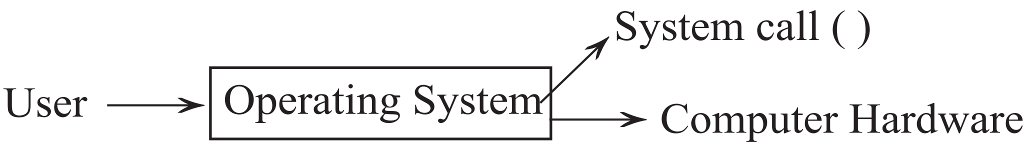
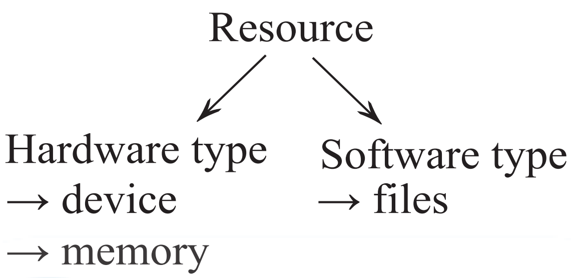
Operating System



OS is the interface between user and computer Hardware.
System call is the request made by user program in order to get the service from an OS.
OS works likes a resource allocator.
OS is responsible for allocating all the resources of the computer program.

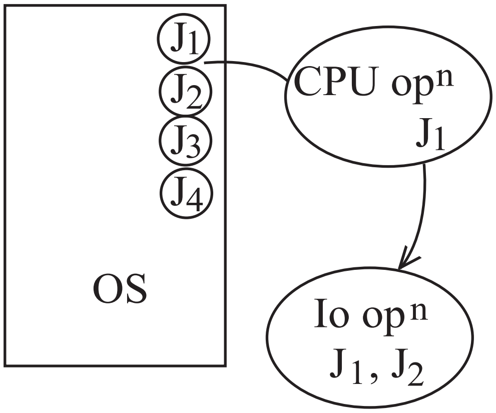


OS is like as a government which covers everything in the computer system.

Goals of OS
Primary Goal → Convenience (Easy to use)
Secondary Goal → efficiency

Types of OS

1. Batch OS



→ If the job is completed completely then only another job will be scheduled

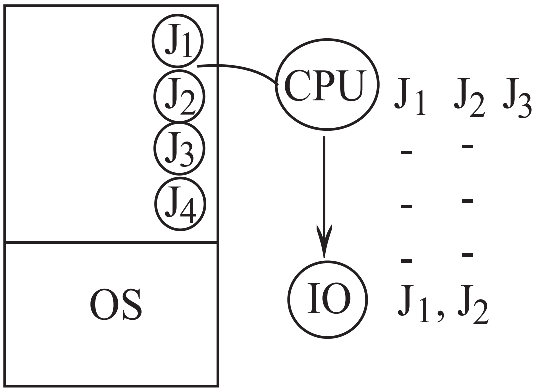
→ So, CPU utilization is very less, hence throughput of the system will ↓

Utilization ∝ throughput

Throughput: The number of jobs completed or executed per unit of time is called throughput.

Eg:- IBM OS/2

2. Multiprogramming:



→ If one job is waiting for I/O transfer then a another job is ready to utilize the CPU

Ex:- Windows, UNIX

→ CPU utilization is very high, so throughput will be ↑.

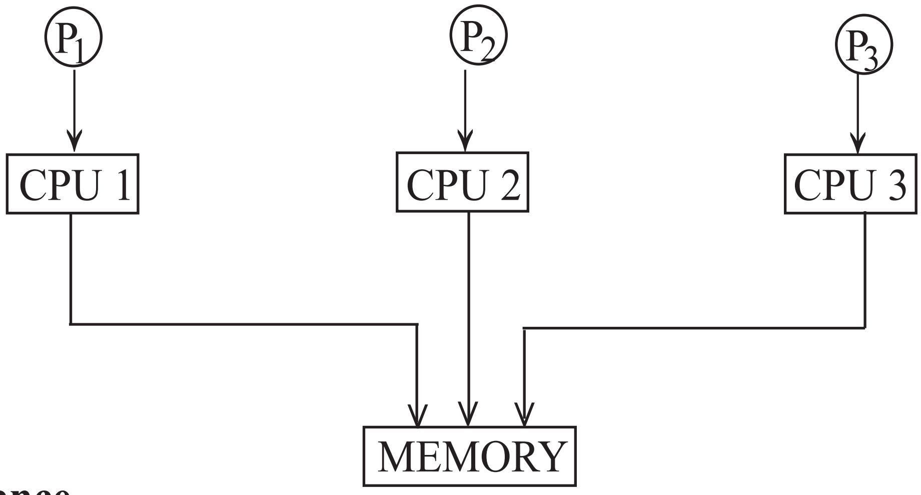
3. Multitasking OS:

→ It is the extension of mutiprogramming OS.

→ In this OS, jobs will be executed in the time sharing mode.

Ex:- Windows 2000, LINUX

Multiprocessing OS



Fault tolerance

In this OS, more than one CPU share the single computer system memory.

Advantages:

→ Fault tolerance

→ Reliability (end to end)

→ CPU utilization is very high

Real time OS

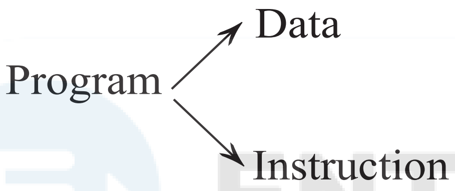
The system which are strict deadly time bound.

Eg:- Missile system (Hard real time OS)
Banking system (Banking sector)

PROCESS MANAGEMENT

→ Process is the program under execution.

→ Program should be reside in main memory to occupy the processor to execute the instruction.



Attributes

→ process_id

→ process state

→ Program counter

→ Priority

→ General purpose register

→ List of open devices

→ List of open files

→ Protection & security

States

New
Ready
Running
Wait/Block
Suspended ready
Suspended wait
Termination or completion

→ process_id → It is unique identification number which is assigned by the OS at the time of process creation.

→ process state → It contains the current state information of the process where it is residing.

→ program counter → It is a register, it contains the address of next instruction to be executed.

→ Priority → It is a parameter which is assigned by the OS at the time of process creation.

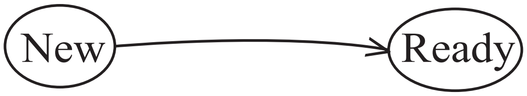
All the attributes are called as context and context of the process is stored in PCB (Process Control Block).

→ Every process will have its own PCB and PCB of the process is stored in the main memory.

Process State Diagram:

→ Initially the process will be in new state.

→ The process is in new state, it means process is under creation or process is being created.

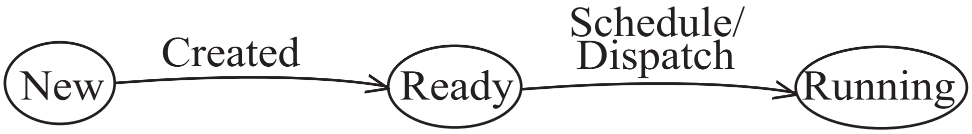


→ Once the process is created, then it move to ready state.

→ Once the process is in ready state, it means process is really for execution.

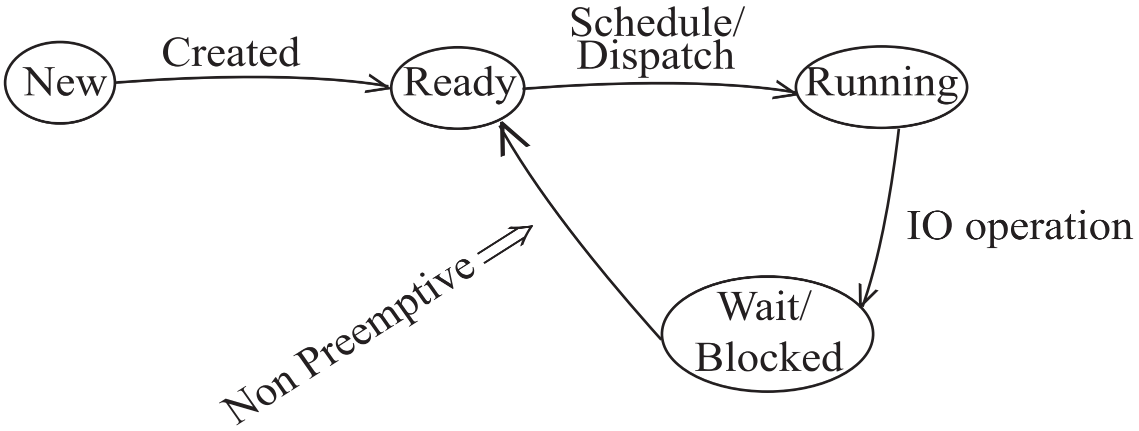
→ The ready state contain multiple number of processes.

→ From multiple number of processes, one process will be selected and it will be scheduled on to the running state.

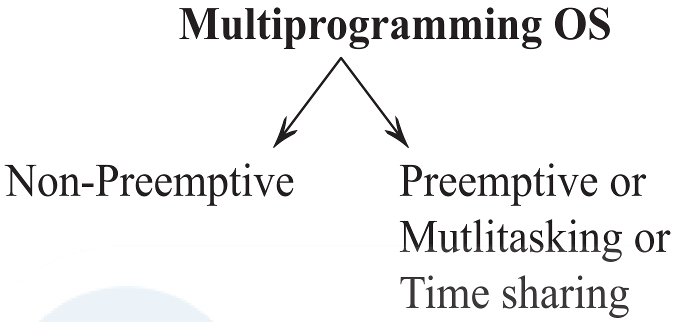


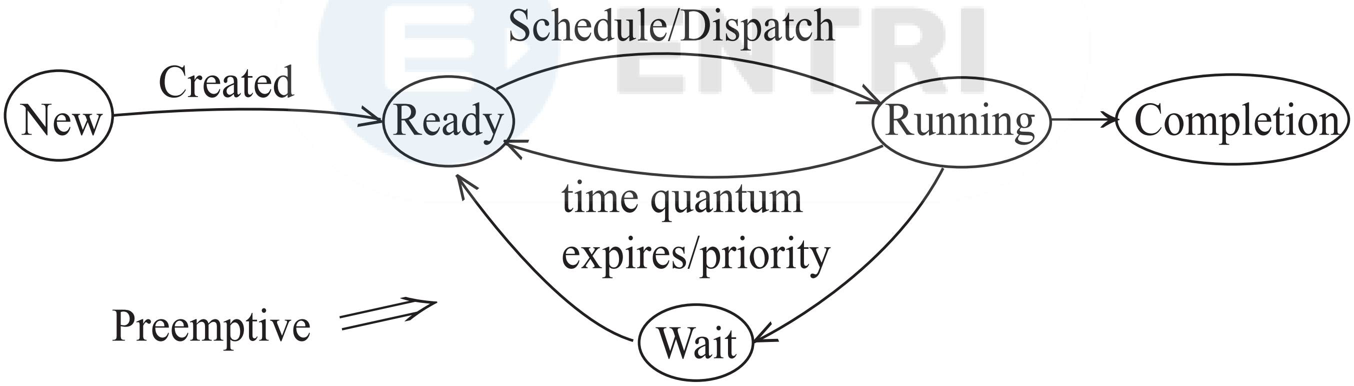
→ In the running state only one process will reside at any point of time.

