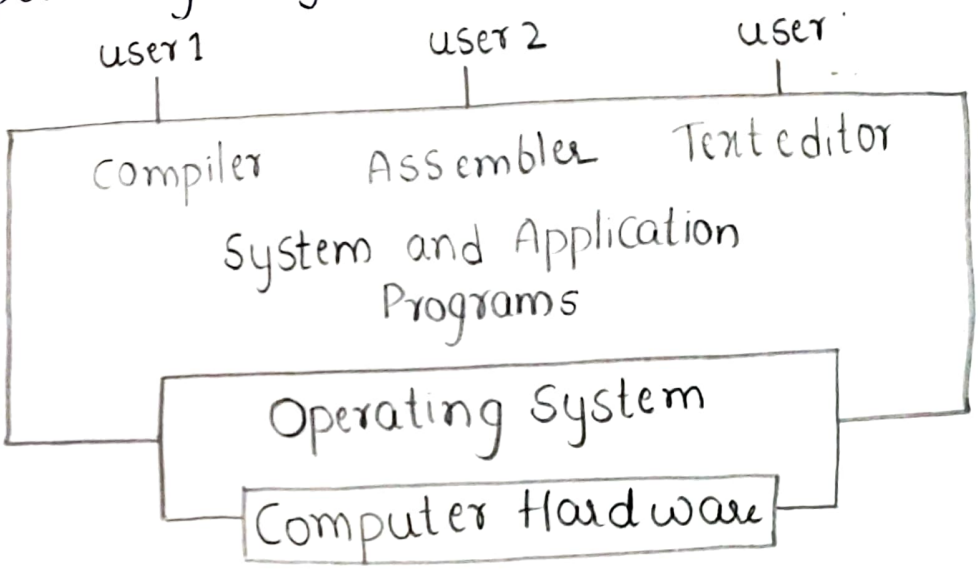
7/1/2020

UNIT - 1

Introduction to Operating System
And
Introduction to UNIX File System

What Operating Systems do?



OS is an interface between computer hardware and software (user)

or

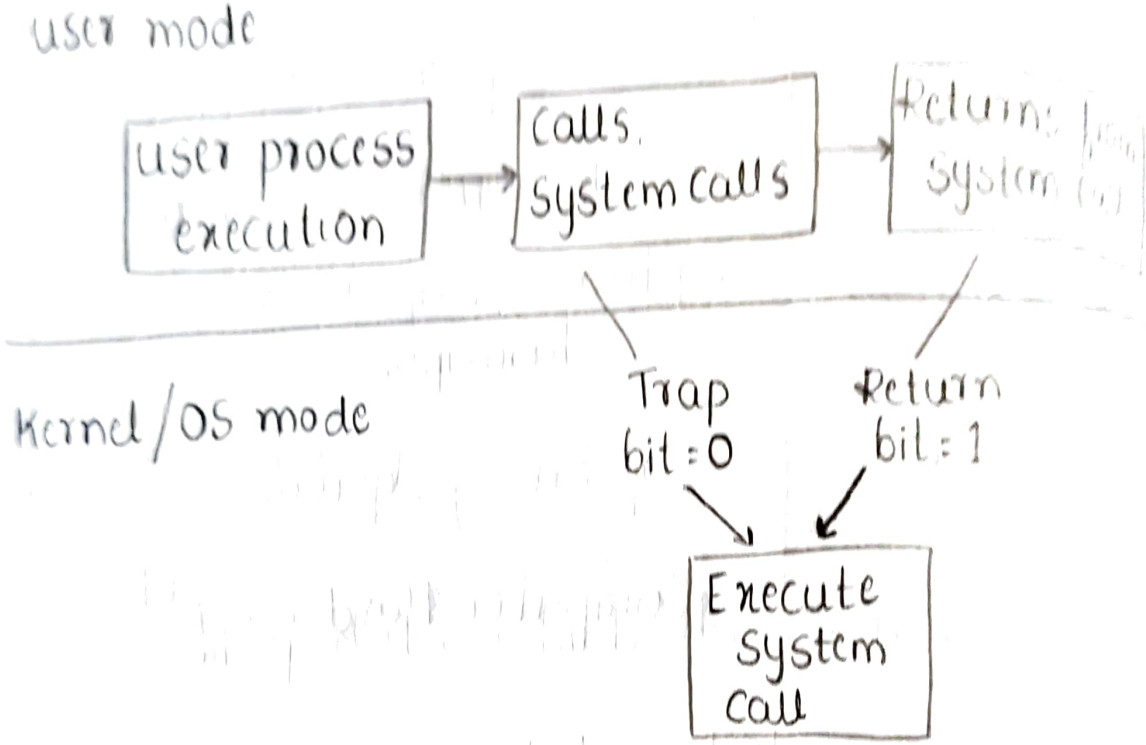
OS provides the means for proper use of or controls and co-ordinator the use of the hardware among the various application programs for the various users

9/1/2020

User view — resource utilization

System view — resource allocation

OS operations



To differentiate OS mode and user mode a mode bit is used if it is 0 - OS mode

1 - user mode

Times:

Fixed - $1/60$ sec

Variable - 1ms - 1sec

OS functions

* Process Management :
  - create, delete process
  - process synchronisation
  - process deadlock
  - process communication

- Create, delete user and system processes
- providing mechanisms for process synchronisation
- providing mechanisms for process communication
- providing mechanisms for deadlock handling
- Suspending and resuming processes

[process is the program i.e. currently executing]

* Memory Management :

1 Track ~~of~~ which part of the memory is used by whom
2 decide which program and data to move in and out of memory
3 allocate and deallocate memory space as and whe needed.

