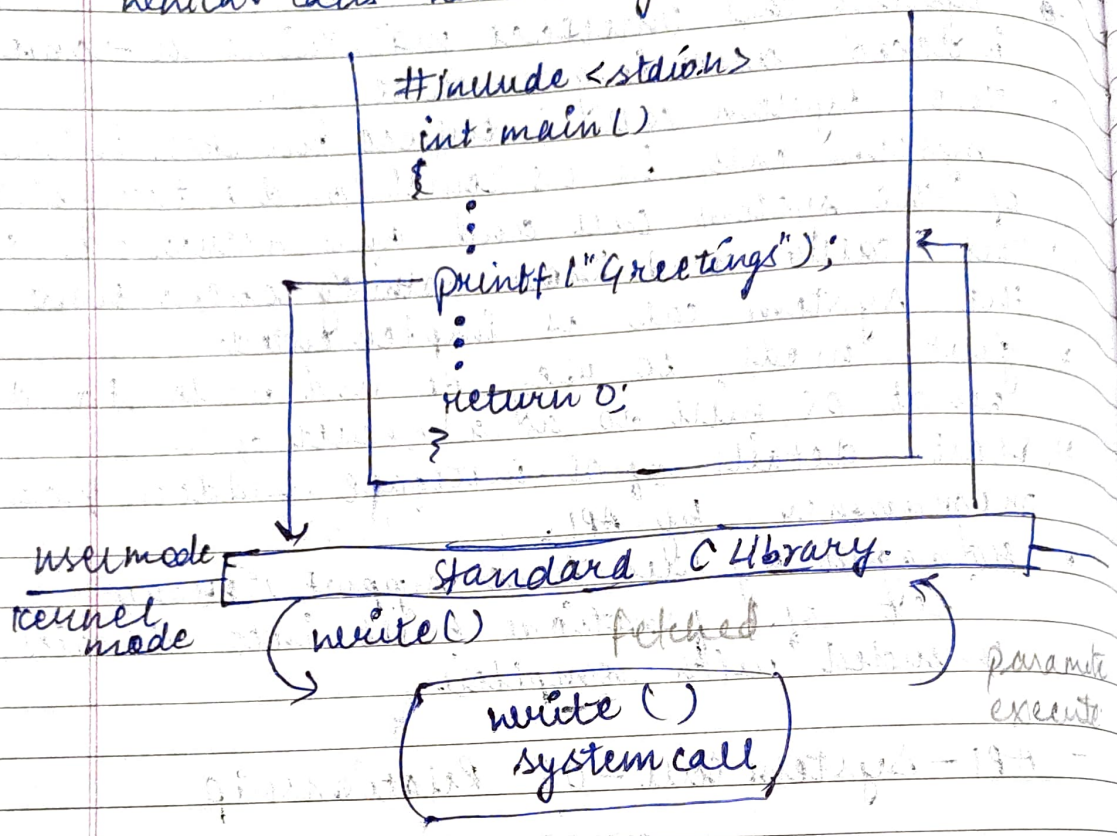
— API – System Call – OS Relationship



PAGE NO.
DATE. / /

– Standard C library Example

→ C program invoking printf() library call, which calls write() system call.



– System call Parameter Passing

→ Often, more information is required than simply identity of desired system call.

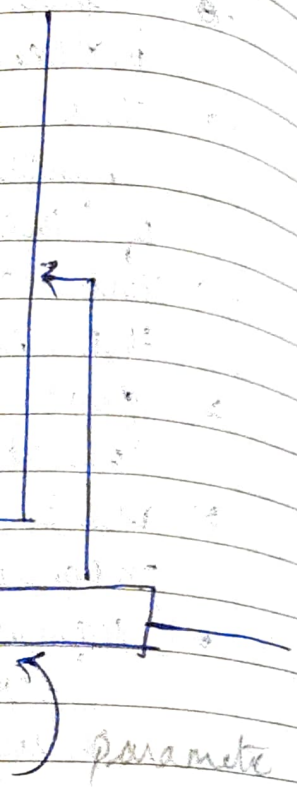
* Exact type and amount of information vary acc. to OS and call.

→ 3 general methods used to pass parameters to the OS

* Simplest – pass the parameters than registers

* In some cares, may be more parameters than registers

ibrary call.