→ If the running process require any I/O operation then it will move to the wait state.

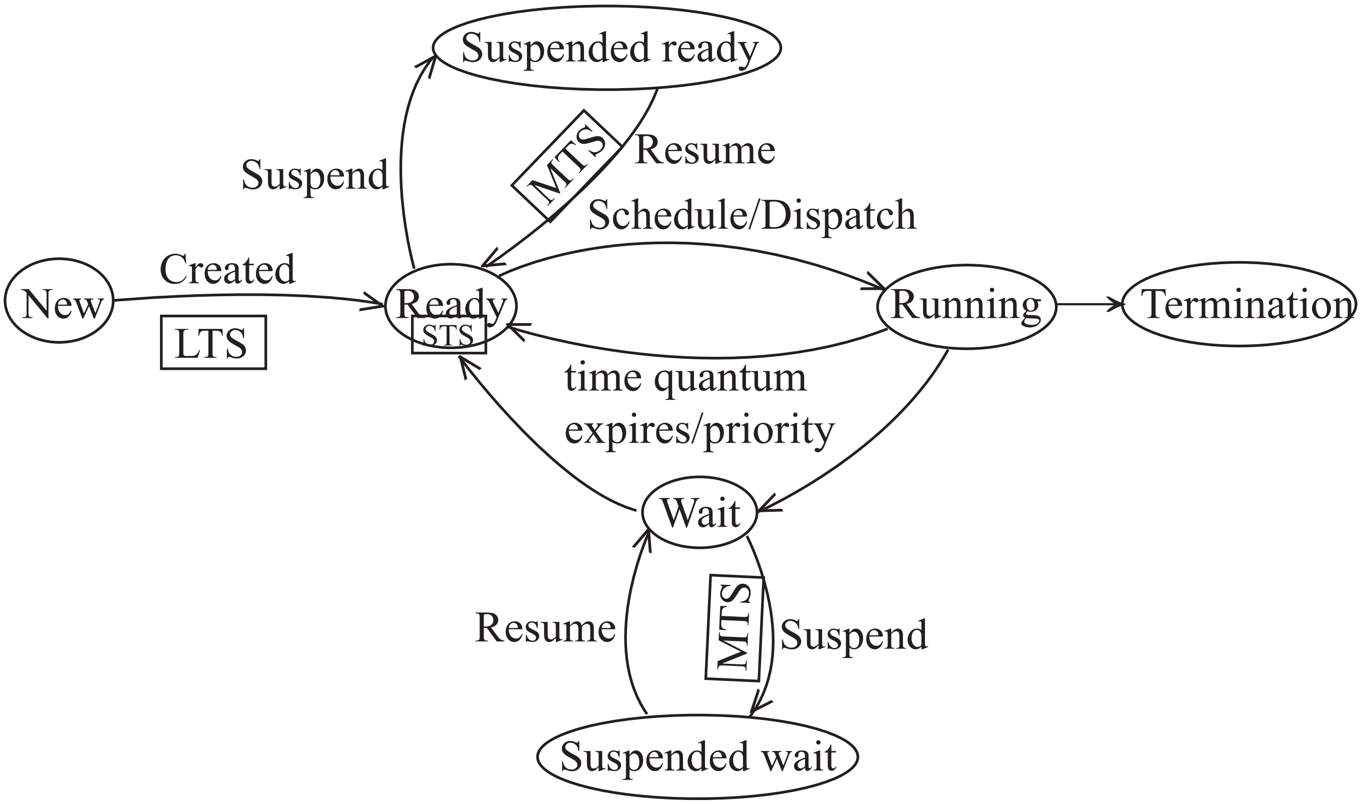


- → In the wait state, the process will perform I/O operation.
- → Once the I/O operation is completed then process will move to the ready state.
- → Wait state contains multiple number of processes (can perform simultaneously operation)
- → In the wait state all the processes perform I/O operation simultaneously.
- → If the process is in ready running or wait state, it means, it is residing in main memory.
- → If the running process complete the execution then process move to the completion.

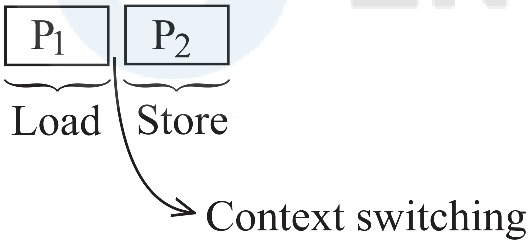




- → If more number of processes are getting created and resources are insufficient to manage all the processes then some of the processes will be suspended and will move to the suspended ready state.
- → Whever the resources are sufficient, then processes will be resume back to the ready state.
- → If the processes is in suspended ready state, it means, it is residing in the backing store or secondary memory.



→ Each and every time when the process is moving from one state to another state the context of the processes will change.



→ Saving the context of one process abd loading the context of another process is called Context switching.

→ Context switching will also take some time. If the context of the process is more than context switching will also increase, which is undesirable.

Scheduler

1. Long term scheduler (Job scheduler)
2. Short term scheduler (CPU scheduler)
3. Mid term scheduler (Medium term scheduler)

→ LTS is responsible for bringing the new process into the system.

→ STS is responsible for selecting one of the process from ready state to schedule onto the running state.

→ Mid term scheduler is responsible for suspending and resuming the process.

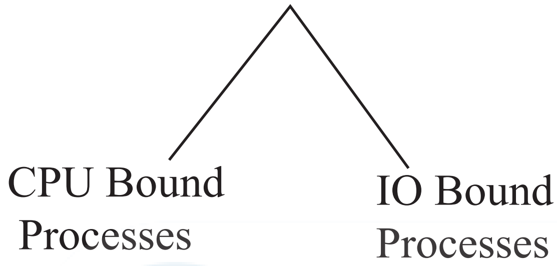
→ The job done by mid term scheduler is known as swapping.

Dispatcher:-

It is responsible for solving and loading the context of the process.

→ The context switching will be done by dispatcher.

→ The processes WRT to their execution time are divided into 2 types.



→ CPU Bound Process: The process which require more CPU time. This type of processes spend maximum time in running state.

IO Bound:- The process which require more IO time. This type of process spend maximum time in wait or block state.

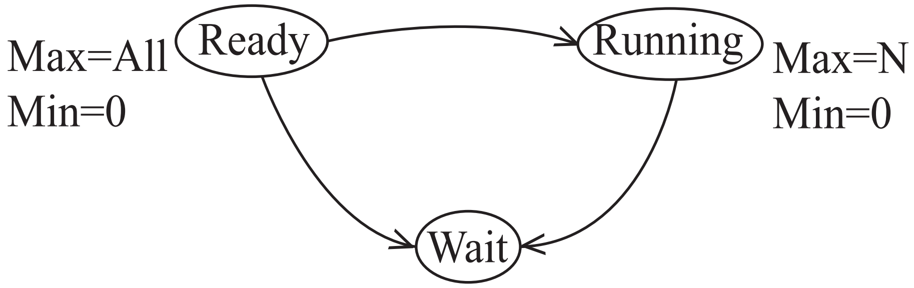
Degree of Multiprogramming

→ The number of jobs or processes present in the memory at any point of time is called degree of mutliprogramming.

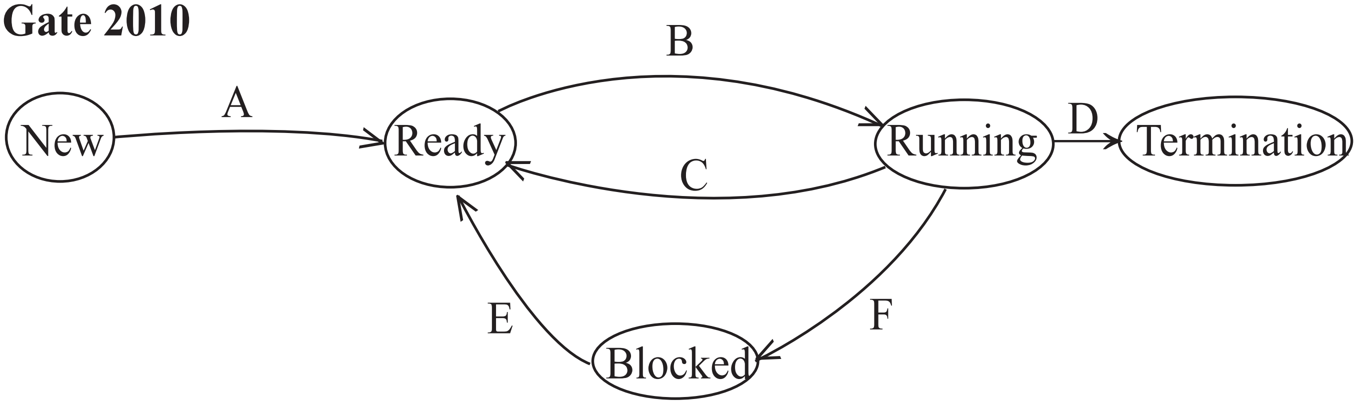
→ LTS should select good combination of CPU bound and IO bound process in order to get good throughput for the system.

→ LTS controls the degree of multiprogramming.

Q. Consider a N-CPU processor system then what is the maximum and minimum number of processes that may present in the ready, running and wait state.



Gate 2010



Consider the following statement WRT to process state diagram.

1. If a process makes a termination D, it would result in another process making transition A immediately.

2. A process P_2 in Blocked state can make transition E while another process P_1 is in Running state.

3. The OS uses Preemptive scheduling.

4. The OS uses Non-Preemptive scheduling.

True $\Rightarrow$ 2, 3

Gate 2008

The P and V operations on counting semaphores, where s is a counting semaphore, are defined as follows

```
P(s) : s = s - 1;
        If (s < 0) then wait;
V(s) : s = s + 1;
        If s ≥ 0 then wakeup a process waiting on s;
```

Assume that P_b and V_b , the wait and signal operations on Binary semaphores are provided.

