3. Process & Process Synchronization & Deadlock

Q.1 What is Mutual Exclusion?

⇒ i) The portion in any program, which access a shared resources is called as the critical section or critical region.

ii) When a process is in the critical section disallow any other process to enter its critical section, i.e. no two processes should be allowed to be inside their critical section at the same.

iii) This is called as Mutual Exclusion.

iv) Primitives of Mutual Exclusion :-

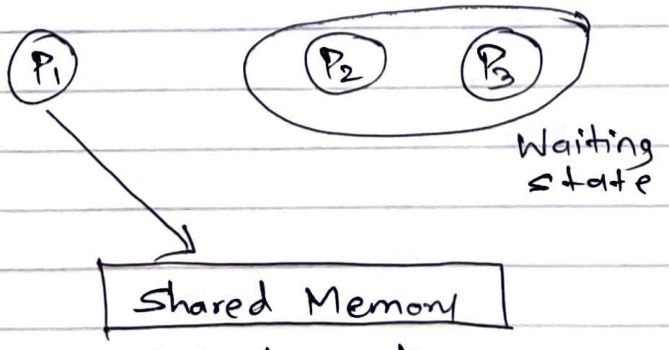
a) So each process must first request permission to enter its critical section.

b) The section of the code implementing this request is called as Entry Section

c) The critical section might be followed by a Leavel Exit section.

d) The structure of a process :-

Entry Section
Critical Section
Exit Section
Remainder section.



v) It's significance is -
1) It avoids race condition.
2) Prevents multiple threads to enter critical section at the same time.

Q.2 What are Semaphores ? Differentiate between counting and binary semaphore.

⇒ • Semaphore

⇒ A semaphore S is integer variable whose value can be accessed and changed by only two operations wait and signal.

- Wait and signal are atomic operations
- The wait operation on semaphore S,

```
Wait (S) : IF S > 0
           THEN S := S-1
           ELSE (wait on S)
```

- The signal operation on semaphore S,

```
Signal (S) : IF (one or more process are waiting on S)
             THEN (let one of these processes proceed)
             ELSE S := S+1
```

Counting semaphore	Binary semaphore
i) No mutual exclusion	i) Mutual Exclusion.
ii) Any integer value	ii) Value only 0 & 1
iii) More than one slot	iii) Only one slot.
iv) Provide a set of processes.	iv) It has a mutual exclusion mechanism.

Q.3 What is producer-consumer problem.? Provide solution to procedure consumer problem using semaphore.

⇒ i) In producer-consumer problem assumes that buffer is bounded buffer.

ii) That means there is a finite numbers of slots available in a buffer.

iii) While producing the items, if the buffe. is full producer process should be suspended.

iv) The code for producer process is as follows -

```
while (true) {

    while (counter == array_size){
        array [in] = nextProduced;
        in = (in + 1) % array_size
        counter ++ ;
    }
}
```

v) The code for consumer process can be modified as -

```
while (true) {

    while (counter == 0)

        nextConsumed = array[out]
        out = (out + 1) % ArraySize;

        counter--;
}
```

- Producer-consumer Problem using Semaphore:

⇒ The solution to producer, consumer problem uses three semaphore namely full, empty and mutex.

- The semaphore 'full' is used for counting the no. of slots available in buffer that are full.
- the 'empty' for counting the no. of slots that are empty.
- semaphore 'mutex' to make sure that the producer and consumer do not access modifiable shared section of the buffer simultaneously.

Q.4 What do you mean by busy waiting ?

⇒ i) Busy waiting is the limitation of semaphore definition.

ii) If critical section is not free (process is already executing in it) then other process trying to enter in critical section should loop continuously in entry code.

iii) This busy waiting is the problem as far as multiprogramming is considered.

iv) In this case a single CPU is shared among many processes.

v) This busy waiting waste CPU cycles which coould have been used by other processes effectively.

Q.5 Define deadlock. Explain necessary & sufficient condition of deadlock.