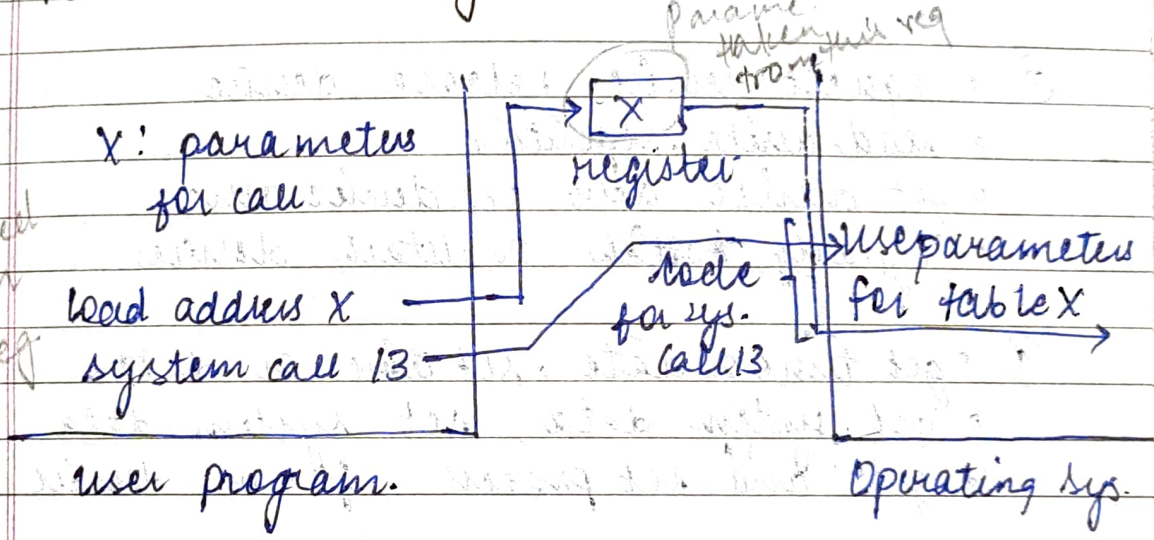
quired than
tem call.
rmation

parameters

than registers
e parameters

→ Parameters stored in a block, or table in memory, and address of block passed as a parameter in a register.
* This approach taken by Linux and Solaris.
→ Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system.
→ Block and stack methods do not limit the number or length of parameters being passed.

— Parameter Passing via Table:



— Types of system calls
① → Process control
② → file management
③ → device management
④ → information maintenance
⑤ → communications

PAGE NO.
DATE. / /

Unit - 2

(1) ◦ End, abort
◦ load, execute
◦ create process, terminate process.
◦ get process attributes, set process attr.
◦ ~~get~~ wait for time
◦ wait event, signal event
◦ allocate and free memory

(2) ◦ create file, delete file
◦ open, close file
◦ read, write, reposition
◦ get and set file attr.

(3) ◦ request device, release device
◦ read, write, reposition
◦ get device attr, set device attr.
◦ logically attach or detach devices

(4) ◦ get time or date, set time or date.
◦ get system data, set system data
◦ get and set process, file or device attr

(5) ◦ create, delete communication connection
◦ send, receive messages.
◦ transfer status information
◦ attach and detach remote devices.

⇒ Representation of Process

- Process in

code is

* Process c
◦ Process s
◦ Program
◦ CPU regi
◦ CPU sched
◦ Memory -
◦ Accounting
◦ I/O statu

Unit - 2nd

PAGE NO.
DATE. / /

~~Process management.~~
~~Representation of process in memory.~~

⇒ Representation of processes in memory.

Process

A B C . C

↓

Compiler

↓

a.out

- Process in memory :

Stack

Hold temp. data (Recursive, func. etc)

Heap

dynamic allocation. (calloc, malloc)

Data section

all the variables in which program change value.

Code section.

(code written). a.out will be pasted

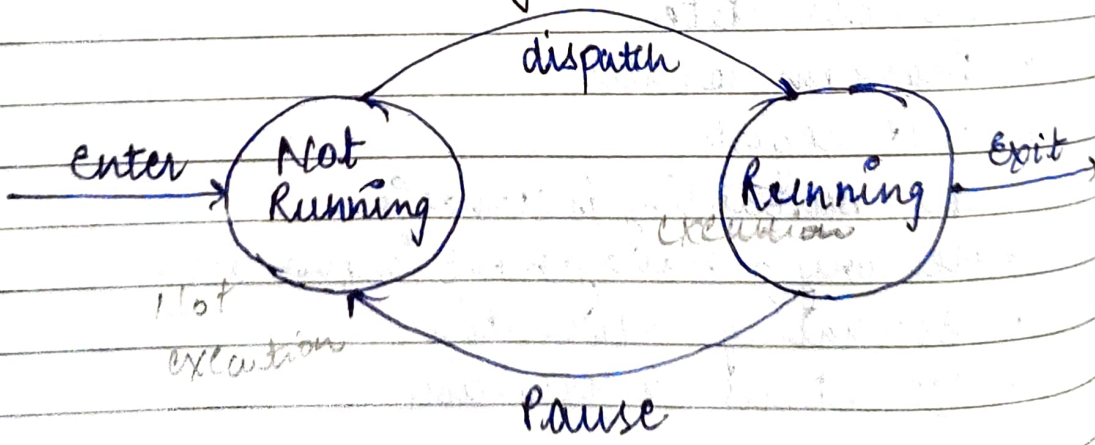
* Process control Block: (complete info about process)
  - Process state (data str.)
  - Program counter (system want to execute next ...)
  - CPU registers (used by the ...)
  - CPU scheduling information
  - Memory - management information (location of files)
  - Accounting information (time of ...)
  - I/O status information.