Two binary semaphores X_b and Y_b are used to implement the semaphores operations P(s) and V(s) as follows:

```
P(s) : $P_b(X_b)$;                V(s) : $P_b(X_b)$;
        s = s - 1;                       s = s + 1;
        If (s < 0)                       If (s ≥ 0)
        {                                $V_b(Y_b)$;
        $V_b(X_b)$;                      $V_b(X_b)$;
        $P_b(Y_b)$;
        }
        Else $V_b(X_b)$;
```

The initial values of X_b and Y_b are respectively

(a) 0 and 0 (b) 0 and 1 (c) 1 and 0 (d) 1 and 1

$s = -2$ $-2 - 1 = \underline{\underline{-3}}$ $\underline{\underline{\text{Blocked}}}$

Q. Two concurrent processes P_1 and P_2 uses four shared resources R_1, R_2, R_3, R_4 as shown below

P_1	P_2
Compute	Compute
Use R_1	Use R_1
Use R_2	Use R_2
Use R_3	Use R_3
Use R_4	Use R_4

Both process are started at the same time and each resource can be accessed by one process at a time.

The following scheduling constraints exist between the access of resources by the processes:

→ P_2 must complete use of R_1 before P_1 gets access to R_1.

→ P_1 ——————— R_2 ——————— P_2 ——————— R_2 .

→ P_2 ——————— R_3 ——————— P_1 ——————— R_3 .

→ P_1 must complete use of R_4 before P_2 gets access to R_4 .

Q. There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed?

(a) 1 (b) 2 (c) 3 (d) 4

Solution:

P_2	P_1
use (R_1)	
signal (s_1)	
	wait (s_1)
	use (R_1)
	use (R_2)
	signal (s_2)
wait (s_2)	
use R_2	
use R_3	
signal (s_1)	
	wait (s_1)
	use R_3
	use R_4
	signal (s_2)
wait (s_2)	
use R_4	

Gate 2013

Three concurrent processes X, Y, Z execute three different code segments that access and update certain shared variables. Process X execute the P OP^n (wait) on semaphores a, b, c process 'Y' execute the P OP^n (wait) on semaphores b, c, d and process 'Z' execute the P OP^n on semaphore c, d, a before entering the respective code segments. After completing the executing of its code segments, each process invokes the V operation (signal) on its three semaphores. All semaphore are binary semaphore initialized to one which one of the following represents a deadlock free order of invoking the P operations by the process?

(a)	X	Y	Z	(b)	X	Y	Z
	P(a)	P(b)	P(c)		P(b)	P(b)=1	P(a)=1
	P(b)	P(c)	P(d)		P(a)	P(c)	P(c)
	P(c)	P(d)	P(a)		P(c)	P(d)	P(d)

(c)	X	Y	Z	(d)	X	Y	Z
	P(b)	P(c)	P(a)		P(a)	P(b)	P(c)
	P(a)	P(b)	P(c)		P(b)	P(c)	P(d)
	P(c)	P(d)	P(d)		P(c)	P(d)	P(a)

Gate 2013

A shared variable x, initiated to zero, is operated on by four concurrent process W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates.
Each processes Y and Z reads x from memory, decrements by two, stores it to memory and then terminates.
Each process before reading x invokes the P operation (wait) on a counting semaphores S and then invokes the V operations (signal) on the semaphores S after store x to memory.
Semaphore S is initialized to two. What is the maximum possible value of x afetr oil processes complete execution?

(a) -2 (b) -1 (c) 1 (d) 2

$S = \cancel{2}\ \ 1$

$\underline{\underline{W}} \qquad x = \cancel{0}\ \ 1$

$S = \cancel{1}\ \ \cancel{0}\ \ 1$

$\underline{\underline{Y}} \qquad x = \cancel{0}$

$-2 \qquad S = \cancel{1}\ \ \cancel{0}\ \ 1$

$Z \qquad x = -\cancel{2}\ \ -4$

$s = \cancel{1}\ \ \cancel{0}\ \ 1$

$W \qquad x = -\cancel{4}\ \ 1$

$s = \cancel{1}\ \ \cancel{0}\ \ 1$

$x = \cancel{1}\ \ 2$

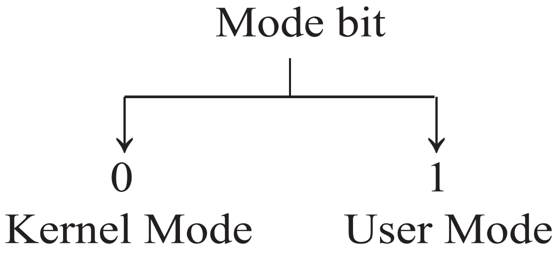
X

$\boxed{x = 2}$

Dual Mode of Operation

User mode or Non-privileged mode
Kernel mode or Privileged mode → Important

Mode bit:- In which mode the present instruction is executed.



→ OS always runs in the Kernel mode.

→ At boot time, the system always start in the Kerneal mode.

Privileged instruction

1. Setting the time of the clock
2. Changing the memory map
3. Context switching
4. Performing IO operation
5. Disable/enable interrupt
6. More secure instruction are kept in the Kernel Mode.

Non-Privileged

→ Read the time of the clock.

fork ( ) system call implementation

```
main ( )
{
  int pid;
   pid = fork (c);
  If (pid < 0)
  {
     Printf ("Fork failed");
   }
  else if (pid =  = 0)
   {
     printf ("Child proces");
    }
```

```
 else
    {
     printf ("Parent process");
    }
}
```

→ Fork is a system call used to create the child process.

→ The fork returns the negative value if the child process creation is unsuccessful.

→ The fork returns value '0' to the newly created child process.

→ Fork return a five positive integer "Process id of the child process" to the parent process.

main ( )
{
  ~~fork ( );~~
  printf ("Hello");
}

printf ("Hello");
Hello → 2
Child → 1

main ( )
  {
    ~~fork ( );~~
    ~~fork ( );~~
    printf ("Hello");
  }

~~fork ( );~~
printf ("Hello")

printf ("Hello");

printf("Hello");

Hello → 4
Child → 3

→ If the program contain 'n' fork call then it will create 2^n - 1 child process and 2^n parent process.

Relative address:

→ Same for both child and parent process

Absolute address:

→ It is different for child and parent process.

→ When programmatically trying to print the address of the process will always print the relative address.

Gate 2005

```
        If (fork ( ) = = 0)
          {
             a = a + s;
             printf("%d, %d", a, &a);
          }
        else
          {
             a = a - s;
             printf ("%d, %d, a, &a);
          }
```

Let u, v be the values printed by the parent process and x, y be the values printed by the child process. Which one is true?

(a) u = x + 10 and v = y (b) u = x + 10 and v ≠ y

(c) u + 10 = x and v = y (d) u + 10 = x and v ≠ y

Gate 2019

Q.

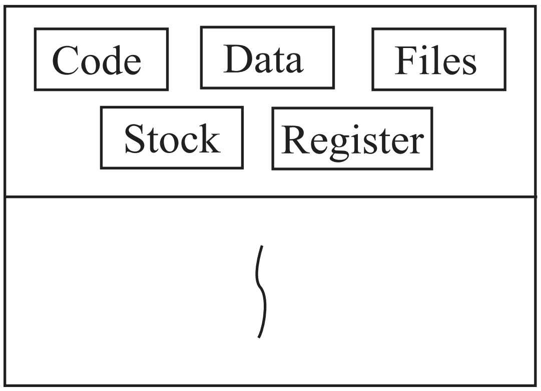
```
int main ( )
  {
     int i;
     for (i = 0; i < 10; i++)
     If (i%2 = = 0)
        fork ( );
  }
```

the total number of child process created is _______?

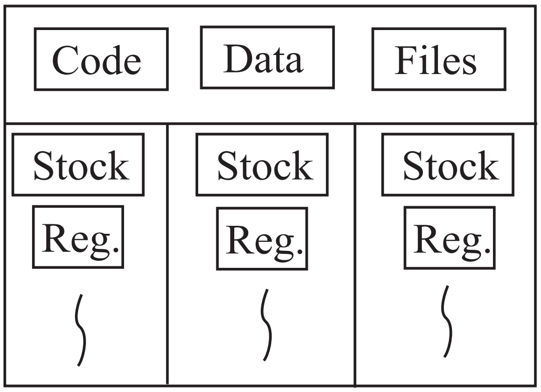
Solution: $n = 5$

child process = $2^5 - 1 = 31$

Thread



Simple Threaded System

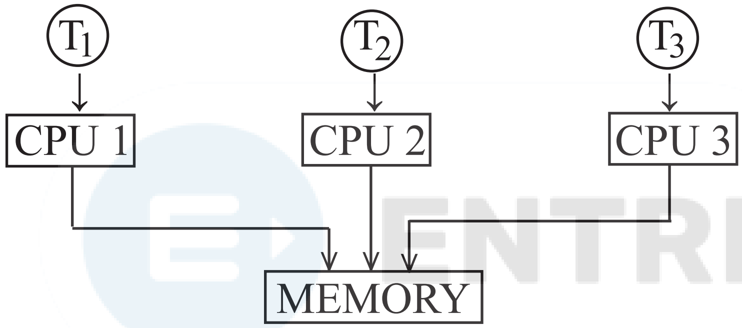


Multi Threaded System

Thread is a light weighted process.

Advantage

1. **The thread will improve the responsiveness.**
   If one thread is completed the execution then the o/p will be responded.
2. **Faster Context Switching**
   $\boxed{CS_T < CS_P}$
3. **Resource Sharing**
   Resource like data, code, files and memory will be shared among all the threads within the process but every thread will have its own stock and register.
4. **Effective utilisation of multiprocessor system→**
   If process is divided into multiple threads then that different threads can be scheduled onto different CPUs then the processor execution will be faster.



5. **Enhanced throughput of the system**
   → When the process is divided into multiple threads as one job then the number of jobs completed per unit of time will increase. Hence throughput of the system will be increased.

6. **Economical**
   Implementing of the threads doesn't require any cost. There are various API's which supports implementation of threads.

User level thread	Kernel level thread
1. Implemented by the user	1. Implemented by OS
2. Not recognised by the OS, OS views user level thread as a process only	2. Recognised by the OS

3. If one user level thread is performing the blocking system call then entire process will be blocked.	3. If one Kernel level thread is performing blocking system call, another thread will continue the execution
4. Dependent	4. Independent
5. Less context	5. More context
6. No hardware support is required.	6. Hardware support is requiired.