⇒ i) We know that all processes needs different resources in order to complete the execution.

ii) So in a multiprogramming environment, many processes may compete for a multiple no. of resources.

iii) In a system, resource are finite. So with finite no. of resources, it is not possible to fullfill the resource request of all processes.

iv) When the process requests a resource and if it is not available at that time, process enters a wait state.

v) There is a chance that waiting processes will remain in same state and will never again change state.

vi) It is because the ~~request~~ resource they have requested are held by other waiting processes.

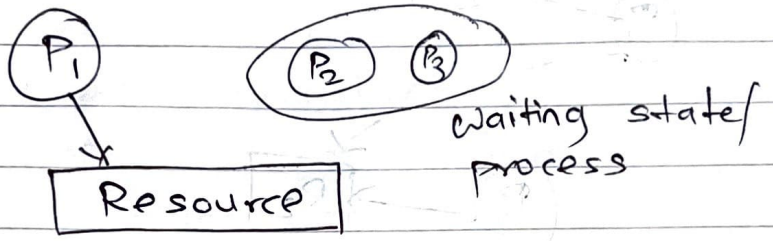
vii) When such type of situation occurs then it is called as deadlock.

- Necessary conditions for deadlock to occur are -

i) Mutual Exclusion

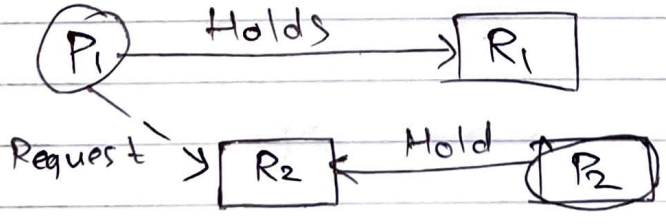
⇒ A resource at a time can only be used by one process.

- If another process is requesting same resource, then it must be delayed until the resource is released.



ii) Hold and Wait

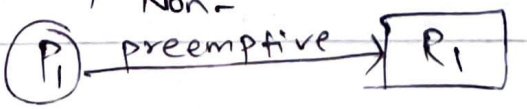
⇒ A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.



iii) No Preemption

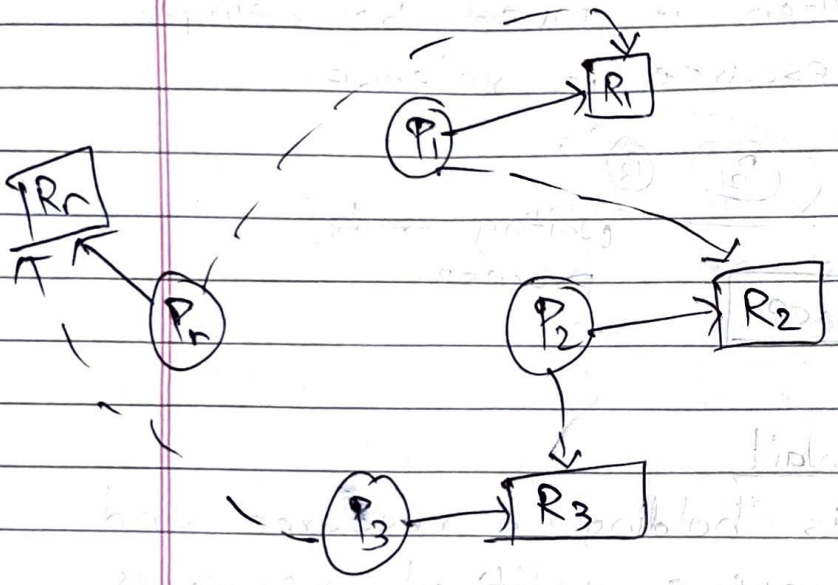
⇒ Resource cannot be preempted.

- Resource can released by process currently holding it based on its voluntary decision after completing the task.



iv) Circular Wait

⇒ A set of processes $(P_1, P_2, \ldots, P_n)$ such that P_1 is waiting for resource hold by P_2, P_2 is waiting for P_3 and P_{n-1} is waiting for P_n and P_n is waiting for P_1 to release its resources.