* Storage Management :

1 File Management
2. Mass Storage Management
3. Cache Management (caching)

1 File management
- Creating and deleting files
- Creating and deleting directories to organise files
- Supporting primitives for manipulating files and directories for mapping files onto secondary storage
- backing up of data

2 Mass storage Management
- Storage allocation
- free space management
- disc scheduling

3 Caching:
- maintaining data which are cached

I/O systems:
- General device driver interface
- buffering, caching and spooling of data
- drivers for specific hardware devices

10/1/2020

Protection and Security:

Distributed System

A distributed system is a collection of physically seperate possibly heterogeneous computer systems that are networked to provide the users with access to the various resources that the system maintains

LAN - Local Area Network (systems connected with a room)

WAN - Wide Area Network (systems connected with building)

MAN - Metropolitan Area Network (systems connect within city/other buildi

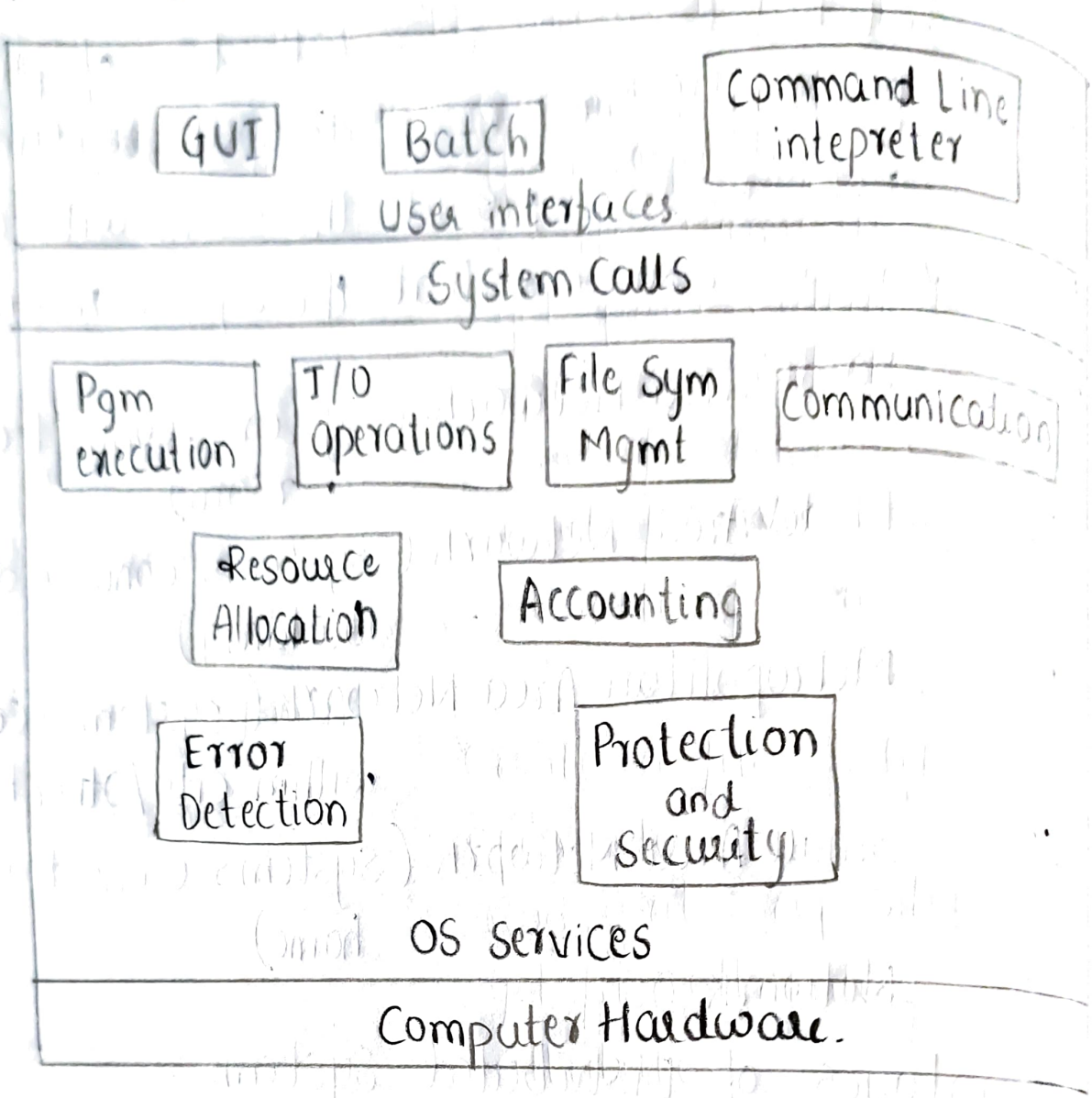
SAN - Small Area Network (systems connected with home)

NOS - Network OS

Advantages of Distributed systems:

- Resource sharing
- increases computation speed
- reliability
- data availability etc.

Operating System Services



System Calls :

Systems calls are interfaces to operating system services

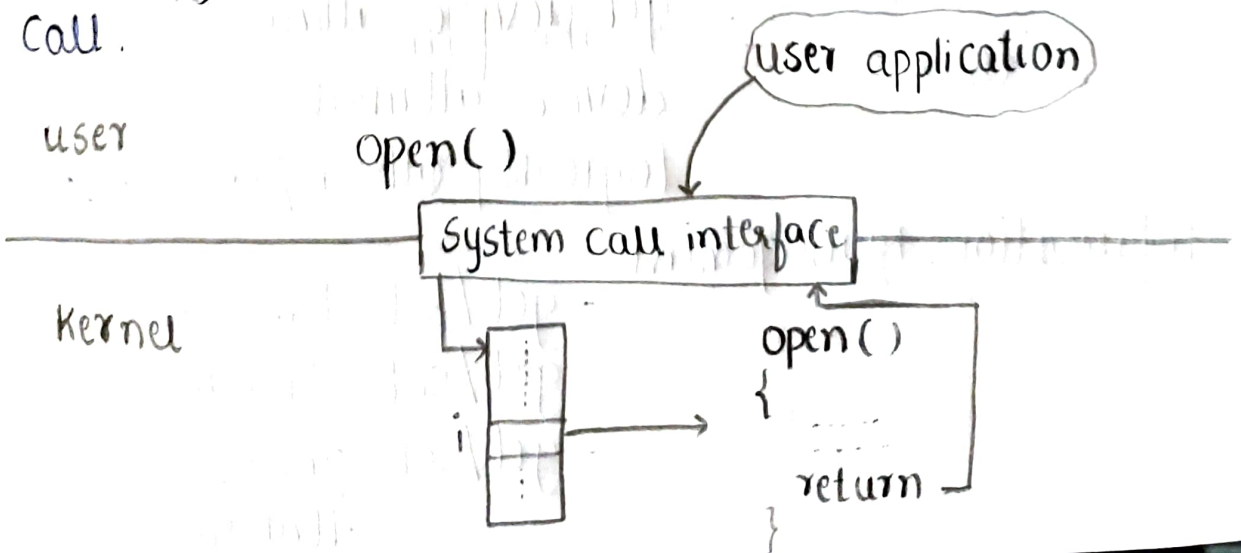
* A sample program to read data from one file and copy them to an another file.