NOTE:-

→ User level thread scheduling is faster than Kernel level scheduling. Both the threads require memory management.

→ User level threads are scheduled by thread library user level)

→ Kerneal level threads are schedule by OS (Kernel level).

Dead lock

Event
↑
Two or more (processes) are waiting on some event to happen which never happens then those processes are said to be involve in a deadlock.

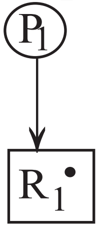
$\textcircled{P_1}$ $\textcircled{P_2}$ → Processes

$\boxed{R_1}$ $\boxed{R_2}$ → Resources

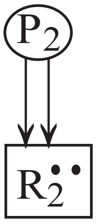
$\boxed{R_1 \bullet}$ → one instance of resource 'R_1'

$\boxed{R_2 \bullet\bullet}$ → two instances of resource 'R_2'

Requesting Edge

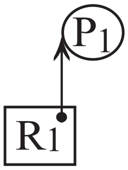


Processes 'P_1' is requesting for one instance of resource 'R_1'

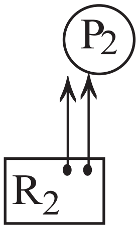


Process 'P_2' is requesting for two instances of resource 'R_2'.

Allocation Edge



One instance of resource 'R_1' is allocated to process 'P_1'



Two instance of resource 'R_2' is allocated to process 'P_2'

Resource-request and release life cycle

1. The process will request for the resource.
2. The OS validate the request for the process.
3. OS check for availability of the resources.
4. If the resource is freely available then it will be allocated to process otherwise process has to wait.

5. If all the resource required by the process are allocated then the process will go for execution.

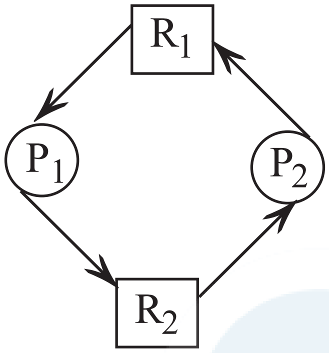
6. If execution of the process is completed then it will release all the resources.

Resource Allocation Graph (RAG)

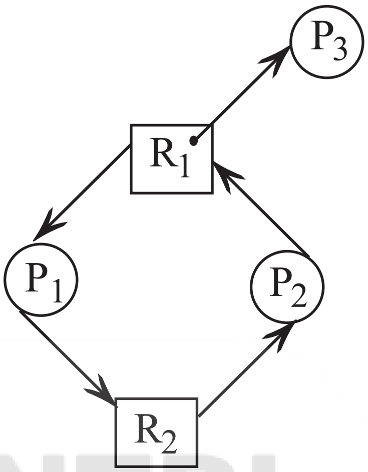
RAG = (V, E)

→ allocating edge or requesting edge

↓ processes and resources



Dead lock



Not in Dead Lock

Q. Consider a system which has n processes and 6 tape drives. If each process requires 2 tape drives to complete the execution then what is the max. value of n which ensure deadlock free operation

(a) 2 (b) 3 (c) 4 (d) 5

Solution:

P_1	P_2	P_3	P_4	P_5
1	1	1	1	1
1				

$\underline{\underline{n = 5}}$

Q. Consider a system with 3 processes where each process require 2 unit of resource ‘R’. What is the maximum number of resource require to ensure deadlock free operation?

Solution:

P_1	P_2	P_3
1	1	1

1 $\underline{\underline{4}}$

$$\boxed{\text{Sum of total resources} < \text{No. of processes} + \text{Min. no. of resources to avoid deadlock}}$$

Q. Consider a system with 3 process P_1, P_2, P_3 the peak demand of each process is 5, 9, 13 respectively. What is the minimum no. of resources required to ensure deadlock free operation?

Solution:

$P_1 \rightarrow 5 \rightarrow 4$

$P_2 \rightarrow 9 \rightarrow 8$

$P_3 \rightarrow 13 \rightarrow \underline{12}$

$24 + 1 = 25$

$5 + 9 + 13 < 3 + x$ $\boxed{x = 25}$

→ Deadlock characteristic

→ Deadlock avoidence

→ Deadlock recovery

→ Deadlock prevention

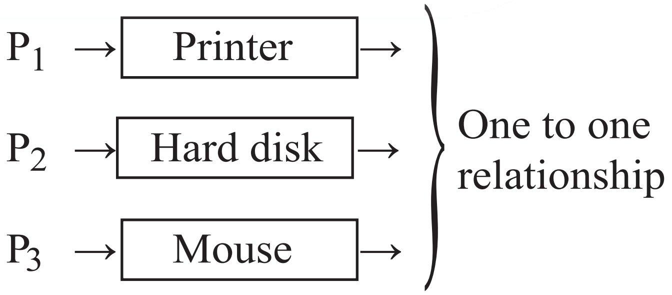
→ Deadlock detection

Deadlock Characteristic

1. **Mutual Exclusion**

→ the resources has to be allocated to only one process

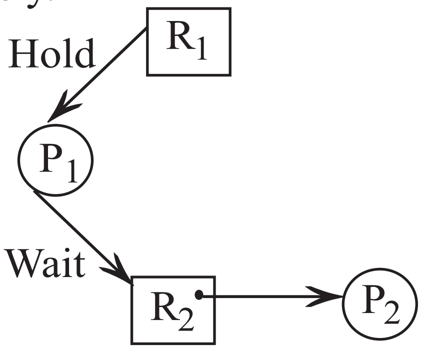
→ there should be one-to-one relationship between the resources and processes.



2. **Hold and Wait**

→ The process is holding some resources and waiting on some other resources simultaneously.

→



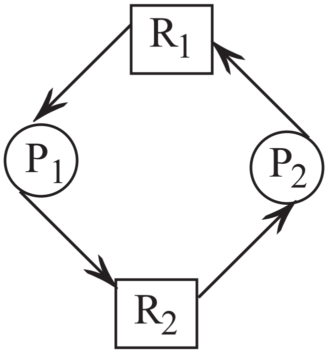
3. **No Preemption:-**

→ The resource has to be volunterily (by it own wish) release by the process after completion of the execution.

→ It is not allowed to preempt the resources forcefully from the process.

4. **Circular Wait**

→ The processes are circularly waiting on each other for the resources



$$\begin{tikzcd} R_1 \arrow[r] & P_1 \arrow[r] & R_2 \arrow[r] & P_2 \arrow[lll, bend left] \end{tikzcd}$$

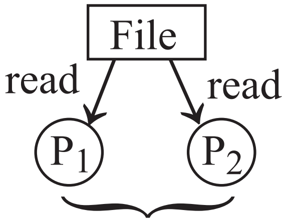
→ If all the four conditions are occuring simultaneously in the system then definitely there exist a deadlock.

→ All the above four conditions are not purely independent because circular wait includes Hold and Wait.

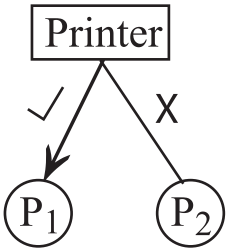
Deadlock Prevention

1. **Mutual Exclusion**

→ It is not possible to dis-satisfy mutual exclusion always because of sharable or non-sharable resources.



Sharable

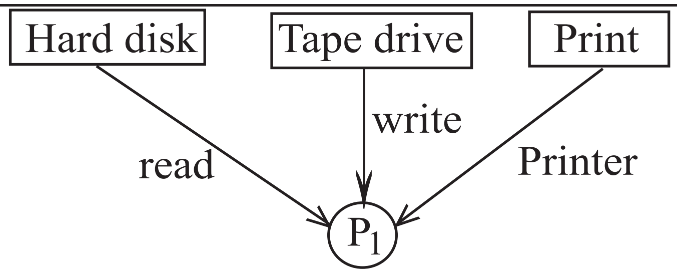


Non-Sharable

2. **Hold and Wait**

→ Allocate all the resources required by the process before the start of the execution.

→



→ Low device utilization

→ The process should release all the existing resources before making the new request.

5. No Preemption

The process 'P_1' is requesting for resource 'R_1'

If the resource 'R_1' is freely available then it will be allocated to process 'P_1'

If the resource 'R_1' is not freely available and it is allocated to process 'P_2'

↓ If the process 'P_2' is in the execution then 'P_1' has to wait

↓ If the process 'P_2' is not in the execution and waiting for another resources

↓

The resource 'R_1' is preempted from the process 'P_2' and allocated to 'P_1'.

4. Circular Wait

→ Every resource will be assigned with numerical number.

→ The processes can request for the resources only in the increasing number of remuneration

$P_1 \rightarrow R_5$	$P_1 \rightarrow R_4$	$P_1 \rightarrow R_8$
✓	X	✓

increasing order →

$P_1 \rightarrow R_5$	$P_1 \rightarrow R_6$	$P_1 \rightarrow R_4$
✓	X	✓

decreasing order →

Deadlock Avoidance

Banker's also rithm

total available

A	B	C
10	5	7

	MAX NEED			Current Allocation			Current Available		
	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

Safe Sequence: P_1, P_3, P_4, P_0, P_2

P_1, P_3, P_4, P_2, P_0

$$\begin{array}{cccc} & 3 & 3 & 2 \\ & 0 & 1 & 0 \\ \hline \text{X} & 3 & 4 & 2 \\ \hline \end{array}$$

← P_0 denied the access

$$\begin{array}{ccc} 3 & 3 & 2 \\ 2 & 0 & 0 \\ \hline 5 & 3 & 2 \\ \hline \end{array}$$

→ P_1 complete its execution and release all the resources.

$$\begin{array}{ccc} 5 & 3 & 2 \\ 3 & 0 & 2 \\ \hline 8 & 3 & 4 \\ \hline \end{array}$$

← P_2 denied the access

$$\begin{array}{ccc} 5 & 3 & 2 \\ 2 & 1 & 1 \\ \hline 7 & 4 & 3 \\ \hline \end{array}$$

← P_3 Complete its execution and release all the resources.

$$\begin{array}{cccl} 7 & 4 & 3 & \\ 0 & 0 & 2 & P_4 \checkmark \\ \hline 7 & 4 & 5 & \\ 0 & 1 & 0 & P_0 \checkmark \\ \hline 7 & 5 & 5 & \\ 3 & 0 & 2 & P_2 \checkmark \\ \hline 10 & 5 & 7 & \\ \hline \end{array}$$

$$\begin{array}{cccl} 7 & 4 & 3 & \\ 0 & 0 & 2 & \\ \hline 7 & 4 & 5 & \\ 3 & 0 & 2 & P_2 \checkmark \\ \hline 10 & 4 & 7 & \\ 0 & 1 & 0 & P_0 \checkmark \\ \hline 10 & 5 & 7 & \\ \hline \end{array}$$

→ If we can satisfy the remaining need of all the processes with currently available sources then system is said to be in safe state otherwise the system in unsafe state.