- All the above four mentioned conditions should occur for a deadlock to occurs.

Q.7 Explain deadlock prevention.

⇒ i) For a deadlock to occur, each of the four necessary-condition : mutual exclusion, hold and wait, no preemption and circular wait should hold simultaneously in the system.

ii) By ensuring that at least on one these conditions cannot hold, we can prevent the occurence of a deadlock.

iii) So deadlock prevention can be acheived by denying the holding of at least one of following four conditions.

a) Mutual Exclusion
b) Hold and Wait
c) No Preemption
d) Circular wait

a) Mutual Exclusion

⇒ This condition ensures that, resource should be used by single process at a time and it remains with using process in non-sharable ~~resource~~ mode.

In terms of resource, mutual exclusion means that multiple processes can't use the same resource at the same time.

For mutual exclusion not to happen, resource should be sharable.

2) Hold and Wait

⇒ This method denies hold and wait condition by ensuring that whenever a process requests a resource, it does not hold any other resource.

One of the following policies are used:

1) A process must request all resource before it begins execution. If all the needed resource are available, they are allocated to process. If one of requested resource are not available, none will be allocated and process would just wait.

2) Instead of requesting all resource before execution starts, a process may request a resource during it's execution if it obeys the rule that it request resources only when it holds no other resources.

---

3) ~~Circata~~ No preemption

⇒ No preemption is a necessary condition for a deadlock.

4) Circular Wait

Q.6 Explain deadlock avoidance method.

⇒ i) In deadlock avoidance, the system dynamically considers every request and decides whether it is safe to grant it at this point.

ii) A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never occur.

iii) The resource allocation state is defined by no. of available and allocated resources, and the maximum demand of process.

iv) Some techniques to Avoid deadlock

⇒

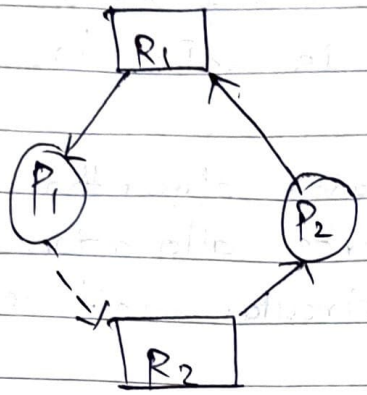
a) Resource-allocation graph algorithm

⇒ Deadlock can be describe with the help of directed graph called as Resource Allocation Graph (RAG)

- The claim edge in RAG is the future requested edge.
- When a process requests a resource, the claim edge is convert to a request edge.
- When a process releases a resource, the assignment edge is converted to claim edge.
- The request can be granted by converting request edge to assignment edge.
- If assignment does not convert into cycle in RAG then system is in safe system.
- Otherwise, if cycle is present, then

it is unsafe statements.

- For e.g;



In this example, the cycle is present in RAG.
Hence, system is in unsafe state.

b) Resource-requested Algorithm

⇒ This algorithm decides whether request of resource can be safely granted so as to system will remain in safe state.

- Let Reqi be the vector to store the resource request for process Pi.

1) If Reqi <= Needi, goto step 2
Else raise an error condition, since the process has exceeded its maximum claim

2) If Reqi <= Av, goto step 3
Else Pi must wait

3) Av = Av - Reqi
Allocᵢ = Allocᵢ + Reqi
Needi = Needi - Reqi

- Once the resource allocated, check to see if the system state is safe.

c) Safety Algorithm

⇒ This algorithm finds out whether system is in safe state or not.

- Let Current_avai and Marked be two vector of length n and p respectively.

