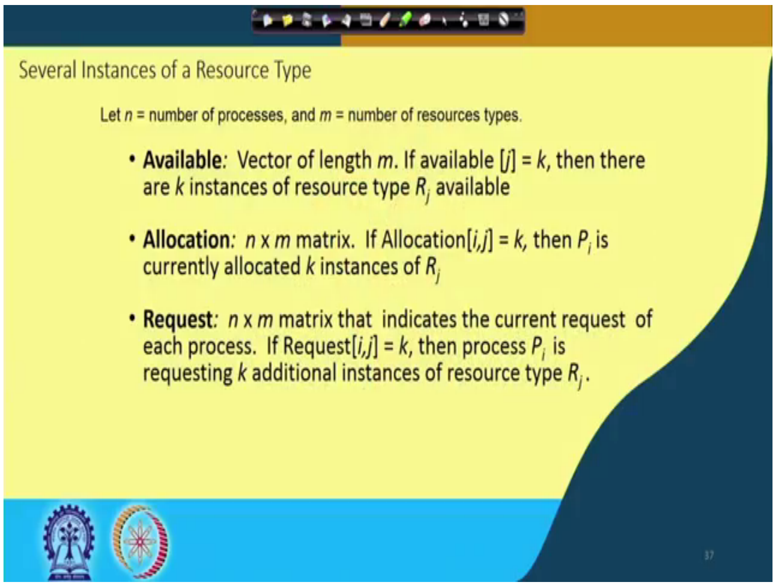
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 40
Deadlock (Contd.)

(Refer Slide Time: 00:29)



Several Instances of a Resource Type

Let n = number of processes, and m = number of resources types.

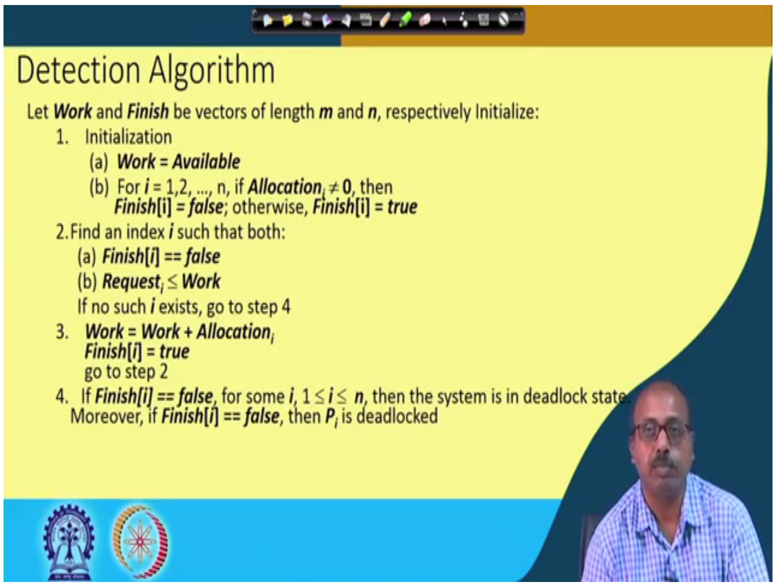
- **Available**: Vector of length m. If available [j] = k, then there are k instances of resource type R_j available
- **Allocation**: n x m matrix. If Allocation[i,j] = k, then P_i is currently allocated k instances of R_j
- **Request**: n x m matrix that indicates the current request of each process. If Request[i,j] = k, then process P_i is requesting k additional instances of resource type R_j.

Next we will look into several instances of resource types the previous deadlock detection algorithm. So, that was based on cycle detection mechanism so, that was using only one instance of each resource. Now if there are multiple instances then we can use an algorithm which is similar to bankers algorithm for this deadlock detection deadlock avoidance mechanism that we discussed previously.

So, again let n be the number of processes and m be number of resource types now we have got an Available array. So Available is a Vector of length m and if available j equal to k then there are k instances of resource type R j available. Then we have got a matrix Allocation n by m matrix if location i, j equal to k, then P i is currently allocated k instances of resource type R j and then Request so this is basically the current request that each process has.