Source file ⟶ Destination file

- Acquire i/p file name
  write prompt to screen
  accept i/p file name
- Acquire o/p file name
  write prompt to screen
  accept o/p file name
- Open i/p file
  if file does not exist, abort
- Create o/p file
  if file exists, abort
- Loop
  read from i/p file
  write to o/p file.
- until read fails.
- close o/p file.
- write completion message to screen
- terminate normally

14/1/2020

* Handling a user applicatn invoking open() system call.



Types of System Calls

1 Process control : It should have system calls
to - end, abort
- load, execute
- create, terminate
- get process attributes, set process attributes
- wait for time,
- wait for event, signal event
- allocate, free ~~mori~~ memory.

2 File management :
- create a file, delete
- open, close
- read, write, reposition
- get file attributes, set file attributes

3 Device Management :
- request release
- read, write, reposition
- get device attributes, set device attributes
- logically attach / detach devic

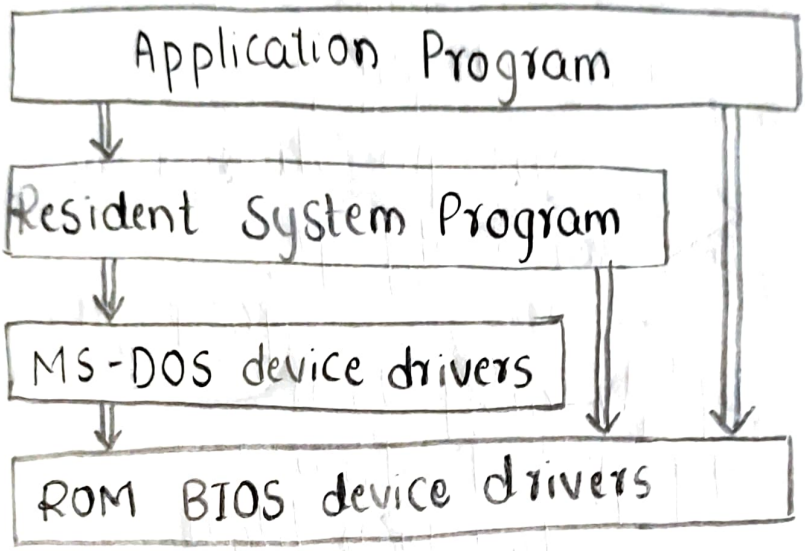
4 Information Management :
- get / set time, date
- get / set system data
- get / set process / file / dev attributes.

5 Communications:

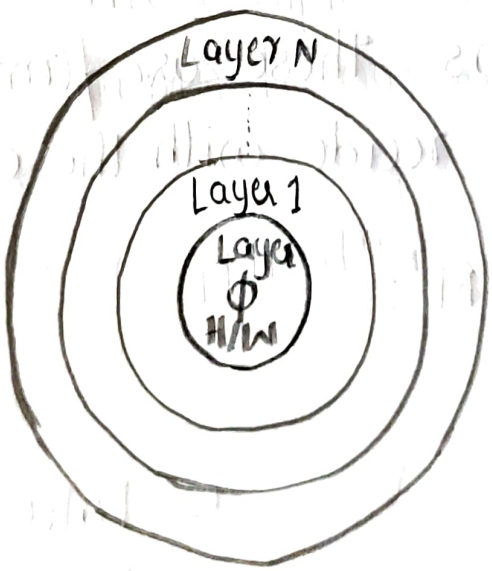
- create, delete communication connectⁿ
- send, receive messages
- transfer status info
- attach / detach remote devices.

OS Structure:

1 Simple Structure: (MS-DOS)



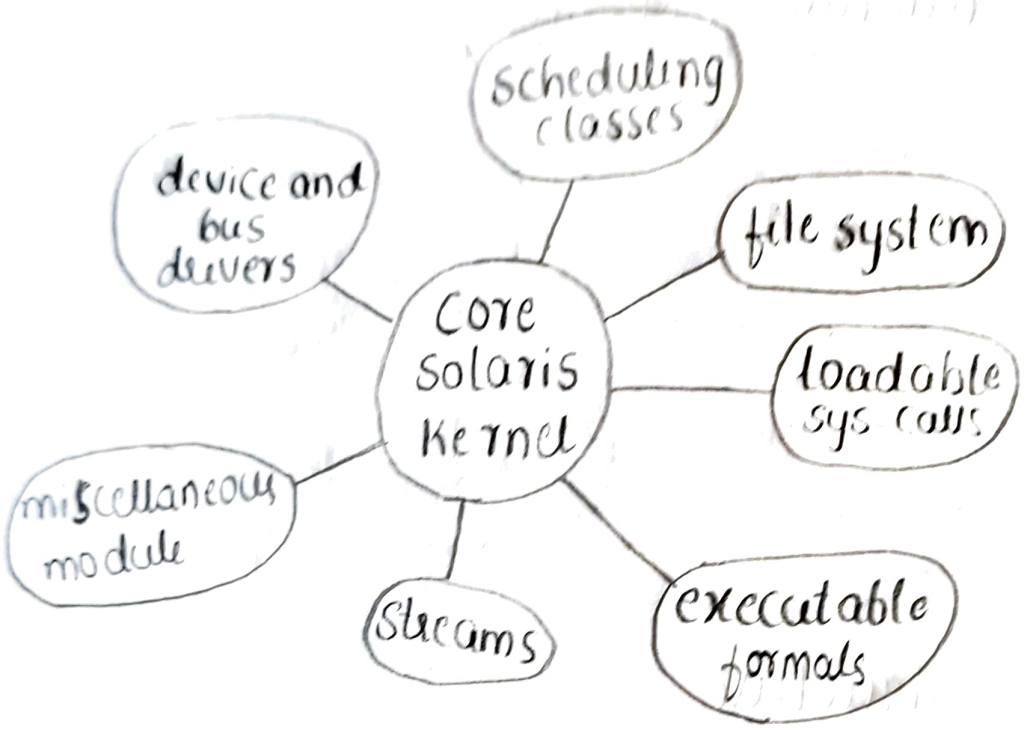
2. Layered Structure (UNIX)



