Methods for handling deadlock

1. Deadlock prevention or Avoidance :-

The idea is not let the s/m into deadlock state.

2. Deadlock Detection & Recovery :-

Let deadlock occur, then do preemption to handle it once occured.

3. Ignore the pblm all together :-

If deadlock is very rare, then let it happen & reboot the s/m. This is the approach that both Windows and UNIX take.

— * —

1. Deadlock Prevention or Avoidance

* Provides a set of methods to ensure that atleast one of the necessary conditions cannot hold.

- Mutual Exclusion :-

→ To violate this condition, all the resources in the s/m should be in shareable mode.

→ But in s/m, there are some resources are non-shareable in nature.

→ So mutual exclusion must hold.

- Hold and Wait :-

→ One protocol that we can use requires each process to request & be allocated all its resources before it begins execution.

→ Alternative protocol allows a process to request resources only when it has none.

→ There 2 protocols has 2 main disadvantages :-

- Resource utilization may be low, since resource may be allocated but unused for a long period.
- Starvation is possible :- A process that needs several resources needs to wait indefinitely.

- No Pre-emption :-

→ If a process is holding some resources & request another resource that cannot be immediately allocated to it then all resources, the process is currently holding are preempted.

→ The process will be restarted only when it can regain its old resources, as well as the new ones that is requesting.

→ Alternatively, if a processes requests some resources, we first check whether they are available.

→ If they are, we allocate them.

→ If they are not, we check whether they are allocated to some other processes that is waiting for additional resources.

→ If so we preempt the desired resources from waiting processes and allocate them to the requesting process.

→ If the resources are neither available nor held by waiting process, the requesting process must wait.

→ A process can be restarted only when it is allocated new resources it is requesting & recovers all resources that were preempted while it was waiting.

• Circular Wait :-

→ A natural no: is assigned to every resource.

→ Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource number.

→ In case increasing order is followed, if a process requires a lesser number resource, then it must release all the resources having larger number and vice-versa.

→ This approach is the most practical approach & implementable.

→ However, this approach may cause starvation but will never lead to deadlock.

Eg:-

	R1	R2	R3	R4	R5
$P_1 \rightarrow$	R_2, R_4, R_5	/	R_5, R_4, R_2		

If P_1 needs R_3, then R_4 & R_5 are released & remains R_2. Then R_3 is allocated, then R_4, R_5.

9/3/2022

Deadlock Avoidance / Prevention Methods

* This strategy involves maintaining a set of data using which a decision is made whether to entertain the new request or not.
* If entertaining the new request causes the s/m to move in an unsafe state, then it is discarded.
* This strategy requires that every process declares its maximum requirement of each resource type in the beginning.
* The main challenge with this approach is predicting the requirement of the processes before execution.
* Banker's Algorithm is an example of a deadlock avoidance strategy.

Banker's Algorithm

* To implement Banker's Algorithm, follo: 4 data structures are used :-

1. Available - Single Dimensional Array. (i/p)
2. Max - Two Dimensional Array. (i/p)
3. Allocation - Two Dimensional Array (i/p)
4. Need = Max - Allocated (Two-Dimensional Array) (o/p)

Safe State Sequence → o/p.

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	4	3
P_2	2	0	0	3	2	2				1	2	2
P_3	3	0	2	9	0	2				6	0	0
P_4	2	1	1	2	2	2				0	1	1
P_5	0	0	2	4	3	3				4	3	1

Work = Available

Work		
A	B	C
3	3	2
2	0	0
5	3	2
2	1	1
7	4	3
0	0	2
7	5	5
0	1	0
7	5	5

Finish				
P_1	P_2	P_3	P_4	P_5
F	F	F	F	F
	T			
			T	
				T
T				
		T		

Check Need $\leq$ Available

(1) $P_1 \rightarrow 7,4,3 \leq 3,3,2$ ✗

(2) $P_2 \rightarrow 1,2,2 \leq 3,3,2$ ✓

$3,3,2 +$
$2,0,0 \rightarrow$ Allocation P_2
$5,3,2$ Update available.