So, n by m matrix that indicates the current request of each process, if request i, j equal to k, then process P i is requesting for k additional instances of resource type R j. So this is the situation so, current allocation current availability and current requests. So with these three things so we will try to see whether there is a deadlock in the system or not.

(Refer Slide Time: 01:46)



Detection Algorithm

Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:

```
1. Initialization
     (a) Work = Available
     (b) For i = 1,2, ..., n, if Allocation_i ≠ 0, then
           Finish[i] = false; otherwise, Finish[i] = true
2. Find an index i such that both:
     (a) Finish[i] == false
     (b) Request_i ≤ Work
     If no such i exists, go to step 4
3.   Work = Work + Allocation_i
     Finish[i] = true
     go to step 2
4. If Finish[i] == false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state
   Moreover, if Finish[i] == false, then P_i is deadlocked
```

So, the algorithm is similar to banker's algorithm that we discussed previously. So, here also that safety algorithm that we have so, some sort of safety algorithm will be run to see whether the system is deadlocked or not. So this Work can Finish so these are vectors of length m and n respectively when do we initialize them. So initialize Work to Available and then. So, if Allocation i is not equal to 0 then Finish i equal to false, otherwise Finish i equal to true.

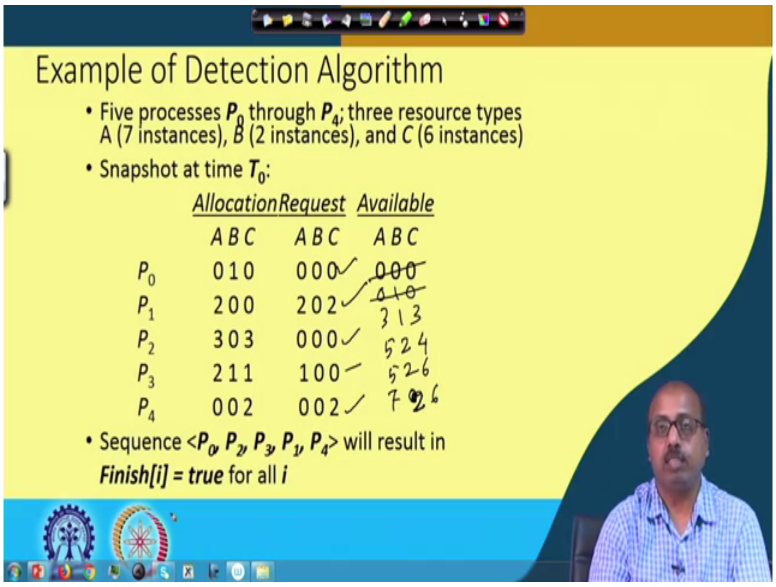
So, if Allocation is Allocation i is equal to 0 that means, this is so, no resource has been allocated. So, those processes need not be considered. So they that is that they are not contained they are not coming into this deadlock process at all because they are not they are not holding any resource. So they cannot participate in the deadlock situation. So, those processes are removed so this is the first modification that is done if there is no resource allocated to a process i; that means, it cannot be part of deadlock.

Then after that we try to find an index i such that it is not finished. So, Finish i is false that is basically it is holding some resource and it is Request is less or equal the currently available one that is Work array. So if Request is less than Work and it is not yet finished then we will we try to find an index i which satisfies this condition. So if no such i exists; that means, again the same thing either 2 a or 2 b has become false, if 2 a is false; that means, Finish i equal to false for if we cannot find. So, if we come to this state for Finish i equal to false or Request i less or equal 1 so, if we cannot find any such i then we come to step 4.

So, we after coming to step 4 if you find that there exists some process which is not yet; which is not yet finished whose Finish i is false; that means, the system is in a deadlock state and otherwise if Finish i is; Finish i is false then this process P i is deadlocked. Otherwise we will if Request i is less or equal Work so, the process has to wait for the resources to be allocate to available. So, that way it has to wait for that and then we update this Work array Work is Work plus Allocation i. So whatever resources are available so, that we allocate to the process P i and assume that the process P i is over and set Finish i to true.