16/1/2020

3. MicroKernels (Mini OS)
   msg Passing

4. Modules (OOPs concept is used)



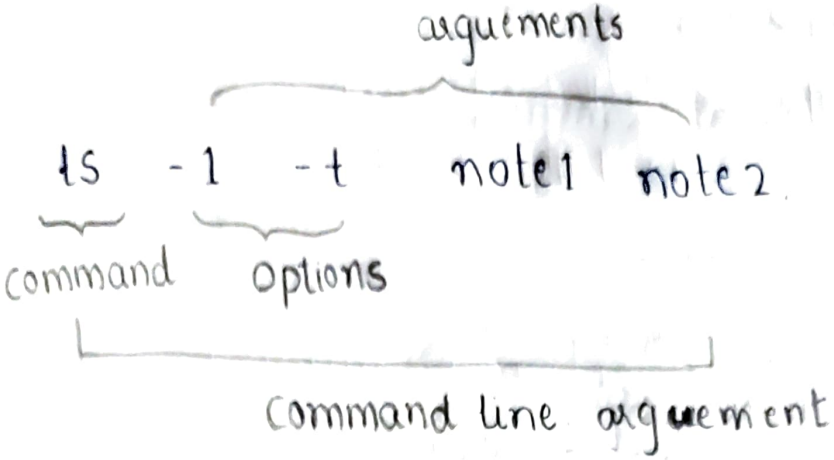
Bootstrap programs: These are firmware, progra
recide with the chip

UNIX FILE SYSTEM:

Inside Unix:

1. multi-user
2. multitasking
3. Do one thing well
4. The featureless file
5. Pattern Matching
6. Programming facili
7. Portability and Sys
   Calls
8. Documentation

Command structure



11/1/2020

* Internal and external commands

ls - /bin or /user/bin

echo - build in

Eg : ls - lists all the command

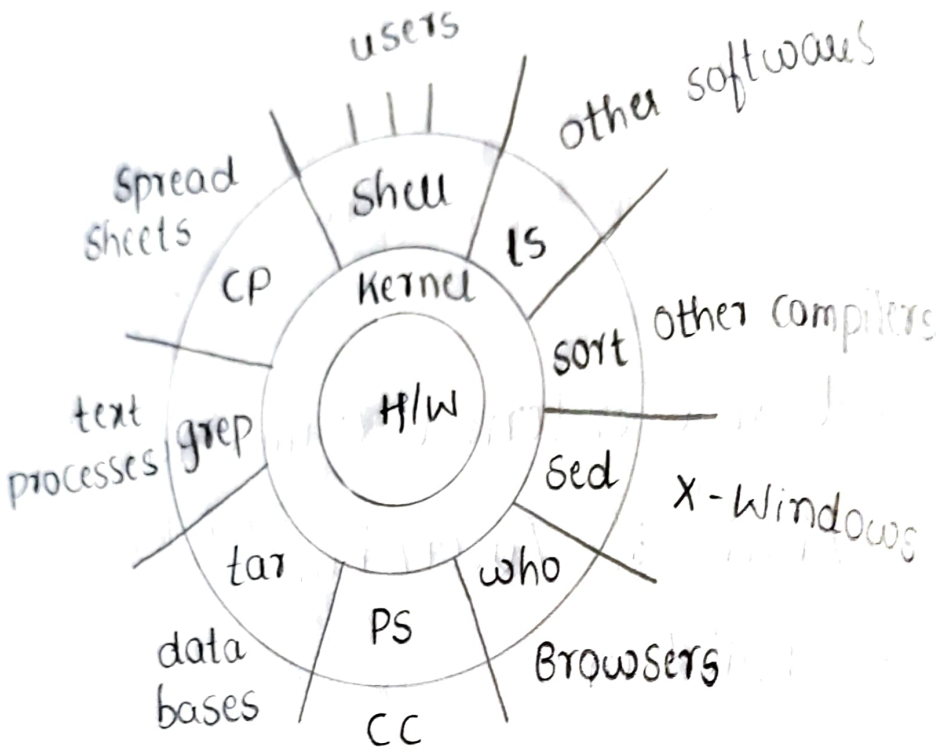
ls -l lists all files allied today

ls chap1 chap2

Kernel, Shell Relationship (after bootstrap prog)

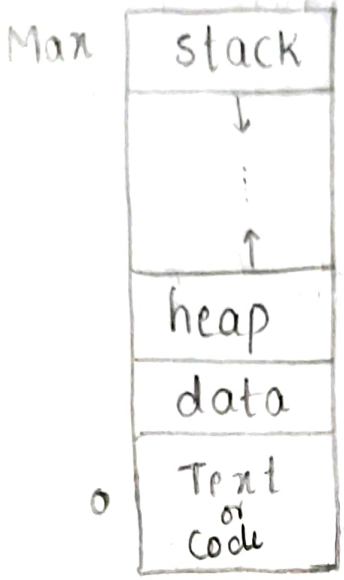
/bin/bash

or file/boot/vmlinuz



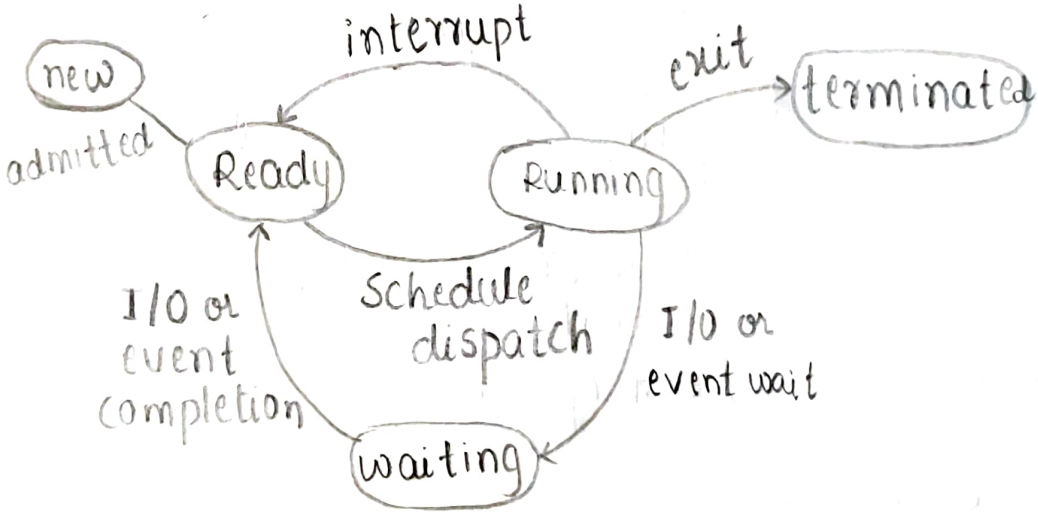
UNIT-2

* Process : Program which is under execution is called process



program - passive entity
process - active entity.