⇒ Process state transition diagram.
- CPU scheduling
- Multiprocessing
- Process synchronization
- Deadlock

* Process State
- Each process moves through a number of phases from entry into the system until it completes or exits.
- From a process point of view it is either running or not
- The reasons for not running has to be maintained by the the systems for appropriate progress of the process to be gained.
- In a multiprogramming environment each process state is maintained in a process table.

→ 2 state Model.

(a) State transition diagram

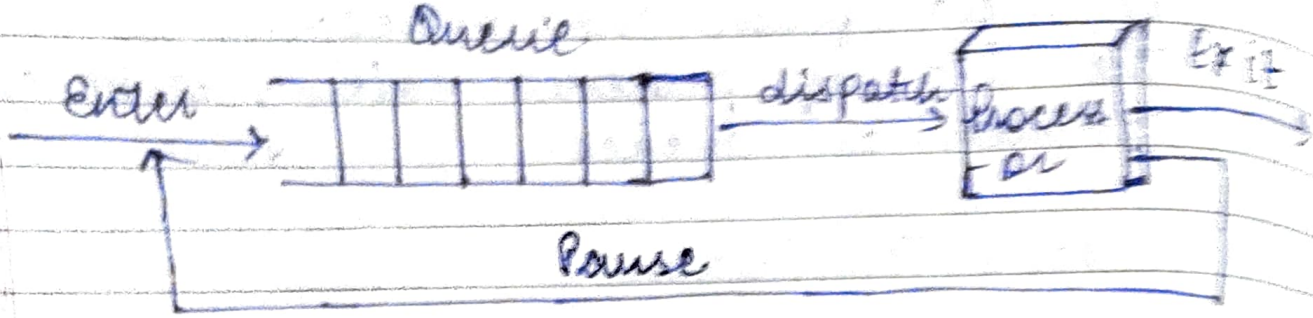


- The 2-state process easily represents the running / not running states from the point of view from the process.
- The list of processes waiting to be executed (to run) are queued.

Process Termination

Normal completion.	Programs ends normally.
Time limit exceeded	a prespecified amount of CPU time has been exceeded.
Memory unavailable	a request for memory allocation fails, either exceeding a prespecified amount or the available system memory is exceeded.
Bounds violation	an address is referred that is not permited.
Protection error.	a resource was accessed but denied permission
Time overrun.	an event has not occurred as expected.
I/O failure	requested ~~[illegible]~~ service from a device has failed usually a hardware error

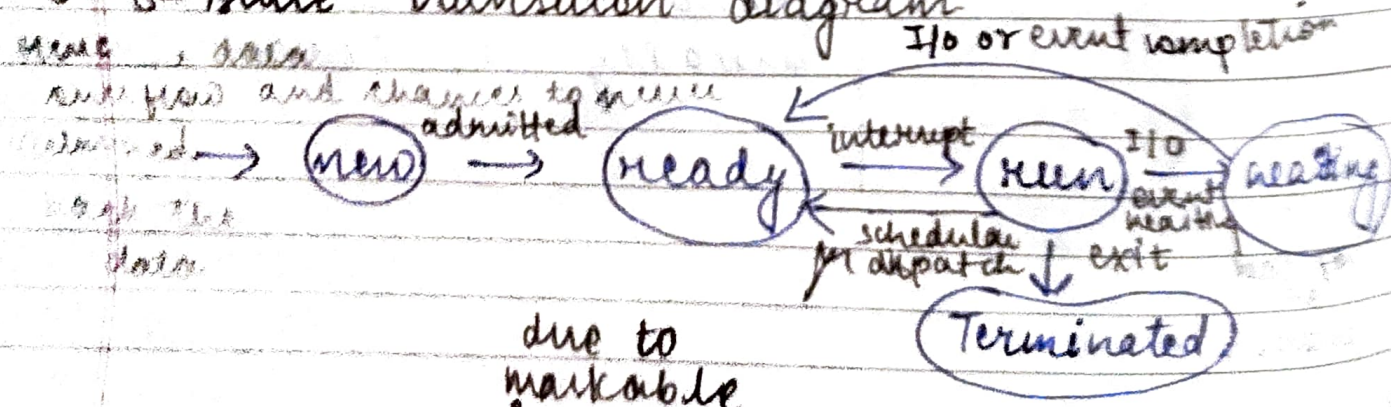
→ Queueing diagram of 2 state model



* Why require more state ?
- This model do not represent the reason for waiting (not running state).