So, with the currently available resources process has maximum request can be satisfied so, that we do and declare that the process is over. So, if we find a situation that Finish i is false for some process at which at the at point of time where we cannot allocate resources to any other processes. So in that at that time we say that the system is in a deadlock state.

(Refer Slide Time: 04:52)



Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0 ✓	~~0 0 0~~
P_1	2 0 0	2 0 2 ✓	~~0 1 0~~ 3 1 3
P_2	3 0 3	0 0 0 ✓	5 2 4
P_3	2 1 1	1 0 0 –	5 2 6
P_4	0 0 2	0 0 2 ✓	7 2 6

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in *Finish[i] = true* for all *i*

So, let us take an example and try to understand that how it works. So suppose we have got 5 processes P 0 to P 4 and there are three resources resource types A 7 instances B 2 instances and C 6 instances. Now if we take a snapshot at time T 0 suppose this is the situation that process P 0 has been allocated 0 1 0 that is 0 instance of A, 1 instance of B, and 0 instance of C. P 1 has been allocated 2 0 0, P 2 3 0 3 so like that and the allocation. So, request the current request that we have is P 0 is not requesting for anything, P 1 is requesting for 2 0 2 so, it is requesting for 2 instances of process A, 0 of B and 2 of C.

So, like that P 2 also not requesting. So, P 1 so this, P 3 is also not requesting so like that. So, when the algorithm starts so if you look into this algorithm it finds that if Allocation i is equal to 0 in that case finish i will be set to true. So, this P 0 and P 2 so, actually for none of them this allocation is allocation array is null. So, as a result all of them their Finish i is equal to false.

Now, currently available resources ABC they are 0 0 0, because A has got 7 instances and this 2 plus 3 plus 2 7 is already in use, B has 2 instances already in use and C has 6 instances that is also already in use so there is nothing available in the system. Now if we run this the detection algorithm so, it tries to see if any process can be finished with the currently available sources.

Now, you see this P 0 and P 2 so, these two processes they are not requesting anything. So, we can assume that without requesting any more resource the process P 0 and P 2 will finish off. So, if you compare with this banker's algorithm which was doing this deadlock avoidance. So, the maximum requirement was specified by the process so the system was checking whether though currently it is not requesting for anything it has the potential to ask for going up to the maximum requirement of the process.

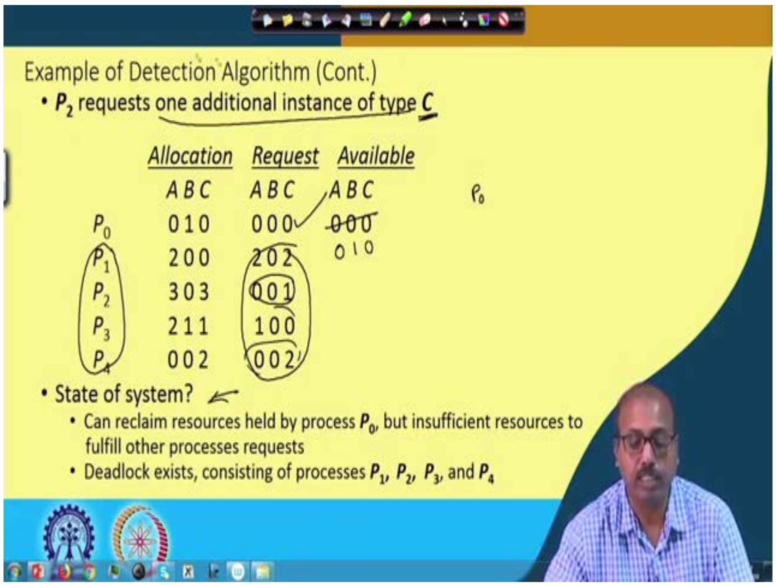
But in this detection algorithm so, we are not bothered about what will happen in future. So, we are just trying to see what is the current situation. So, we assume that if at present the process is not requesting for any resource we are optimistic and we say that at present if we can satisfy the request. So, at present at least the system is not in deadlock state so we can continue with the system so, that is the difference.