Process State Diagram : indicates diff. state of process.



1. new : The process is being created.
2. ready : The process is waiting to be assigned to a processor.
3. running : instructions of the process are being executed

4 waiting :

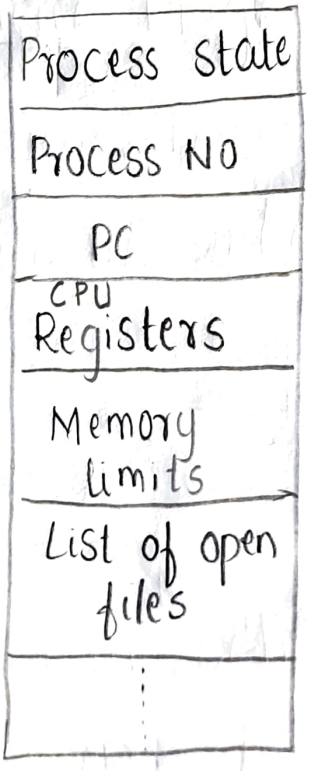
The process is waiting for some event to occur such as an i/o completion or reception of a signal or wait for the child processes to complete execution

5 laminated.

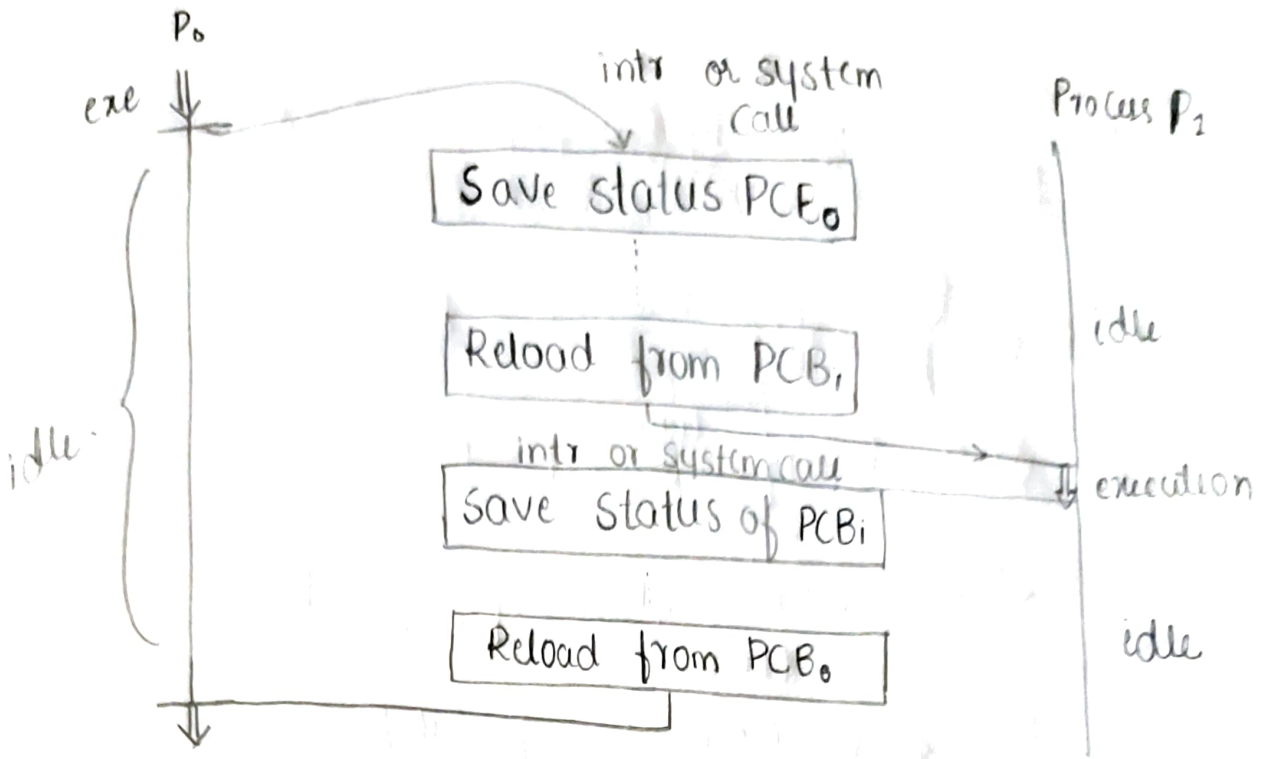
The process has finished execution

18/1/2020

Process control Back



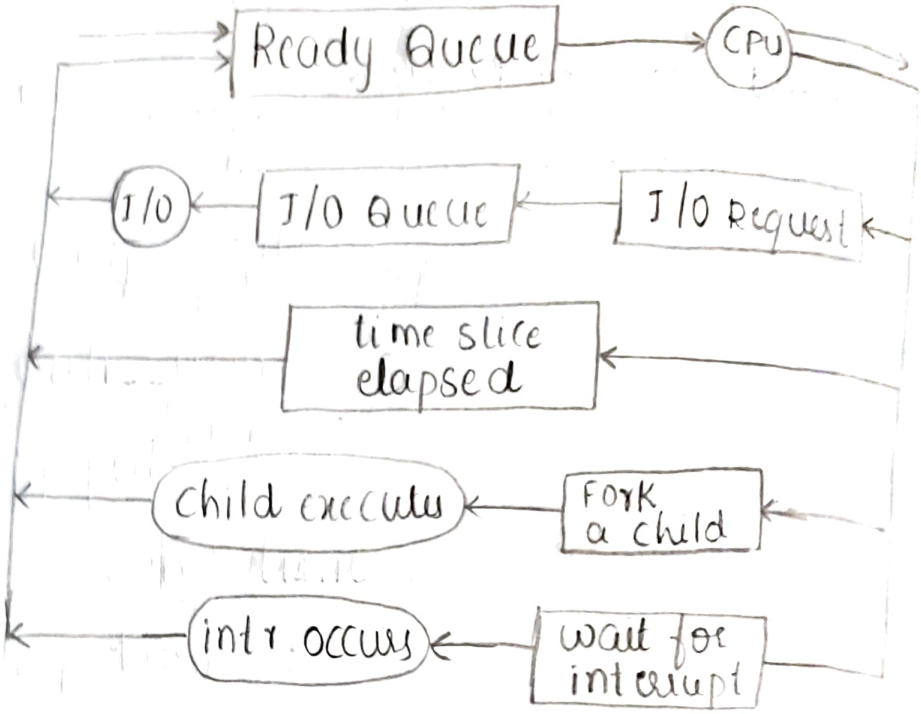
* CPU switch from process to process.