→ If the system is in unsafe state then it is possible for deadlock.

→ The order in which we satisfy the remaining need of all the processes is called safe sequence.

→ The safe sequence cannot be unique we can have multiple safe sequence.

→ The unsafe state purely depends on the behaviour of the process.

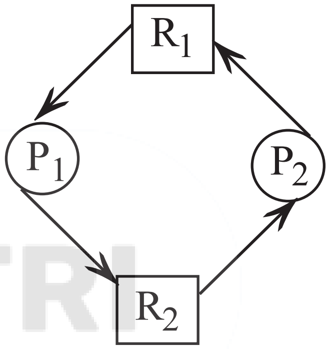
Deadlock Detection

→ If all the resources are single instance type then cycle in resource allocation graph is necessary and sufficient condition for occuring a deadlock.

→ **Necessary :-** Deadlock may be possible or may not be possible.

Sufficient :- There should be deadlock

$$\begin{tikzcd} R_1 \arrow[r] & P_1 \arrow[r] & R_2 \arrow[r] & P_2 \arrow[lll, bend left] \end{tikzcd}$$



→ If all the resource are not of single instance type then cycle in the RAG is a necessary but not sufficient condition for occuring a deadlock.

→ If the resource are of multiple instance type then the banker's algorithm will be used to identify the remaining need of all the processes are satisfied or not.

→ If the remaining need of all the processes are satisfied with currently available resources then deadlock does not exist otherwise deadlock exist in the system.

Deadlock Recovery

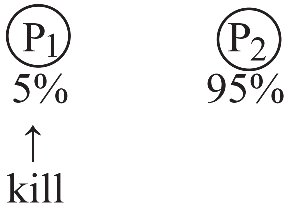
1. Killing the process

→ Kill all the processes which involve in the deadlock

→ Kill one by one

(i) low priority process

(ii) % of process completion



(iii) Based on Number of resources the process is holding.

$\textcircled{P_1}$ $\textcircled{P_2}$

2R 20R

$\uparrow$

$\underline{\underline{\text{Kill}}}$

2. **Resource Preemption**

→ The resource will be preempted from the processes which are involved in the deadlock.

3. **Ostisch Algorithm**

→ Ignore the deadlock.

Gate 2007

```
      P1                                  P2
while (true)                        while (true)
{                                   {
   wants 1 = true;                     wants 2 = true;
   while (wants 2 = = true);           while (wants 1 = = true);
                 CS                                  CS
   wants 1 = false;                     wants 2 = false;
}                                   }
```

Q. wants 1 and wants 2 are shared variables which are initialized to false which one is correct?

(a) It doesn't ensure mutual exclusion

(b) It doesn't ensure bounded waiting

(c) It requires that processes enter into the critical section in strict alteration

(d) It doesn't prevent deadlock but ensure mutual exclusion.

Ans. (d)

Gate 2016

Semaphore n = 0; semaphore S = 1;

```
Void producer ( )
{
 while (true)
 {
   produce ( );
   semwait (s);
   add to buffer ( );
   semsignal (s);
   semsignal (n);
  }
}
```

```
Void consumer ( )
{
 while (true)
 {
   semwait (s);
   semwait (n);
   remove from buffer ( );
   semsignal (s);
   consume ( );
  }
}
```

which one is true?

(a) The producer will be able to add an item to the buffer but the consumer can never consume it.

(b) The consumer will remove no more than one item from the buffer.

(c) Deadlock occurs if the consumer succeeds in a acquiring semaphore when the buffer is empty.

(d) The starting value for the semaphore must be 1 and not 0 for deadlock free operation.

Ans. (c)

Gate 2017

A system shares 9 tape drives

Process	Current Allocation	Max requirement
P_1	3	7
P_2	1	6
P_3	3	5

Which of the following describes the current state of the system?

(a) Safe, deadlock (b) Safe, not deadlock

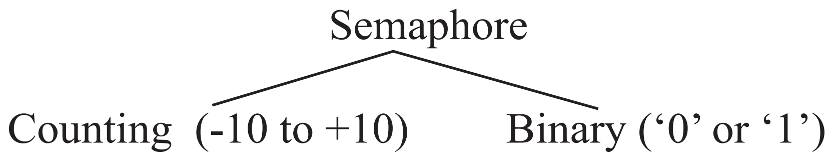
(c) Not safe, deadlock (d) Not safe, not deadlock

Ans. (b)

Semaphore

→ Semaphore is an integer variable which is used by various processes in mutually exclusive manner to achieve synchronisation.

→ Improper use of semaphore will also give improper result.



Semaphore

Counting (-10 to +10) Binary (‘0’ or ‘1’)

Two types of operation

1. Down ( ) or wait ( ) or P ( )
2. Up ( ) or release ( ) or signal ( ) or V ( )

1. Counting semaphore

```
Down (semaphore s)
 {
  s.value = s.value - 1;
   If (s.value < 0)
     {
      Block the process and place its PCB in the suspended list;
     }
 }
up (semaphore s)
 {
  s.value = s.value + 1;
    If (s.value ≥ 0)
     {
      select a process from suspended list and wakeup ( );
     }
 }
```

→ After performing the down operation if the process is getting blocked then it is called unsuccessful down operation.

→ After performing the down operation if the process is not getting blocked them it is called successful down operation.

→ If it is successful down operation then only the process will be continued in the execution.

→ The down operation of the counting semaphore is successful only when semaphore value greater than 0 or equals to 1

$\boxed{s \geq 1}$

→ If $\boxed{s = +6}$, then we can perform successful down operation.

→ If $\boxed{s = -6}$, it represent there are 6 suspended process.

→ up operation is always successful.

→ There is no unsuccessful up operation.

Binary Semaphore

```
Down (semaphore s)
  {
   If (s.value = = 1)
    s.value = 0
  else
     {
       Block the process and place its PCB in the suspended list ( );
     }
}
  up (semaphore s)
   {
    If (suspended list ( ) is empty)
        s.value = 1;
    else
        {
          select a process from suspended list and wake up ( );
        }
     }
```

→ The down operation of Binary semaphore is successful only if the semaphore value is 1.

→ There is no unsuccessful up operation.

→ Up operation is always successful.

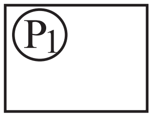
Gate 2003

```
P1 to P9                        P10
Repeat                          Repeat
P (mutex);                      V (mutex);
    cs                              cs
V (mutex);                      V (mutex);
foreover                        foreover
```

The initial value of Binary semaphore mutex is ‘1’. What is the maximum number of processes hat may present inside critical section at any point of time.

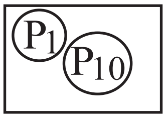
Solution:

mutex = $\cancel{1}$ 0



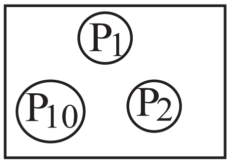
After mutex value is 0, then execute process ‘P_{10}’ code then v (mutex) in process ‘P_{10}’ increment the semaphore value 1.

mutex = $\cancel{1}$ $\cancel{0}$ 1



→ At this time mutex = 1, so P_2 can easily go inside the critical section & mutex becomes 0

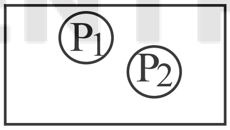
mutex = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ 0



→ At this time none of the process can enter inside the critical section because mutex = 0

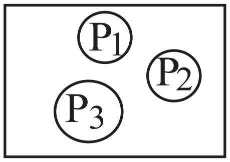
→ To enter any process inside the $\boxed{\text{cs}}$, P_{10} release the cs and mutex = 1

mutex = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ $\cancel{0}$ 1



→ At this time P_3 can easily go inside the $\boxed{\text{cs}}$ & mutex becomes 0.

mutex = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ 0



Again execute process ‘P_{10}’ code and mutex = 1 and so on all the process can enter inside the cs

∴ Maximum number of processes = 10

Q. $\underline{P_1 \text{ to } P_9}$

Repeat
p (mutex)
$\boxed{\text{cs}}$
v (mutex);
forever

$\underline{P_{10}}$

Repeat
v (mutex);
$\boxed{\text{cs}}$
p (mutex);
forever

We interchange process 'P_{10}' code in this manner, then what is the max. no. of process may present inside the cs at any point of time?

(a) 2 (b) 3 (c) 4 (d) 10

Gate 2003

Q. Suppose we want to synchronise the 2 concurrent processes P and Q using Binary semaphore S and T.

```
Process 'P'                    Process 'Q'
while (1)                      while (1)
{                              {
  w :                            y:
  print'0';                       print '1'
  print '0';                      print '1'
  X;                              Z;
}                              }
```

Which of the following will always lead to an output string with 00110011 00.......

(a) W = P(T) X = V(T) Y = P(S) Z = V(S) S = T = 1

(b) W = P(T) X = V(T) Y = P(S) Z = V(S) S = 1, T = 0

(c) W = P(T) X = V(S) Y = P(S) Z = V(T) S = 1 = T

(d) W = P(T) X = V(S) Y = P(S) Z = V(T) T = 1, S = 0

Option - A

S = 1 = T, so T can be started

then 110011----------- may be printed, so it is wrong.

Option - C

So c is also wrong

Option - B

S = 1, then process 'Q' will be executed, so initially 11 will be executed, so it is wrong.

Option - D

Correct

Gate 2004

Consider 2 processes P_1 & P_2 accessing the shared variable x and y protected by 2 binary semaphore Sx and Sy respectively and both are initialized to ‘1’. P and V denote the usual semaphore operator where P decrement the semaphore value and V increment the semaphor value.

```
   P1                  P2
while (1)           while (1)
{                   {
  L1;                 L3;
  L2;                 L4;
  X = X + 1;          Y = Y + 1;
  Y = Y - 1;          X = X - 1;
  V (Sx);             V (Sy);
  V (Sy);             V (Sx);
}                   }
```

In order to avoid deadlock the correct operators at L1, L2, L3, L4 respectively.