→ 5-State Process Model.
As a process executes, it changes 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 process.
- terminated :- The process has finished execution.

o 5-State transition diagram



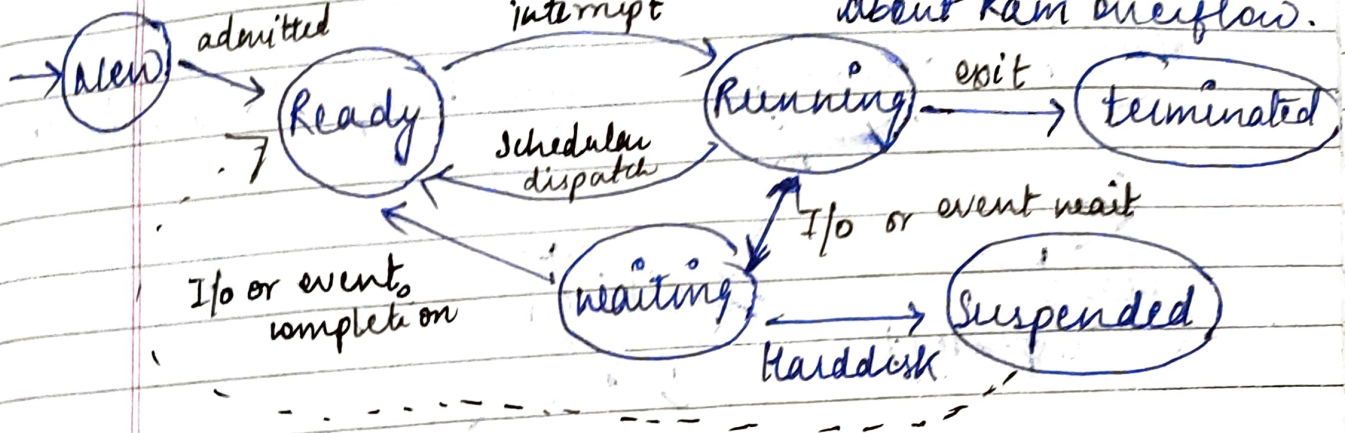
due to markable interrupt we have directly Run to Ready arrow

⇒ Yes, we process running to ready directly. CPU free to move/process the int.

No, we can't move wait to Run directly. We will if we move CPU collapse and it destroy process int.

→ 6 State transition diagram

- Here we discuss about Ram overflow.



- Process Model.

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 process
Terminated The process has finished execution
Suspended The process is suspended from waiting state due to memory.

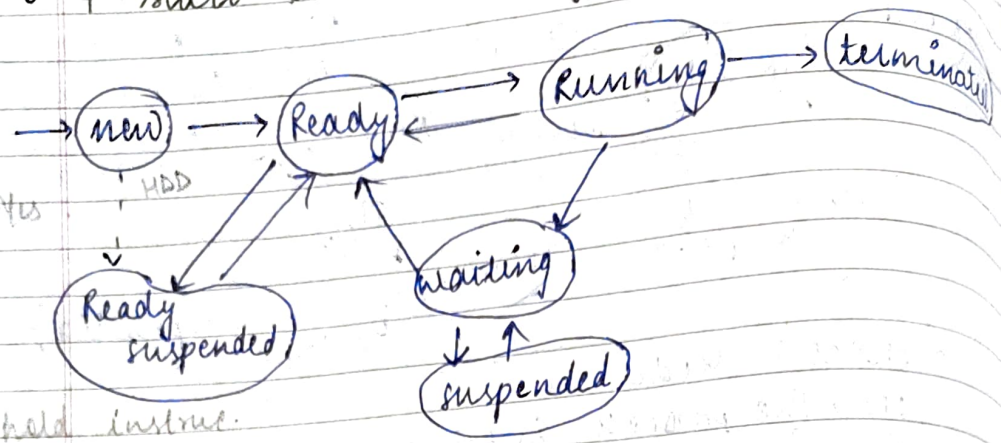
→ 7- State Process Model

New ''
Running ''
Waiting ''
ready ''
terminated ''
Suspended ''
Ready Suspended The process is suspended from ready state due to memory.

DATE. / /

* Degree of m

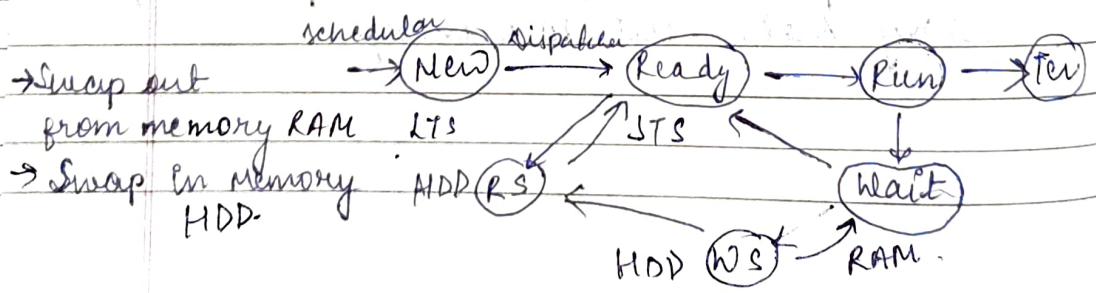
- 7-state transition diagram



hold instruc. which ready for execution (Lack of memory)

→ Schedular & dispatcher.
(schedule task / process). (send task from one place to other).

There are 3 kind of schedular
(a) Long term schedular select the process (sends the process to Ready state) New state
(b) Short term schedular
(sends the process to Ready state
(c) Mid-term schedular RAM → HDD } Mid Term
HDD → RAM } Schedular



Mid-term sch
selection of p
stronger swap

Q1. which sche
invoked ?
Short term
RA

Q2. which sche
of multipr
Long term
x process is
state.