- Each process is represented in the OS by a process control block also called task control block (PCB)
- A PCB contains the following info.

i. Process State
ii. Process NO.
iii. PC
iv CPU Registers
v Memory limits / management info.
vi ~~List of open files~~ / CPU scheduling info
vii. Accounting info.
viii. I/O status info.

* Scheduling Queues :



Queue Diagram.

Job queue, Ready queue, Device queue

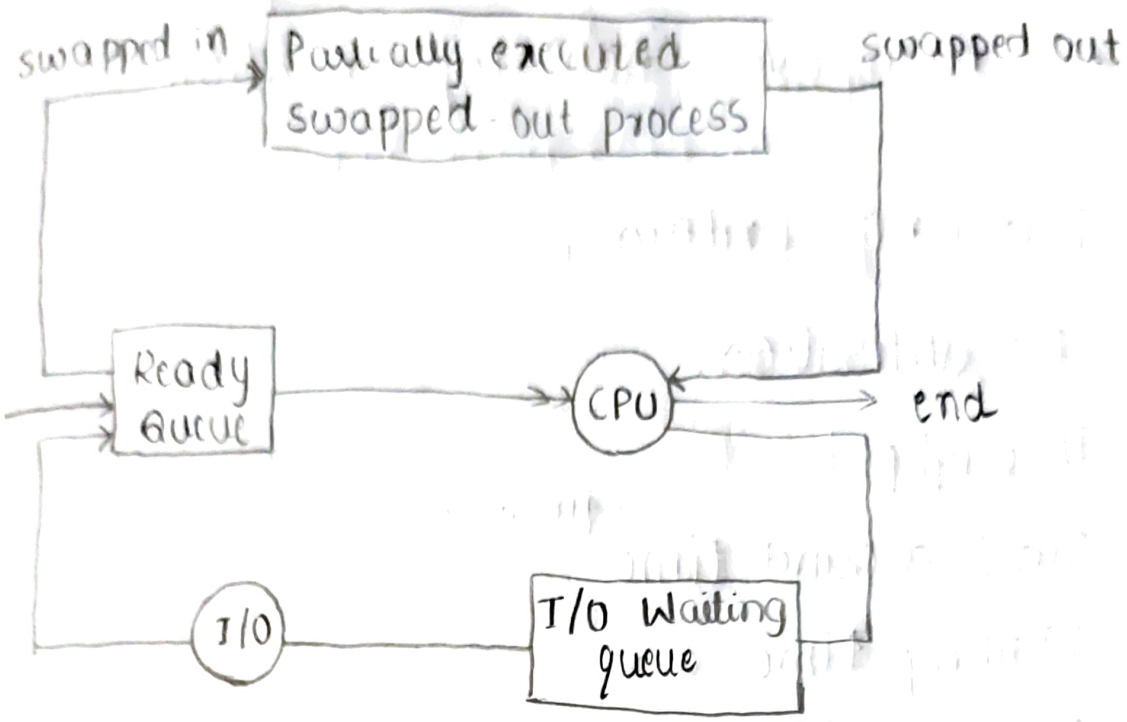
Thread - single flow of control.

21/1/2020 :

Schedulers :

1 Long-term : It will be scheduled only once
- It will Send process to memory Control multi-programming.
2. Short term or CPU : It will be scheduled again and again.

Medium-term scheduler



Context switch : The ~~processor hc~~ switching of processor from one process to another by saving the contents of first process

- Instruction which require CPU time are called CPU bound instructions.
- Instruction which require use of I/O device are called I/O bound

CPU bound
=
I/O bound

CPU bound
=
I/O bound

⇒

CPU bound → end with CPU bound.

2 Major types of scheduling

1. Preemptive : execution with pauses/stop
2. Non-preemptive : non stop execution.

* Scheduling Criteria :

1. CPU utilisation.
2. Throughput - finds out how many processes finished within a time
3. Turn around time
4. Waiting time
5. Response time

1. CPU utilisation : Keeping the CPU busy as as possible. from 0 to 100%.
2. Throughput : The no. of processes that completed per time unit
3. Turn around time : The time interval from time of submission of a process to the time of completion is turn around time.
4. Waiting time : This is the sum of time spent waiting in the ready queue

5. Response time: This is the time from the submission of a request until the first response is produced for a process

Scheduling - Algorithms:

1. First come first served scheduling:

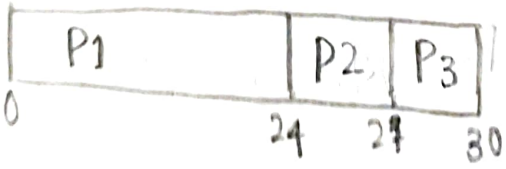
The process that requests the CPU ~~pa~~ first is allocated to the CPU first

23-01-2020

1

Process	Burst Time
P_1	24
P_2	3
P_3	3

Gantt Chart



Waiting time for P1 = 0 ms.

" " " P2 = 24 ms

" " " P3 = 27 ms.

Avg. WT: $(WT(P1) + WT(P2) + WT(P3))/3$

$= (0+24+27)/3 = 17$ ms.

Turnaround time of P_1 = 24 ms

" " P_2 = 27 ms

" " P_3 = 30 ms

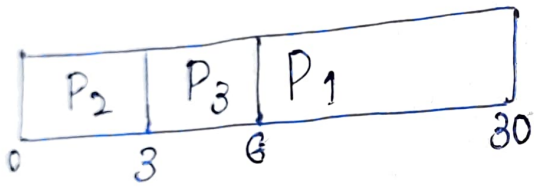
Avg TAT = (TAT(P_1) + TAT(P_2) + TAT(P_3)) / 3

= (24 + 27 + 30) / 3

= 27 ms

2

P_2	3
P_3	3
P_4	24



Waiting time of P_1 = 6

" " P_2 = 0

" " P_3 = 3.