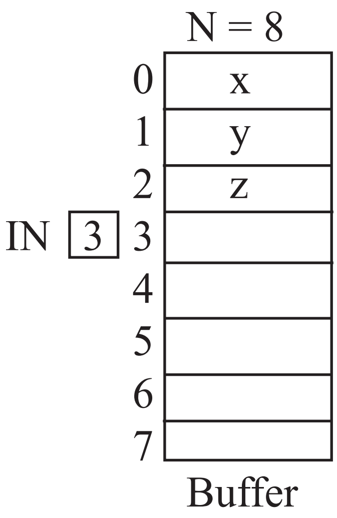
(a) P(Sy), P(Sx), P(Sx), P(Sy) (b) P(Sx), P(Sy), P(Sy), P(Sx)

(c) P(Sx), P(Sx), P(Sy), P(Sy) (d) P(Sx), P(Sy), P(Sx), P(Sy)

(a)	deadlock	$Sy = \cancel{1}\ 0$	$Sx = \cancel{1}\ 0$
(b)	deadlock	$Sx = \cancel{1}\ 0$	$Sy = \cancel{1}\ 0$
(c)	deadlock	$Sx = \cancel{1}\ 0$	$Sy = \cancel{1}\ 0$
(d)	not in deadlock		

Classical Problem of IPC

Producer and Consumer Problem



out = 0

item c = ‘x’

item p = ‘k’

mutex = 1

empty = 5 4 5

full = $\frac{3}{8}$ $\frac{4}{8}$ $\frac{3}{8}$

```
semaphore mutex = 1;        semaphore empty = N;        semaphore full = 0;

    void producer (void)
    {
     int item p;
     while (true)
     {
      produce_item (item P)
      down (empty);
      buffer [IN] = item P;
          IN = (IN + 1) mod N;
          up (mutex);
          up (full);
     }
    }

    void consumer (void)
    {
     int item c;
     while (true)
     {
      down (full);
      down (mutex);
      Item c = buffer [out];
          out = (out + 1) mod N;
          up (mutex);
          up (empty);
          process_item (item c);
       }
    }
```

→ Mutex is a binary semaphore variable used by the producer and consumer to access the buffer in a mutually exclusive manner.

→ Empty is a counting semaphore variable which represents the number of empty slots in the buffer at any point of time.

→ Full is a counting semaphore variable, it represents number of fill slots in the buffer at any point of time.

Q. Which down (empty) and down (mutex) are interchange in producer port then what are the effects?

(a) Solution works fine and there is no problem at all.

(b) It is possible for both producer and consumer to use the buffer at the same time.

(c) Some times product producer by the producer will be lost.

(d) It is possible for deadlock.

Ans. (d)

mutex = 1

when buffer is full

$E = \cancel{0} \quad -1$ | Both producer and consumer will be

$F = \cancel{8} \quad 7$ | suspended

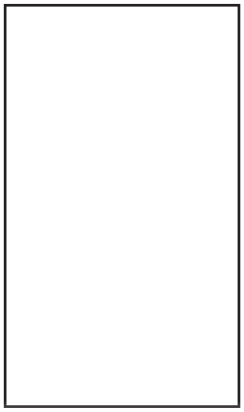
NOTE:

→ When down operation are interchanged then there is always possibility for deadlock.

→ When up operation are interchanged then there is no problem at all. Solution works fine

Reader and writer problem

```
semaphore mutex = 1
semaphore db = 1
        int  rc = 0
void reader (void)
        {
          while (true)
           {
             down (mutex);
             rc = rc + 1;
        If (rc = = 1) down (db);
             up (mutex);
                    DB
             down (mutex);
             rc = r  - 1;
        If (rc = = 0)
             up (db);
             up (mutex);
           }
        }
```



database

```
void writer (void)
{
    while (true)
   {
    down (db);
        DB
   }
}
```

R - W ✗

W - R ✗   rc = 0

W - W ✗   mutex = 1

R - R ✓   db = 1

→ Mutex is a binary semaphore used by the reader in a mutually exclusive manner.

→ 'db' (database) is also a binary semaphore variable used by the reader and writer in a mutually exclusive manner.

→ ‘rc’ (reader count) is an integer variable represents the number of readers present in the database at any point of time.

Gate 2015

Q. The following two functions P_1 & P_2 that share variable ‘B’ with on initial value of 2 execute concurrently

```
        P1 ( )                       P2 ( )
        {                            {
(1)     C = B - 1;           (3)     D = 2 × B;
(2)     B = 2 × C;           (4)     B = D - 1;
        }                            }
```

The number of distinct values that ‘B’ can possibly take after the execution __________?

(1)	C = 1	(1)	C = 1	(1)	C = 1	(1)	D = 4	(3)	D = 4
(2)	B = 2	(3)	D = 4	(3)	D = 4	(2)	B = 3	(1)	C = 2
(3)	D = 4	(2)	B = 2	(4)	B = 3	(3)	C = 2	(2)	B = 4
(4)	B =$\textcircled{3}$	(4)	B = 3	(2)	B =$\textcircled{2}$	(4)	B =$\textcircled{4}$	(4)	B = 3

$$B = \underbrace{2, 3, 4}_{\text{3 values}}$$

Gate 2006

Q. Let P[0] - - - P[4] be the processes and m[0] - - - m[4] be mutexes be binary semaphore initialized to ‘1’.

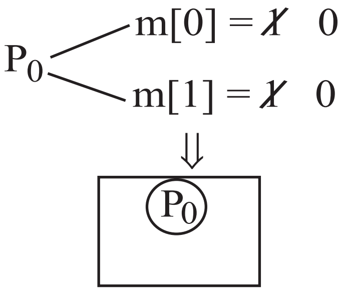
```
wait (m[i]);
wait (m[i + 1] mod N);
    [cs]
signal (m[i]);
signal (m [i + 1] mod N);
```

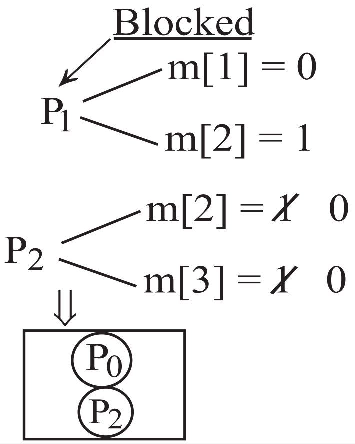
(1) mutual exclusion is satisfied.

(2) —————— not ——————

(3) Deadlock is possible.

Solution:





Ans. Statement 2, 3 are true.

NOTE:

There are two possibility for deadlock

(1) When more than 1 process enter into critical section then deadlock is possible.

(2) If one process restricted another process to enter into critical section or vice-versa then deadlock is possible.

Gate 2016

Q. Consider a non-negative counting semaphore S. The operation P(S) ↓ S, V(S) ↑S, during an execution 20 P(S) operation and 12 V(S) operation are issued in some order. The largest initial value of S for which atleast one P(S) operation will remain block is ________?

Solution

I = Initial value of semaphore

P = Number of wait operations

V = Number of signal operations

then resultant value of semaphore $= \boxed{I - P + V}$

$-1 = I - P + V \quad \Rightarrow \quad -1 = I - 20 + 12 \quad \Rightarrow \quad -1 + 8 = I$

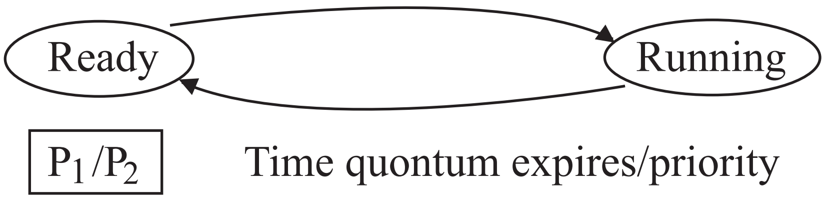
$-1 = I - 8 \quad \Rightarrow \quad I = 7$

SYNCHRONIZATION

→ The process wrt to synchronization are two types.
(1) Co-operative process (2) Independent process

Coperative Process : The execution of one process effects or affected by other process. Then these processes are said to be co-operative process, otherwise they are independent process.

NOTE : Interrupt or preemption can occur at any point of time or at any where

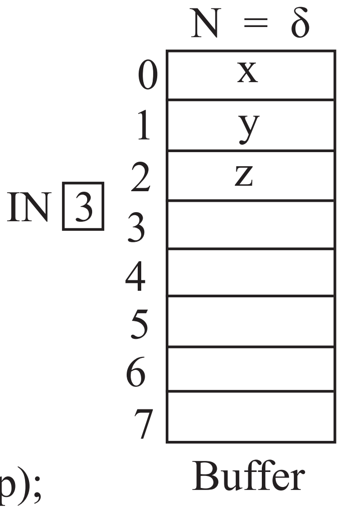


(1) Problem arises not having proper synchronization between the processes.
(2) Conditions to be followed to achieve the synchronization.
(3) Solutions (wrong solutions or right solutions).

Producer and Consumer

Producer

```
int couut = 0;
void produce (void)
ξ
        itemp,
while (true)
ξ
Producer, itemc (itemp);
while (cout == N);
Buffer [IN] = itemp;
```



couut = ~~3~~ ~~2~~ 4

Item P = ‘x’

Item C = ‘n’

```
Void Consumer (void)
ξ
int itemc;
while (true)
ξ
While (Count == 0);
Item C = Buffer [∞t]
```

```
IN  =  (IN + 1) mod N;
Count  =  Count t1;
-ξ
-ξ
I       Load Rp, M[count]
II      Iner Rp
III     Store M[count], Rp
        P - I          Rp  = ~~3~~4
            II
        C - I          Rc  = ~~3~~ 2
            II
            III
        P - III
```

```
out  =  (out tl)mod N;
Count  =  Count - l;
Process - item (itemC)
ξ
-ξ
I        Load Rc, M[count]
II      DEC Rc
III     Store m[count] Rc
```

→ 'IN' is a variable used by the producer to identify the next empty slot in the buffer.

→ 'OUT' is a variable used by the consumer from where it has to be consumed the item.

→ 'Count' is a variable used by the producer and consumer to identify the number of fill slots in the buffer at any point of time.

→ Shared resource are (1) Buffer (2) Count

NOTE : If the buffer is full then producer is not allowed to produce the item.

→ If the buffer is empty than consumer is not allowed to consume the item.