→ CPU sche
(1) Scheduling
(2) Algorithms

(1) (a) Process
(b) Through
(c) Turnarou
(d) waiting

* degree of multi-prog - no of process in RAM.

PAGE NO.
DATE. / /

Mid-term scheduler is responsible for selection of process which we want to sort through swapping.

terminated

Q1. which scheduler is most frequently invoked ?

Short term scheduler. (CPU scheduler).

RAM → CPU

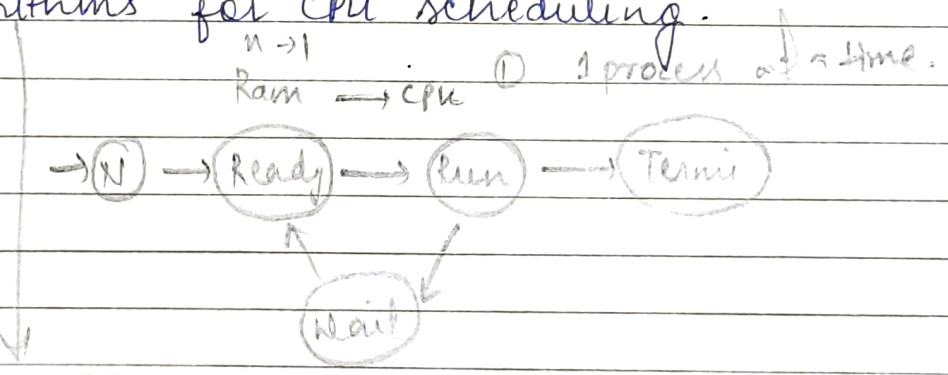
Q2. which scheduler controls the degree of multiprogramming ?

Long term scheduler. coz it send the X processes to Ready state from new state.

→ CPU scheduling (short term scheduler)

① Scheduling & Performance criteria.

② Algorithms for CPU scheduling.



Mid Term scheduler

→ Ter

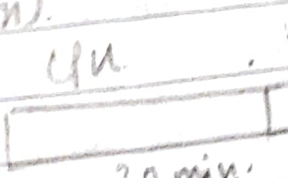
① (a) Processor utilization.

(b) Throughput

(c) Turnaround time (Completing)

(d) waiting time.

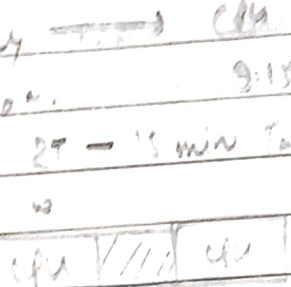
Turnaround time (process takes to complete its execution).



TAT = 30 + 10 + 10 = 50 min.

TAT = BT + WT
↓ ↘ (waited outside I/O process)
Burst time
(execution in CPU)

(2) Response time
subset of waiting time.
→ 1st waiting time



RT = ST of CPU = (15-5) + (30-20) = 20.
WT = (5-0) + (20-15) = 10.
TAT = 30 min → 5)
Response time

PAGE NO.
DATE. / /

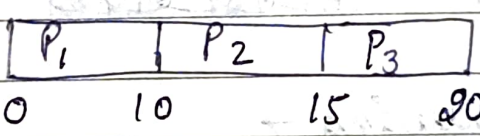
② Algorithms for CPU scheduling.

(a) FCFS
(b) SJF
(c) Priority scheduling
(d) Robin Round.

(a) FCFS → when we don't provide any rules & regulations. then this very effective (First come first serve).

	BT
P_1	10
P_2	5
P_3	5

The process will arrived first it will be executed first.



In this way we represent the process algo of any processes.

AT		BT	WT
0	P_1	10	0
0	P_2	5	10
0	P_3	5	15

WT = Starting time − Arriving time
= 10

TWT = 0 + 10 + 15 = 25

TAT = BT + WT
= 10 + 0 = 10
5 + 10 = 15
5 + 15 = 20

TAT = WT + BT
TAT = End − Arrived.

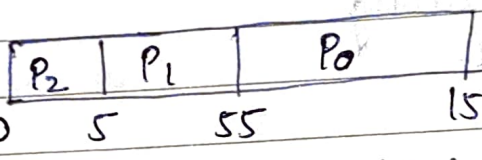
TTAT = 45

(process time) Burst Time = BT

(b) Shortest Job first (SJF) → It processed / execute the Processes which have smaller process time first. Here, it will be used for longer processed time processor execution criteria to processed.

	BT	AT	FCFS WT	SJF WT
P_0	100	0	0	55
P_1	50	0	100	5
P_2	5	0	150	0
		TOT = 250		AVG 60

SJF reduced almost $\frac{1}{4}$th time (processing) or more than to FCFS.
So, it is the advantage of SJF.



Total time of chart = last time - first time.
= Total Burst time (BT).
[if we idle portion in mid].