Avg = (6 + 0 + 3) / 3

= 3 ms.

Turnaround time of P_1 = 30 ms

" " P_2 = 3

" " P_3 = 6.

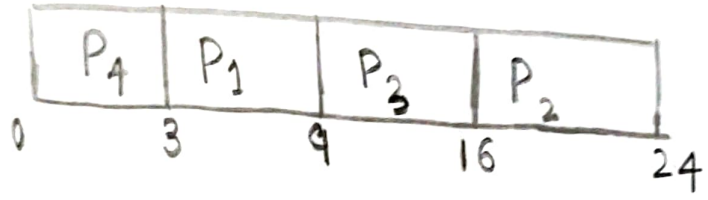
Avg Turn around time = (8+3+6)/3

= 13ms.

2 Shortest Job First :

The process which has the shortest CPU burst time will be allocated to the CPU first

Process	Burst
P_1	6
P_2	8
P_3	7
P_4	3



Waiting time of P_1 = 3

" " P_2 = 16

" " P_3 = 9

" " P_4 = 0

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

= 7

Turnaround time of P_1 = 9

P_2 = 24

P_3 = 16

P_4 = 3

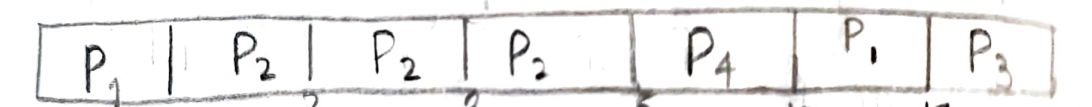
Avg. turn around time = (9 + 24 + 16 + 3) / 4

= 13.

Process Arrived ti

3. Shortest Remaining Time / Job First.

Process	Arrived time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



$P_1 = 7$ $P_1 = 7$ $P_1 = 7$ $P_1 = 7$ $P_1 = 7$ $P_3 = 9$

$P_2 = 4$ $P_2 = 3$ $P_2 = 2$ $P_3 = 9$ $P_3 = 9$

$P_3 = 9$ $P_3 = 9$ $P_4 = 5$

$P_4 = 5$

Waiting time of P_1 = 9

" " " P_2 = (1 - 1) = 0

Waiting time of P_3 = (17 - 2) = 15

" " " P_4 = (5 - 3) = 2

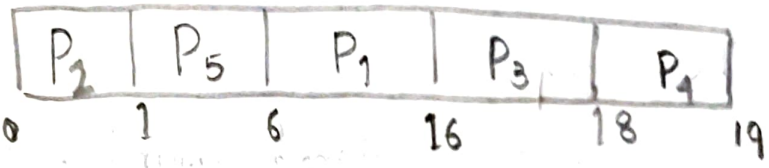
Avg waiting time = $\frac{9+0+15+2}{4}$ = 6.5

Turnaround time = completion time - arrival time

Waiting time = turnaround time - burst time.

4. Priority Scheduling:

Process	Priority	Burst Time
P_1	3	10
P_2	1	1
P_3	4	2
P_4	5	1
P_5	2	5



WT for P_1 = 6

P_2 : 0

P_3 : 16

P_4 : 18

P_5 : 1

Avg. WT : (6+0+16+18+1)/5 : 8.2ms.

Turn around time of P_1 = 16

P_2 = 1

P_3 = 18

$P_4 = 19$

$P_5 = 6$

Avg. Turn around time $= (16 + 1 + 18 + 19 + 6)$

$= 12$ ms.

5. Round Robin Scheduling :

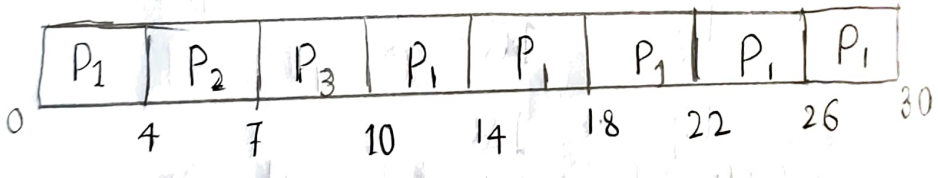
In this algorithm every process is given a fixed small unit of time to execute as they arrive into the ready queue

24-1-2020

Eg :

Process	CPU burst time.
P_1	24
P_2	3
P_3	3

Time quantum = 4



WT of $P_1 : (10 - 4) = 6$

$P_2 = 4$

$P_3 = 7$

Avg WT : (6+4+7)/3

: 17/3

: 5.6ms

Turn around time : T_1 : 30

T_2 : 7

T_3 : 10

Avg TAT : (30+7+10)/3

: 15.6 ms.

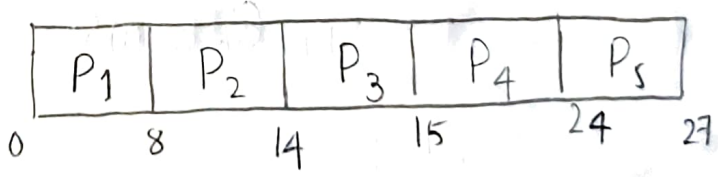
Eg :

Process	CPU burst	Priority	Arrival Time
P_1	8	4	0
P_2	6	1	1
P_3	1	2	2
P_4	9	2	3
P_5	3	3	4.

2 ms

FCFS :

~~WT for P_1~~



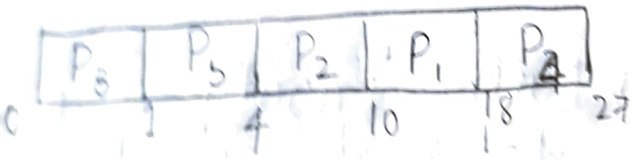
WT of P_1 : 0
P_2 : 8
P_3 : 14
P_4 : 15
P_5 : 24

TAT of P_1 : 8
P_2 : 14
P_3 : 15
P_4 : 24
P_5 : 27

Avg WT : (0+8+14+15+24)/5
: 12.2

Avg TAT : 88/5 = 17.6

2. SJF



WT of P_1 : 10
P_2 : 4
P_3 : 0
P_4 : 18
P_5 : 1

Avg WT of P : (10+4+0+18+1)/5
= 6.6 ms.