So here we do not do that max minus allocation as the maximum requirement the process can have so that particular calculation is not there. So here we just try to satisfy the request ok. So, here you see this P 0 and P 2 so, they can be satisfied immediately because they are not requesting for any more resource. So, if P 0 is finished then if P 0 is finished then it will return the current allocation. So, the allocation with the available will become 0 1 0 and then P 2 will finish so, P 2 finishes then this 0 1 0 will be becoming 3 1 3.

With that 3 1 3 we can satisfy this P 3 and once P 3 is over so, this 2 1 1 will be available. So, this will become 5 2 4 ok, with that 5 2 4 we can satisfy this P 4 so this 0 0 2. So, P 0 P 2 is already done. So, after P 3, so we can satisfy P 4 with this 0 0 2. So, this 0 0 2 so, that is satisfied so, this becomes 5 2 6 and with that we can satisfy P 1 and P 1 can be so, this 2 0 0 will be written so that is 7 2 7 2 6 it is ok. So, in this way we can satisfy the current requests for of all the processes. So, there exists a sequence in which all the resource all the processes requests can be satisfied.

So, at that point so, in this way the algorithm continues then it will have Finish i equal to true for all i. So, we can say that system is not in a deadlocked state. So, we can continue with the system operation. So, in future some other process may request for some resource and then the situation may become a deadlock situation. So, that is that possibility is there and then the deadlock detection algorithm will declare that there is a deadlock in the system.

(Refer Slide Time: 09:29)



Example of Detection Algorithm (Cont.)

- P_2 requests one additional instance of type C

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	~~0 0 0~~ 0 1 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

P_0

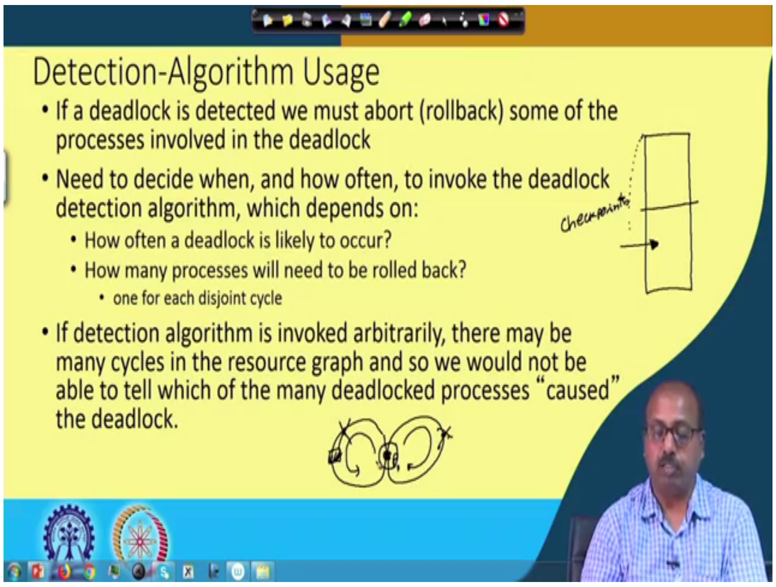
- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

So, suppose this is the situation, suppose this process P 2 so, it requires 2 more instances of type C. So, process P 2s P 2 sorry one additional instance of prior type C. So, process P 2's requirement become 0 0 1, now you can say that whether the system is in say deadlock state or non deadlock state so, let us try to see. So P 0 can be satisfied P 0 equals 0 0 0. So, once P 0 is done then this current availability becomes 0 1 0, with this 0 1 0 you see you cannot satisfy any of this request. So, this is done. So, 2 0 2 cannot be satisfied, 0 0 1 cannot be satisfied, 1 0 0 cannot be satisfied, 0 0 2 also cannot be satisfied. So, none of the requests can be satisfied.

So, the all the processes they will now be waiting for the resources to be released which are held by other processes. So, system will go into a system is in a deadlock state. So, if we try to look into what is the state of the system, we can reclaim resources held by process P 0, but insufficient resources to fulfill other process requests. So, deadlock exists and consisting of processes P 1 P 2 P 3 and P 4 so, whichever process the Finish i equal to false that becomes a part of deadlock.