Universal Assumption

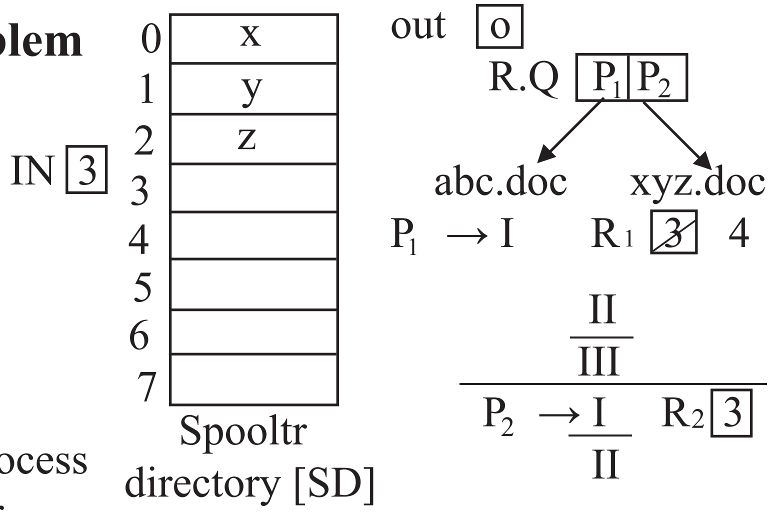
→ While execitomg an instruction, if the interrupt comes, the interrupt will be serviced only after completion of the current micro instruction.

Due to this, there are 3 problems.

(1) In consistency (2) Loss of daa (3) Deadlock

The producer and consumer are not properly synchronized by sharing a common variable count. Hence it is leading to inconsistency problem.

Printer and spooltr problem



Respective process resistor

Enter File

(1) Load R_i , M[IN]

(2) Store SD[R_i], "file-name";

(3) INCR R_i ;

(4) Store M[IN], R_i ;

→ 'IN' is a variable used by all the process to identify the next empty slot in the spooltr directory.

→ 'OUT' is used by the printer to identify from where it has to print the document.

→ Shared Resources
'IN' and 'OUT' variable

→ The processes are not properly synchronized while sharing the common variable 'IN', hence it is leading to loss of data problem.

Deadlock : - If the proper synchronization is NOT there between the processes then it is also possible for deadlock.

(2) Conditions to be followed to achieve synchronization.

Definitions:

(1) Critical section

→ The portion of program text where the shared variables or shaped resources are placed.

(2) Remainder section or Non-critical section :

→ The portion of present text where the independent code of the process will be placed.

(3) Race condition :

→ The final value of any variable depends on the executiion sequence of the processes.

Conditions :

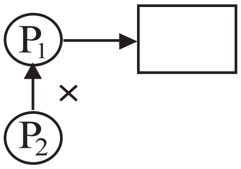
(1) Mutual exclusion :-

→ No 2 process may be simultaneously present inside the critical section at any point of time.

→ Only one process should be present inside the critical section at any point of time.

(2) Progress :-

→ No process running outside the critical section should block the other interested process from entering into the critical section when critical section is free.



(3) Bounded wait : -

→ No process should have to wait forever tp enter into the critical section. There should be a bound in getting chance to enter into critical section.

→ If bounded waitings is not satisfied then it is possible.

Solutions :

(1) Software type of solutions :

→ (a) Lock variables

→ (b) Strict alternation and Decker's algorithm

→ (c) Peterson's solutions

(2) H/w type of solution

→ Test and set lock instruction set (TSL)

(3) OS type of solutions

→ Counting semophore

→ Binary semophore

(4) Programming language type of solution

→ Monitors

Lock variables

Entry section

(1) Load R_i , M[lock]

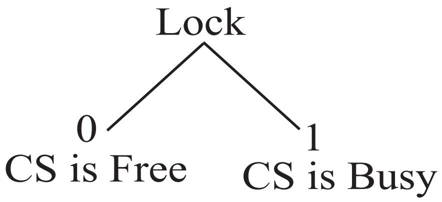
(2) CMP R_i,

(3) Jn2 to step (1)

(4) Store m[lock], ≠ ①

(5) $\boxed{\text{CS}}$

(6) Store m[lock], ≠ 0



R.Q $\boxed{P_1 \mid P_2}$

Lock = 0

$P_1 \rightarrow$ I $\qquad$ R_1 $\boxed{0}$

II

III

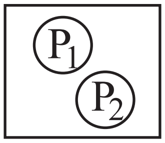
$P_2 \rightarrow$ I $\qquad$ Lock = $\cancel{0}$ 1

II $\qquad$ R_2 $\boxed{0}$

III

IV

V

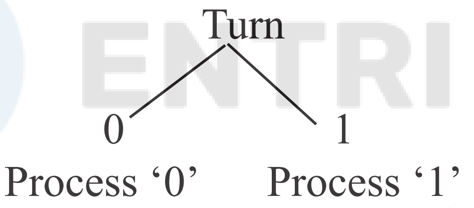


$P_1 \rightarrow$ IV $\qquad$ Lock = $\cancel{0}$ 1

→ We have prove that both the process P_1 & P_2 are simultaneously present inside the critical section at same point of time.

Hence, mutual exclusion is not satisfied and the solution is bound to be incorrect.

(2) Strict alternation and Decker's algorithm (Process takes turn to enter into critical section)



Process 'P_0'

While (true)

ξ

non-(sc);

While (turn 1 = 0);

$\boxed{cs}$

turn = 1,

-ξ

turn = 0

Process 'P_1'

While (true)

ξ

non-(sc);

while (turn 1 = 1)

$\boxed{cs}$

turn = 0;

-ξ

For mutual exclusion execute code until process enter into $\boxed{cs}$.

turn = $\cancel{0}$ 1

exit

$\boxed{P_0} \Rightarrow \boxed{}$

turn = $\cancel{1}$ 0

exit

$\boxed{P_1} \Rightarrow \boxed{}$

Progress : - Process 'P$_1$' enter and exit
Process 'P$_2$' enter and exit
Now if second process re-entry is successfull.
Then progress is satisfied.

Det : - Which process will go next in $\boxed{cs}$ is decided by only those process who want to go into $\boxed{cs}$

→ In this decision, the process in remainder section and the process which is not interested to go into $\boxed{cs}$ should not participation.

Bounded wait

→ If number of process countable, finite, bounded then between is satified.

Gate 2010

$$\begin{array}{ll} \underline{\quad P_1 \quad} & \underline{\quad P_2 \quad} \\ \text{while } (s_1 == s_2); & \text{while } (s_1 != s_2); \\ \boxed{cs} & \boxed{cs} \\ s_1 = s_2; & s_2 = \text{not } (s_1); \end{array}$$

$s_1 = 1$
$s_2 = 0$

P_1 → comes → Condition false, so, P_1 enter into cs

$\boxed{P_1}$ → and s_2 assign in s_1. So $s_1 = 0$ and P_1 comes out.

After fully code is executed then process (P_1) comes out from $\boxed{cs}$.

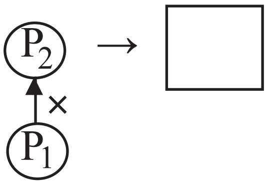
Now $s_1 = 0$ $s_2 = 0$

$P_2 \xrightarrow{\text{comes}}$ condition false, So P_2 enter into $\boxed{cs}$ $\boxed{P_2} \overset{\text{exit}}{\Rightarrow}$ $\left.\begin{array}{l} s_1 = 0 \\ s_2 = 1 \end{array}\right\}$

Progress → Second process executive

$s_1 = 0$
$s_2 = 1$

Now second process re-entry is unsuccessfull
b'Co2 condition is true for second (P_2) process.
So progress is not satisfied.



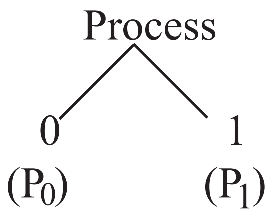
(3) Bounded wait :

Process is countable, so between is satisfied.

Peterson’s solution (2 process solution)

```
#  define N 2
#  define TRUE 1
#  define FALSE 0
        int turn;
        int interested [N];
  Void enter, region (int process)
   ξ
        int other;
        other  =  1 - process ;
        interested [process]  =  True ;
        Turn  =  process ;
        While (turn  =  process & interested [other]  =  =  True);
  ξ
        cs
  Void icone - resistor (ict process)
  ξ
      interested [process]  =  false ;
  ξ
        interested [0]  =  FALSE
        interested [1]  =  FALSE
```

P_0	P_1
other = 1	other = 0



(1) Mutual exclusion ✓

(2) Progress ✓

(3) Bounded wait ✓

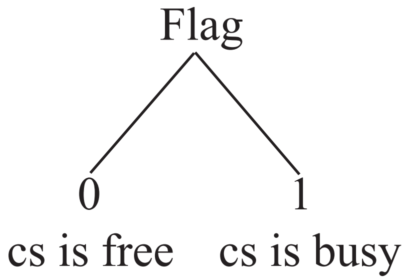
TSL (Test and Set Lock) Instruction Set

TSL Register Files

→ Copy the current value of flag into register and store the value of ‘1’ into the flag in a single arithmic cycle without any preemption.

Entry Section

(1) TSL R_i, m(flag)
(2) CMP R_i, # 0
(3) NJ2 to Step (1)
(4) $\boxed{\text{cs}}$
(5) Store M[flag], # 0



$\boxed{P_1}\boxed{P_2}$

flag = $\cancel{0}$ 1

$P_1 \rightarrow$ I $\quad R_1\ \boxed{0}$
II
III $\quad \boxed{P_1}$
IV $\quad$ cs

$P_2 \rightarrow$ I $\qquad$ flag = $\cancel{0}$ $\cancel{1}$ 1
II $\quad R_2\ \boxed{1}$
III

Mutual exclusion ✓
Progress ✓
Bounded wait ×

	Mutual exclusion	Progress	B.W
Lock various|cs	×	✓	×
Strict alternation & Decker's also	✓	×	✓
Peterson's Solution	✓	✓	✓
TSL	✓	✓	×

Q. LA = 32 bits

PAS = 64 MB $\rightarrow 2^{26}$ Bytes

Page size = 4 kB

Memory is Byte addressable.

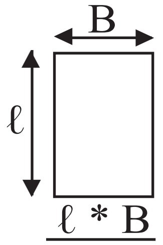
Page tale entry size = 2B

Approximate page table size in Bytes ?