Turn around time of P_1 : 18

P_2 : 10

P_3 : 1

P_4 : 21

P_5 : 4

Avg TAT : (18+10+1+21+4)/5

: 12

SRTF

Waiting time of P_1 : 11-1 : 10

P_2 : 1 + 3 : 4

P_3 : 0

P_4 : 18

P_5 : 0

Avg WT : (10+4+0+18+0)/5 : 5.8

TAT of $P_1 = 18 - 0 = 18$

$P_2 = 11 - 1 = 10$

$P_3 = 3 - 2 = 1$

$P_4 = 27 - 3 = 24$

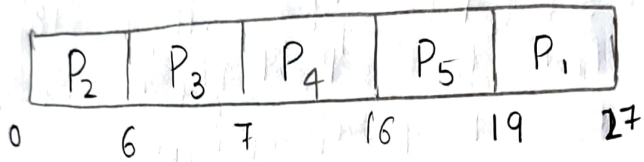
$P_5 = 7 - 4 = 3$

Avg TAT : $(18+10+1+24+3)/5$

$= 11.2$ ms

4 5 Avg WT 9.6

4 Priority



WT of $P_1 = 19$

$P_2 = 0$

$P_3 = 6$

$P_4 = 7$

$P_5 = 16$

Avg WT : $(19+0+6+7+16)/5$

: 9.6 ms

TAT of $P_1 = 27$

$P_2 = 6$

$P_3 = 7$

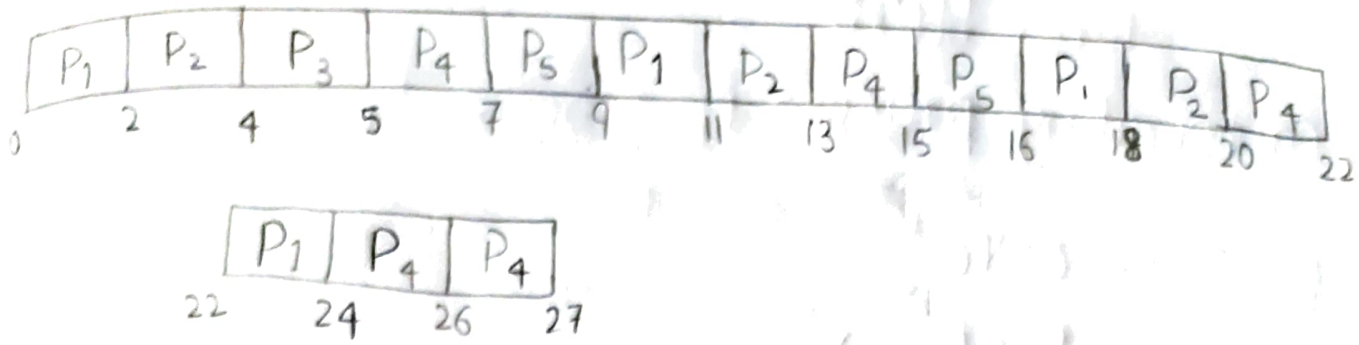
$P_4 = 16$

$P_5 = 19$

Avg TAT : $(27+6+7+16+19)$

$= 15.$

5 Round Robin



WT of P_1 = (9 - 2) + (16 - 11) + (22 - 18)

= 7 + 5 + 4 = 16

P_2 = 2 + (11 - 4) + (18 - 13)

= 2 + 7 + 5

= 14

P_3 = 4

P_4 = 5 + (13 - 7) + (20 - 15) + (26 - 22)

= 20

P_5 = 7 + (15 - 9)

= 13

Avg WT = (16 + 14 + 4 + 20 + 13) / 5 = 13.4

TAT of P_1 = 24

P_2 = 20

P_3 = 5

P_4 = 27

P_5 = 16

Avg TAT = (24 + 20 + 5 + 27 + 16) / 5 = 18.4

PROCESS SYNCHRONISATION

Co operating process

Producer code

```
while (true)
{
    while (counter == Buffer_size);
    Buffer[in] = next_produced;
    in = (in+1) % Buffer_size;
    counter++;
}
```

Consumer Code

```
while (true)
{
    while (counter == 0);
    next_Consumed = Buffer[out];
    out = (out+1) % Buffer_size;
    counter--;
}
```

Race

A situation where several processes access and manipulate the same data con-currently and the outcome of the execution depends on the particular order in which the access takes place is called race condition

Critical Section:

Each process has a segment of code called critical section in which the process may be changing common variables, updating a table, writing a file and so on

Critical section problem:

General structure:

```
do {
        Entry section
            Critical section;
        Exit Section
            Remainder section;
} while (true);
```

1 Critical section must specify their requirements as stated below:

1 Mutual exclusion : If process P_i is executing 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 some processes wish to enter their critical sections then only those processes that are not executing in their remainder section can participate in the decision on which will enter next

3. Bounded waiting : There exists a bound or limit on the no. of times that other processes are allowed to enter their critical sections after a process makes a request

7/2/2020.

* Approaches

1. Premptive kernels - good for co-operating procs
2. Non Premptive kernels.

* Peterson's solution.

Process structure P_i : int turn. boolean flag;

```
do {
    flag[i] = TRUE;
    turn = j
    while (flag[j] && (turn == j));

    Critical section

    flag[i] = FALSE;

    remainder section;
} while(TRUE);
```

Synchronisation Hardware :

```
do {
    Acquire lock
        Critical condition ;
    Release Lock
    Remainder section ;
} while(TRUE);
```

8/2/2020

Semaphores :

```
wait(s)
{
    while(s<=0);
    s--;
}

signal(s)
{
    s++;
}
```

Two types :

i. Binary - counts only betⁿ 0 and 1 - generally used for implementing process

ii. Counting - used for resources/instances attached to the system

* Mutual Exclusion implementation using Binary Semaphore.

```
do {
        waiting(mutex);
                Critical Section;
        signal(mutex);
                remainder section;
} while(TRUE);
```

A semaphore 's' is an integer variable that apart from initialisation ~~whose~~ is accessed only through 2 standard atomic operations wait and signal

wait is represented as P → to test

signal " " V → increment.