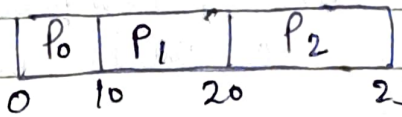
It has two variations →
→ Presented (AT time same).
→ Non Presented. (AT not same)

(c) Priority scheduling →
provide the external priority to which process execute first like - user process is more the system whether the process is for interaction purpose or background purpose. there are some priority criterias which helps to schedule the algo processes.

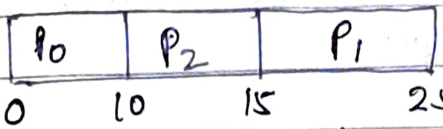
Eg.

	BT	AT	P.
P_0	10	0	3
P_1	10	1	1
P_2	5	02	2

- FCFS

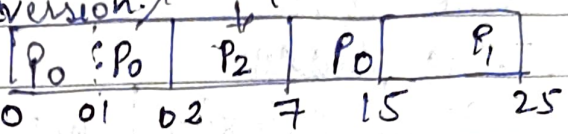


- SJF N.P. version.



$P_0 = 10$
$P_0 = 8$.
$P_1 = 10$
$P_2 = 5$

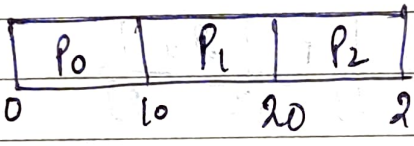
Preemptive version → out in



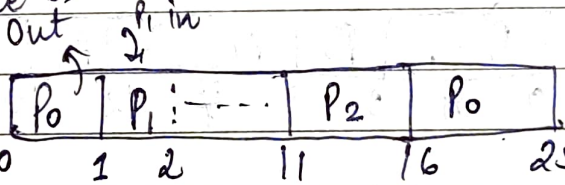
[By default → If nothing is given as external priority then, smaller no. means higher priority.

Higher no → Higher Priority.]

- Priority (Non - Preemptive).



Preemptive →



Advantage → user or system can provide the external priority and on the basis of those priority processes will be executed. So, system will be smooth in terms of execution. Since, the process of system is already decided.

Disadvantage → They don't consider the interactivity. They consider there specific criteria.

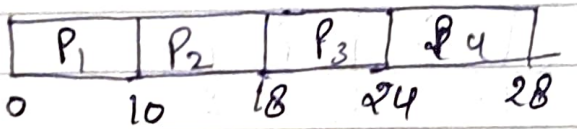
(d) Round Robin → Pure preemptive algorithm on the process on the basis of time process will be schedule.
fix quant time schedule for the execution of the process specific time to the CPU.

CPU → Time Quant
Time slice.

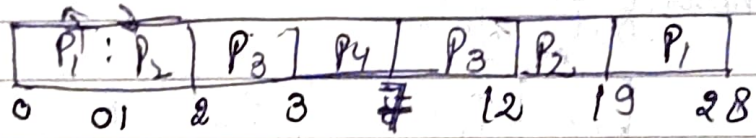
	BT	AT	Priority	WT (FCFS)	WT(SJF)	WTP.	WTRR
P_1	10	0	1	0			22
P_2	8	1	4	10			22
P_3	6	2	2	7			14
P_4	4	3	3	4			8

Time slice = 2

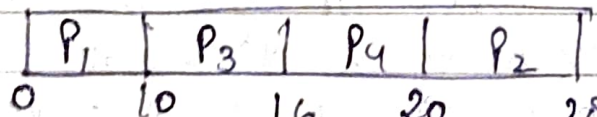
- FCFS



- SJF

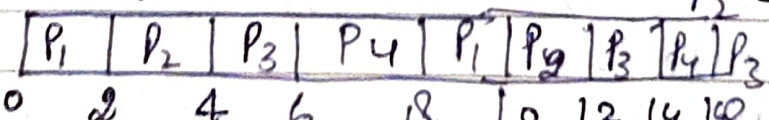


- Priority



- Round Robin (AT is same). $P_1 = 10-2=8$

6 4 2 TQ = 2 $P_2 = 8-2=6$
4 2 0 $P_3 = 6-2=4$
2 0 $P_4 = 4-2=2$
0



P_1	P_2	P_3	P_1	P_2	P_1

16 18 20 22 24 26 28

Waiting time :

$P_1 = 6+6+6+4$
$= 22$

$P_2 = 6+8+8$
$= 22$

$P_3 = 8+6$
$= 14$

$P_4 = 8$

F)	WTP.	WTRR
		22
		22
		14
		8

$0-2=8$
$8-2=6$
$6-2=4$
$4-2=2$

⇒ Multiprocessing
- ↳ Asymmetric multiprocessing
- ↳ Symmetric multiprocessing.

When multiple processors are involved in a task. called multiprocessing

- Load Balancing
- Migration
  - ↳ Push
  - ↳ Pull
- Pro. Affinity
  - ↳ $P.A \propto \frac{1}{L.B}$

⇒ Process Synchronisation.
↳ for one point of time only one process can access
- Require it ?
  - ↳ bcz if our process are not scheduled/synchronised in a right way. So it can delete the things, overwrite the things or give the wrong result.