(3) $P_3 \rightarrow 6,0,0 \leq 5,3,2$ ✗

(4) $P_4 \rightarrow 0,1,1 \leq 5,3,2$ ✓

$5,3,2 +$
$2,1,1 \rightarrow$ Allocation P_4
$7,4,3$

(5) $P_5 \rightarrow 4,3,1 \leq 7,4,3$ ✓

$7,4,3 +$
$0,0,2$ Allocation of P_5
$7,4,5$

(6) $P_1 \rightarrow 7,4,3 \leq 7,4,5$ ✓

$7,4,5 +$
$0,1,0$ Allocation of P_1
$7,5,5$

(7) $P_3 \rightarrow 6,0,0 \leq 7,5,5$ ✓

$7,5,5 +$
$3,0,2$ Allocation of P_3
$10,5,7$

$\therefore$ Safe State // $< P_2, P_4, P_5, P_1, P_3 >$

Safety Algorithm

(1) Let Work & finish be vectors of length 'm' & 'n' respectively.
- Initialize : Work = Available.
- Finish[i] = False ; For i=1, 2, 3, 4, ... n

(2) Find an i s.t both
- Finish[i] = False.

(3) $Need_i <= Work$ if no such i exists goto step (4).
- Work = Work + Allocation[i]
- Finish[i] = true.
- goto step (2)

(4) if Finish[i] = true for all i
- then the s/m is in safe state.

Resource Request Algorithm

1) If $Request_i <= Need_i$ — P_i Req × Need
Eg : 10, 5, 2 <= 7, 4, 3
Goto step (2) ; o.w, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i <=$ Available
Eg : 4, 3, 2 <= 3, 3, 2
Goto step (3) ; o.w, P_i must wait, since the resources are not available.

3) Have the s/m pretend to have allocated the requested resources to process P_i by modifying the state as follows :-

Available = Available − Request$_i$

Allocation$_i$ = Allocation$_i$ + Request$_i$

Need$_i$ = Need$_i$ − Request.

Eg: P_1 req : 3 2 1 ≤ 7 4 3 Need.

≤ 3 3 2 Available

∴ Available = 3 3 2 −
3 2 1
0, 1, 1

Allocation = 0 1 0 +
3 2 1
3, 3, 1

Need = 7 4 3 −
3 2 1
4 2 2

2. Deadlock Detection & Recovery

* When a deadlock detection Algthm determines that a deadlock has occured in the S/m, the S/m must recover from that deadlock..

* 2 approaches of breaking a deadlock :-

→ Process Termination

→ Resource Pre-emption.

→ Process Termination :- To eliminate the deadlock, we can simply Kill one or more processes. For this, we use 2 methods:

✓ Abort all the deadlocked Processes :

1. Aborting all the processes will certainely break the deadlock, but with a great expenses.
2. The deadlocked processes may have computed for a long time & the result of those partial computations must be discarded & there is a probability to recalculate them later.

Eg : process pid : 5906 Kill -9 5906

✓ Abort one process at a time until deadlock is eliminated :

1. Abort one deadlocked process at a time, until deadlock cycle is eliminated from the s/m.
2. Due to this method, there may be considerable overhead, because after aborting each process, we must run deadlock detection algorithm to check whether any processes are still deadlocked.

→ Resource Pre-emption :- To eliminate deadlocks using resource preemption, we pre-empt some resources from processes & give those resources to other processes. This method will raise 3 issues :-

✓ Selecting a Victim :

We must determine which resources and which processes are to be preempted & also the order to minimize the cost.

✓ Roll Back:

1. We must determine what should be done with the process from which resources are pre-empted. One simple idea is total rollback.
2. That means abort the process & restart it.

✓ Starvation:

1. In a s/m, it may happen that same process is always picked as a victim.
2. As a result, that process will never complete its designated task.
3. This situation is called Starvation & must be avoided.
4. One solution is that a process must be picked as a victim only a finite number of times.