Ans. Page table size = Number of pages in page table * PTES

$$= \frac{\text{LAS}}{\text{Page size}} * \text{PTES} = \frac{2^{32}\,\text{B}}{2^{12}\,\text{B}} * 2\text{B}$$

$$= 2^{20} * 2\text{B} = 2\text{ MB.}$$



Q. Consider a system having a page table with 4K entries and LA = 29 bits PA is > If system has 5/2 frames.

LA	
p	d
12	17

PA	
f	d
9	12

Q. LAS = 256 MB PA = 24 bits

PAS is divided into 8 KB frames. How many pages ?

$$\text{Number of pages} = \frac{\text{LAS}}{\text{Page size}} = \frac{2^{28}\text{B}}{2^{\text{B}}\text{B}} = 2^{15} = 32\text{ K.}$$

Q. LAS = PAS = 2^{16} Bytes Page Size = 5/2 B = 2^9

PTES = 2B

Page table entry contain besides other information like

1 bit for valid/Invalid

1 bit for reference

1 bit for dirty

3 bit for protection

How many bits are still available in page table entry to store the passing information ?

$$\longleftarrow 2\text{B} = 16\text{ bit} \longrightarrow$$

$$1 \quad \boxed{7 + 1 + 1 + 1 + 3}$$

$$\text{Number of fames} = \frac{2^{16}}{29} = 2^{\textcircled{7}}$$

Remaining bit = 16 - 13 = 3 bit

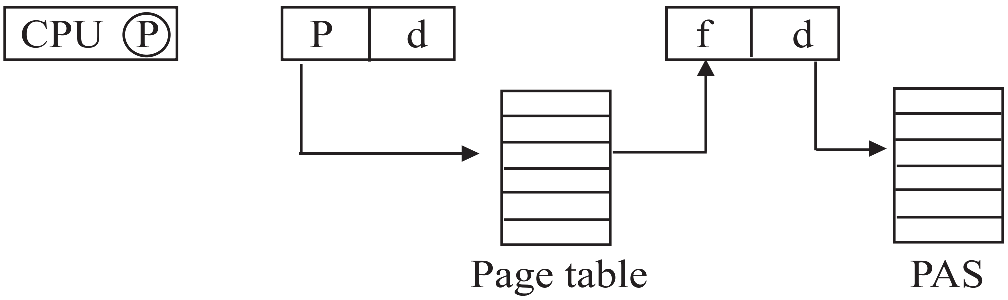
NOTE : PTES = 2B $\Rightarrow$ 16 bits, is not all the frame number of bits. If may contain some other informations.

PERFORMANCE OF PASING :

Without TLB

Main memory Access time = ‘m’

* Page table also stored in mm then EMAT = mtm = 2m



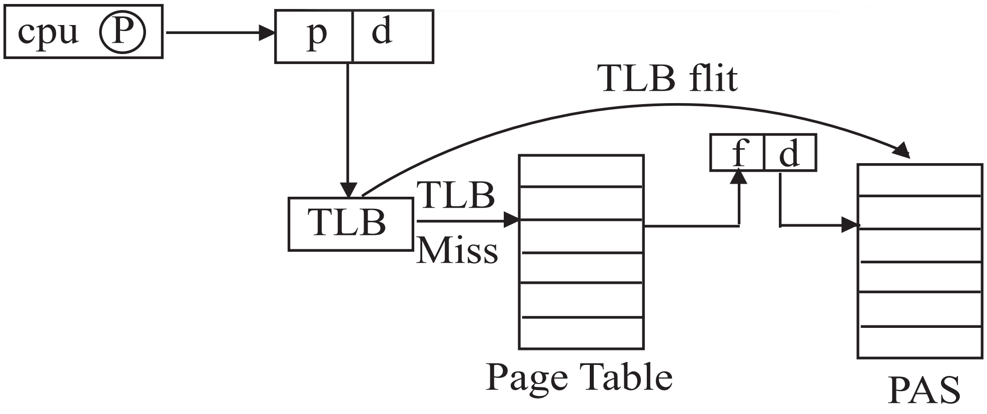
→ If the TLB (translation lookside buffer) is added to improve the performance them TLB contains frequently referred page number and corresponding frame number.

→ The TLB is associative High speed memory the TLB contains only a few of the page table entries.

→ If the page number is found then its frame number is immediately available.

→ If the page number is not found (TLB miss) the a memory reference to the page table must be made.

With TLB



TLB flit ratio = x

TLB Access time = c MMAT = m

Then effecitve memory Acess time

EMAT = x(c + m) (1 - x) (c + m + m)

$$= \underset{\substack{\text{TLB}\\ \text{filt}}}{\underline{x\ (c + m)}} + \underbrace{(1 - x)\ (c + 2m)}_{\text{TLB miss}}$$

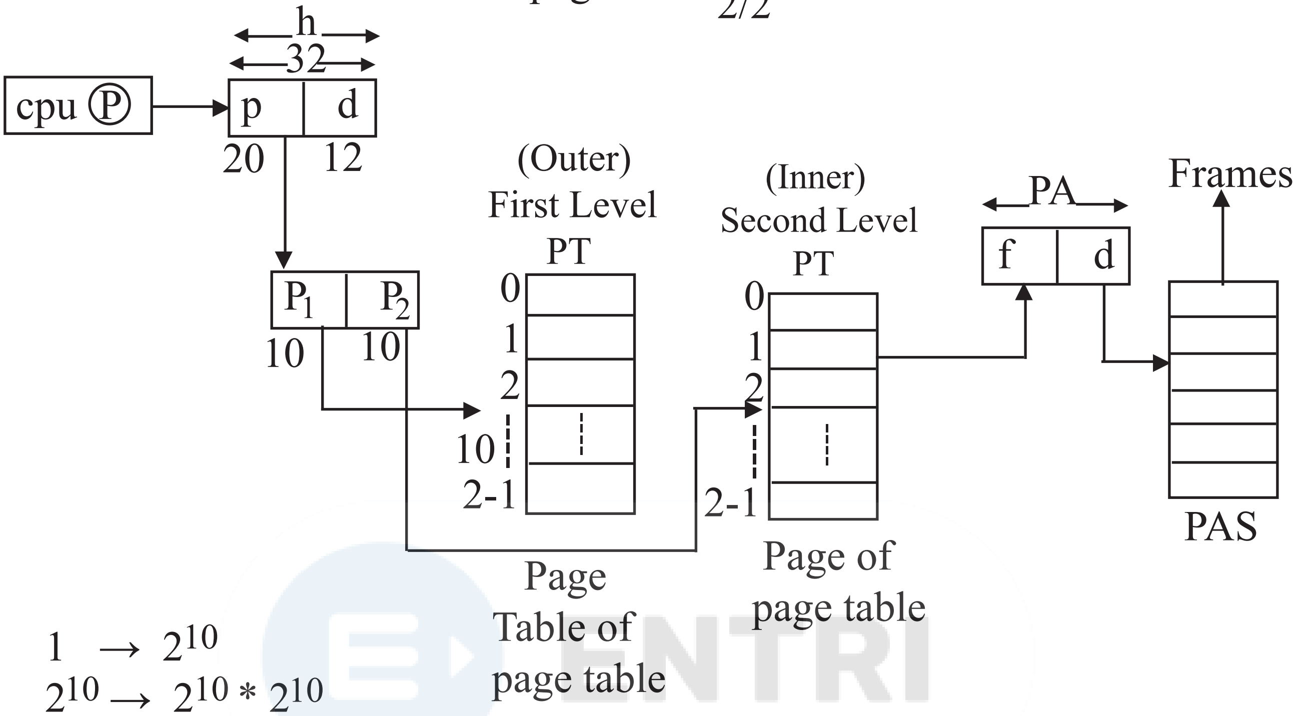
Multilevel Paging

Logical address = 32 bit PTES = 4B

Page size 4K words

Page size of page table = 1 KW

Number of pages = $\frac{2^{32}}{2/2}$ = 2^{20}



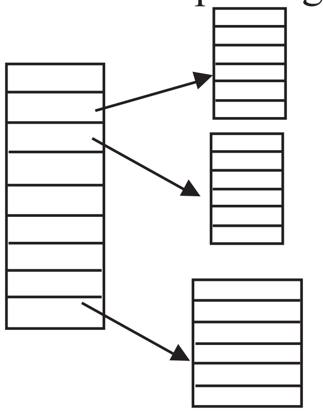
$1 \rightarrow 2^{10}$

$2^{10} \rightarrow 2^{10} * 2^{10}$

2^{20}

→ To avoid overhead to maintaining the large page table multilevel passing will be implemented.

Number of pages on pase table = $\frac{2^{20}}{210}$ = 2^{10}



P_1 = Number of bits required to represent the pages of page table.

P_2 = Number of bits required to represent the page size of page table.

Q. Consider a system with 2-level paging applicable. The page table has divided into 2k pages each of size 4k words. If PAS is 64M words which is divided into 16k frames & page table entry size is 4B. Then calculate

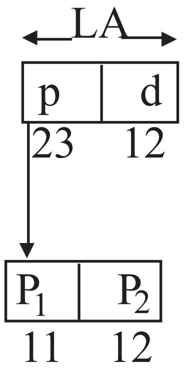
(1) Length of logical address

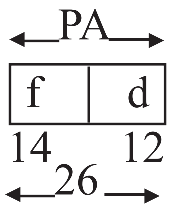
(2) Length of physical address

(3) Page table size of first level page table

(4) Page table size of second level page table.

Solution :





PAS = 64 M words

= $2^6 \times 2^{20}$ words

= 2^{26} words.

(1) LA = 35 bits

(2) PA = 26 bits

(3) Page table size at first level

= Number of pages at first level × PTES = $2^{11} \times 4B$ = 8KB

(4) Page table size at second level = Number of pages at second level × PTES = $2^{12} \times 4B$ = 16 KB

Performance of two-level paging

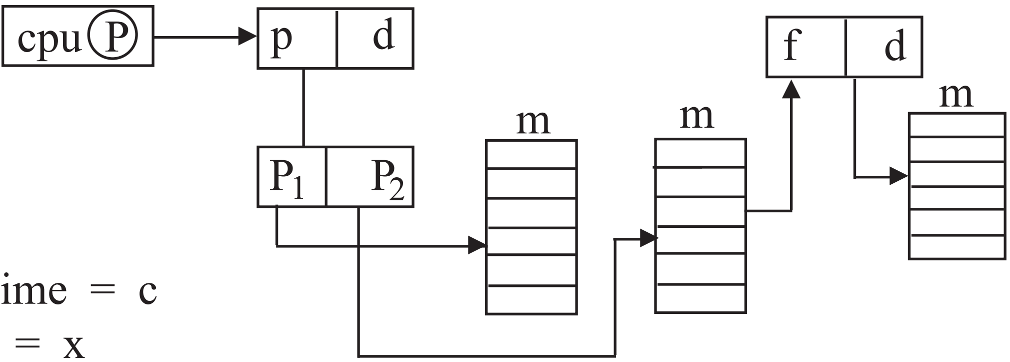
Without TLB

MMAT = 'm'

If page table is also stored in mm then

EMAT = (m + m + m) = 3 m

With TLB



TLB Access time = c

TLB hit ratio = x

MMAT = 'm'

Then EMAT = x(c + m) + (1 - x) (c + 3m)