- Race condition: when 2 or more processes are trying to extra certain things or execute certain parts. then if resultant the value is dependent upon order of execution, how they are executing on the basis of their execution their resultant value depend.

Unit :- 05.

Deadlocks & Memory management.

⇒ Hardware Based solution

1. disabling of Interrupts.
2. Test and set lock (TSL):

```
boolean TSL(boolean &target){
        boolean rv = target;
        target = true;
        return rv;
    }
do {
    while (TSL(lock));
    critical section
    lock = false;
    remainder section.
}
```

PAGE NO.
DATE. / /

cesses
ertain
result-
den
on the
t value

Algo 2: TSL using Swap

```
void swap (boolean &a, boolean &b){
        boolean temp = a;
        a = b;
        b = temp;
}
n shared data (initialized to false):
        boolean lock;
        boolean waiting[n];
n process Pi
do {
        key = true;
        while (key == true)
            swap (lock, key);
        critical section
        lock = false;
        Remainder section.
}
```

⇒ Critical section [variable, Hardware, counter, process / which sharing b/w 2]
conditions :

① Mutual Exclusion : If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

② Progress : If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the

critical section next cannot be postponed indefinitely.

③ Bounded waiting: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

- Assume that each process executes at a non-zero speed
- No assumption concerning relative speed of the n processes.

Initial attempts to solve problem.

- Only 2 processes, P_0 and P_1
- General structure of Process P_i (other process P_j)

```
do {
        entry section
            critical section
        exit section
            reminder section
}
while (1);
```

> Process may share some common variables to synchronize their actions.

be postponed.

ust exist
ther processes
critical
ade a request
nd before that

cutes at a

ute speed of the

blem.

(other process

non variables

Algo 1.: Using Turn variable.

> shared variables :
int turn;
initially turn = 0
turn = i ⇒ P_i can enter its critical section
> Process P_i
do {
while (turn != i);
critical section
turn = i;
Remainder section.
}
while (1);
> satisfies mutual exclusion, but not progress coz due to first process, second process can't processed so this Halt the progress.

Algo 2:
> shared variables
boolean flag [2];
initially flag [0] = flag [1] = false
flag [i] = true ⇒ P_i Ready to enter its critical section.
> Process P_i
do {
flag [i] = true;
while (flag [j]);
critical section
flag [i] = false;

PAGE NO.
DATE. / /

remainder section
} while (1);
> Satifies mutual Exclusion but progress not satifies.

Algo 3: Peterson soln.
> Combined shared variables of algorithms 1 and 2.
> Process P_i

```
do {
    flag [i] = true;
    turn = j;
    while ( flag [j] and turn = j );
                critical section.
    flag [i] = false;
            remainder section
}
while (1);
```

> It meets all the requirements.

⇒ Semaphores. [it's a variable].
① Binary Semaphore (mutex) [It is provide a mutual exclu
② counting semaphore.

```
wait (S):
    while S<0 do no-op;
            S--;
signal (S):
            S++;
```

Countin
Define
type
int
stru
} se

assume
block
wakeu
blocked
semaph
wait (
S.value
if

signal
S.valu

⇒ Deadl
Dead

PAGE NO.
DATE. / /

t progress

algorithms 1 & 2

);

s.

exclu-
provide a mutual .

Counting Semaphore without spinlock

Define a semaphore as a record

```
typedef struct {
int value ;
struct process *L;
} semaphore;
```

assume two simple operations :
block suspends the process that invokes it
wakeup (P) resumes the execution of a blocked process P.
semaphore operations now defined as

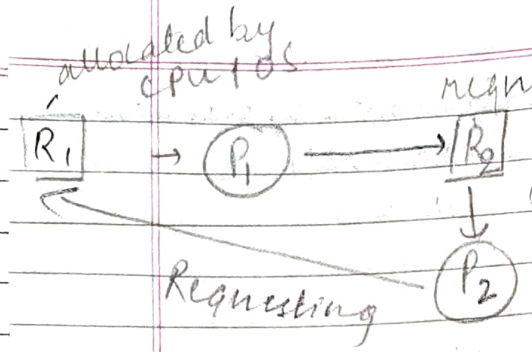
```
wait (S) :
S.value --;
    if (S.value < 0) {
            add this process to S.L;
        block;
    }
signal (S) :
S.value ++;
    if ( S.value <= 0) {
            remove a process P from S.L;
        wakeup (P);
    }
```

⇒ Deadlock Resource crisis.

Deadlock characteristic (Behaviour)
- ↳ Mutual Exclusion
- ↳ Hold and wait
- ↳ No prempton
- ↳ circular wait .

These conditions are occur simultaneously than only Deadlock occurs.