1) Current_avai = Av
2) Initialize Marked as:
   for (i=1 ; i<=p ; i++)
      Marked[i] = false
3) Find a process Pi such that Needi <= Current_avai and Marked[i]=False
   If no such i exists then goto step 5
4) if (Found) {
      Current_avai = Current_avai + Alloc i
      Marked[i] = true
      Save process no in safestring[]
      go to step 3
   }
5) if (Marked[i]==true) for all processes, then system is in safe state
   PRINT safestring
   ELse System is unsafe.

d) Banker's Algorithm

i) It is applicable to the resource allocation system with multiple instances of each resource type.

ii) It is less efficient than resource allocation graph algorithm.

iii) Newly entered process should declare maximum no. of instances of each resource type which it may require.

iv) The request should not be more than total no. of resources in the system.

v) System checks if ~~reso~~ allocation of requested resource will leave the system in safe state.

vi) IF it will the requested resource are allocated.

vii) IF system determines that resource cannot be allocated as it will go in unsafe state, the requesting process should wait until other process free the resource.

viii) Following data structure is used

⇒

- A [m] → Array A of size m shows no. of available resources.
- M [n] [m] → 2D Array M shows maximum requirement resource by each process
- C [n] [m] → 2D Array C shows current allocation status of each process
- N [n] [m] → 2D Array N shows remaining possible need of each process

- For e.g.,

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	3	0	1	4	5	1	1	7	0	3	0	1
P_1	2	2	1	0	3	2	1	1				
P_2	3	1	2	1	3	3	2	1				
P_3	0	5	1	0	4	6	1	2				
P_4	4	2	1	2	6	3	2	5				

⇒

- Instances of each resource :
  - A = 12
  - B = 13
  - C = 6
  - D = 8

- Need Matrix :

	A	B	C	D
P_0	2	1	0	3
P_1	1	0	0	1
P_2	0	2	0	0
P_3	4	1	0	2
P_4	2	1	1	3

- We have Available resource = 0 3 0 1 and the need of P_0 and P_1 cannot be fulfilled as they required more resources.
- But the need of P_2 can be fulfilled.

- Need ≤ Available
  (0 2 0 0) ≤ (0 3 0 1)
  and (0 3 0 1) - (0 2 0 0)
  = 0 1 0 1 available in system

- P_2 Complete execution and Free the resource.
  ⇒ (0 1 0 1) + (3 3 2 1)
  = (3 4 2 2)

- P_1 : (3 4 2 2) - (1 0 0 1)
  = 2 4 2 1 available in system

- When P_1 finishes
  ⇒ (2 4 2 1) + (3 2 1 1)
  = (5 6 3 2)

- P_3 : (5 6 3 2) - (4 1 0 2)
  = (1 5 3 0)

- When P_3 finishes
  ⇒ (1 5 3 0) + (4 6 1 2)
  = (5 11 4 2)

- Not possible to allocate resource to P_0 or P_4.
- Hence, system is not in safe state.

Q.6 Explain deadlock detection and mention the recovery solution.

⇒ i) IF a system is not able to implement neither deadlock prevention or a deadlock avoidance algorithm, it may lead to a deadlock situation.

ii) For deadlock detection, the system must provide -

a) An algorithm that examines the state of system to detect whether a deadlock has occured.

b) And an algorithm to recover from the deadlock.

iii) It has two parts :-

a) Single Instance of Each Resource Type

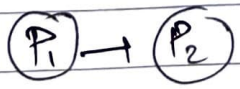
⇒ Requeries the creation and maintainence of a wait-for-graph.

For e.g.,

P_1 is requesting resource R_2 which is held by P_2

(RAG)

Above situation in RAG is represented as shown in wait for graph -



b) Many instances of Each Resource type
⇒ The wait for graph is not applicable to many instances of a resource type. In this case, we have to use detection algorithm similar to Banker's, which simply investigates every possible allocation sequence for the process which remain to be completed.

- Recovery Solution

⇒
i) Optimistic approach
⇒ Preempt some resource from process and give these resource to other processes until the deadlock cycle is broken.

ii) Pessimistic Approach
⇒ Abort one process at a time and decide next process to abort after deadlock detection.