DATE. / /



when two or many process are waiting to happens but it never happens called deadlock

Process → P_i
Resource → R_1

when R_1 having more than one instance then -

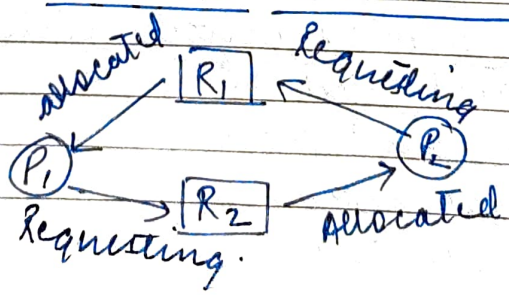
R_1 : Two instance
R_1 . one instance

Requesting : $P_1 \rightarrow R_1$ Requesting single instance of R_1

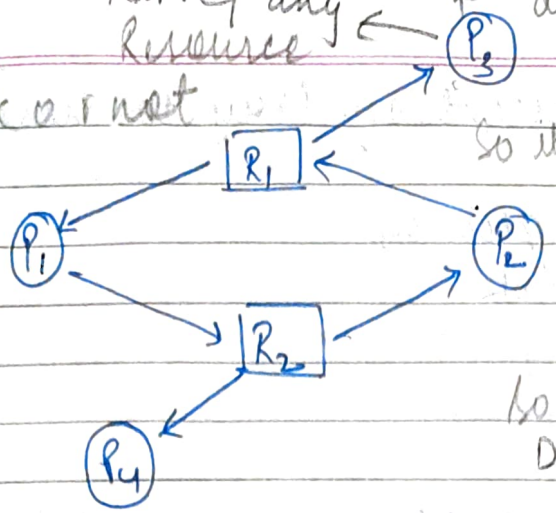
$P_1 \Rightarrow R_1$ Re. Double instance of R_1

Allocation : $P_1 \leftarrow R_1$

RAG :- Resource Allocation Graph



* if cycle detect then deadlock might occur



It doesn't require anything it simply allocated R_1 So it isn't in waiting state it is in execution

Deadlock or not

So there is no Deadlock.

Ques- consider a system which has n process and 6 Resources Each process requires 2 resources to complete their Exec^n then, what is the max value of n which ensures deadlock free operation.

Total = 6 Resources.

$n \longrightarrow 2R$ $\quad \leq 6$

Let as assume. $\quad \neq 2$

$\{ P_1 \; P_2 \}$ Take

$2 \quad 2$

P_1	P_2	P_3	
2	2	2	no deadlock.

Then,

P_1	P_2	P_3	P_4	
2	2	1	1	no deadlock

6 5 1 remai.

P_1	P_2	P_3	P_4	P_5
1	1	1	1	1

executed process no deadlock.

completed & free up

6 0

P_1	P_2	P_3	P_4	P_5	P_6
1	1	1	1	1	1

Here, no process get executed so, there is deadlock $n = 1$.

Deadlock prevention (Try to dissatisfy the condition).

- Mutual Exclusion
- Hold & wait
- No preexemption
- Circular wait

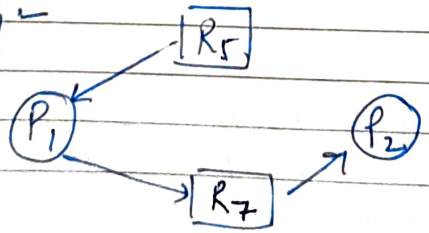
- Shared resources such as read-only files do not lead to deadlocks.
- Unfortunately some resources such as printers and tape drives, require exclusive access by a single process.

- To prevent this condition processes must be prevented from holding one or more resources while simultaneously waiting one or more others. There are several possibilities for this. Require that all process request all resources at one time.

- A resource can be released only voluntarily by the process holding it.

- Resource should be assigned by unique number and process can request for resources in increasing order.

Eg:-



Deadlock A
Resources : A

Process.
Max Need of Resources for its comp.

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Res. cur
inst
at
no

Remaining ne

P1 — P3

current

Deadlock
(1) If the r
type →
(2) If the r
type.
L
L
sequen
Otherwi
sequen

dissatisfy the

read-only
ocks
ces such as
require
process.

ocesses must
e or more
ynating one
several
at all process
time.

only volunt-
it.

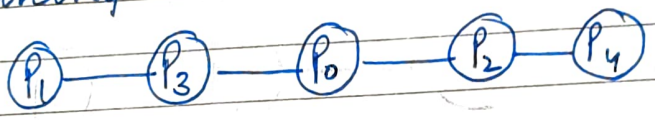
ed by unique
request for
es.

Deadlock Avoidance (Banker's Algorithm)

Resources : A, B, C

Process	Max Need of Resources for its comp.			Current Resource allocation already provided to the process.			Current Available Resource instance			Remaining Need.		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	3	3	2	7	4	3
P_1	3	2	2	2	0	0	2	0	0	1	2	2
P_2	9	0	2	3	0	2	5	3	2	6	0	0
P_3	2	2	2	2	1	1	7	4	3	0	1	1
P_4	4	3	3	0	0	2	7	5	3	4	3	1
							10	5	7			

Remaining need = max need − current allocation



safe sequence.

current available = 1057.

Deadlock detection

① If the resources are of single instance type. → If a cycle exist in RAG

② If the resources are of multiple instance type.
- → Just apply Banker's Algorithm
- → If sequence we got is safe sequence then no Deadlock.

Otherwise Deadlock occur (unsafe sequence)