So, for P 0 Finish i is true, but for P 1 P 2 P 3 P 4 Finish i remains equal to false so, they become part of these deadlocks. So, all these four processes they form the part of deadlock. So, this deadlock detection algorithm will do like this and it will be; it will be able to detect if there is a deadlock in the system. Now, once we have detected deadlock, then what can we do.

(Refer Slide Time: 11:16)



Detection-Algorithm Usage

- If a deadlock is detected we must abort (rollback) some of the processes involved in the deadlock
- Need to decide when, and how often, to invoke the deadlock detection algorithm, which depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

So, if a deadlock is detected we must abort or rollback some of the processes involved in deadlock because the so rollback is a technical term rollback means. So, we have to; we have to take the process out of the system and whatever it is actually in the middle of some computation. So, the difficulty is that it is holding some resources. So, we have to take back the resources from the process.

So, now you see that a process may be it is a big one it is a big one and as the process was at this point when this deadlock has been detected and the process has to be rolled back. Now one possibility is that we can restart the process from the beginning, if we restart the process from the beginning then lots of computations are to be done again or other possibilities that we can have some safe points within the program which we called checkpoints. So, you can have some checkpoints and we can restart that process from there now from the nearest checkpoint.

So, these checkpoints are such that at this point the process will not hold any resource ok. So, as if the as far as this resource allocation is concerned as if the process is coming afresh ok. So, or we can go back to the beginning of the process and start from there. So, if deadlock is detected we must abort a rollback some processes involved in the deadlock. So, whether roll back rolling back one process is sufficient or not that depends the way in which these deadlock cycles are formed. So, it may be that we need to roll back multiple processes and need to decide when and how often to invoke the deadlock detection algorithm.

So, as we have said that deadlock detection algorithm so it is costly so, it is quite taking quite some time. So, to answer this question let that is when should we run this deadlock detection algorithm. So, how often a deadlock is likely to occur? So, that is the first question that we need to answer.

If we say that the possibility of occurrence of deadlock is pretty high in that case we have to run this deadlock detection quite frequently, otherwise we can do it at a much longer interval of time and how many processes will need to be roll back one for each disjoint cycle. So, if we have several cycles for single instance situation the wait for graph so, we can have several cycles and for each cycle that we have there so, we have to roll back one process.

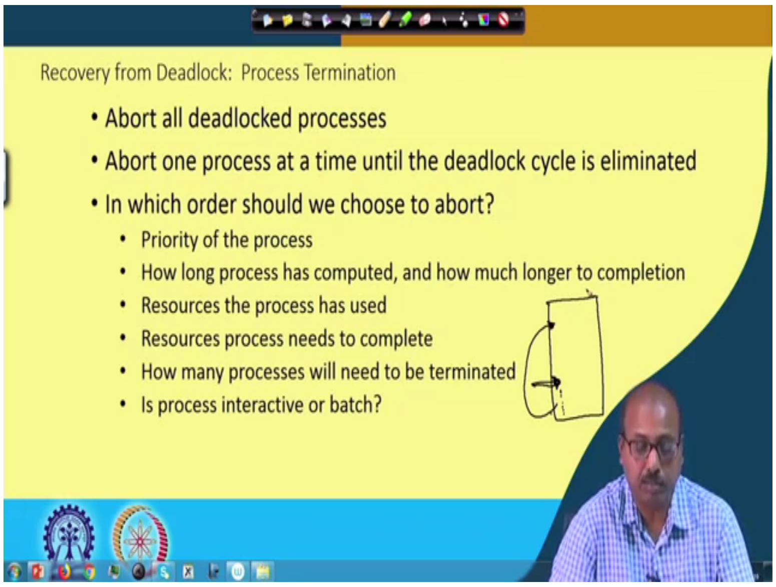
And for this multiple instance one so, there is no straight cut answer. So, you have to see after rolling back whether the system is still in deadlock or not by running that detection algorithm and if the detection algorithm again finds a deadlock in the graph in the system then we have to again roll back some other processes so, that way it is quite costly.

If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlock processes caused the deadlock. So, it may so happen that once we run this deadlock detection algorithm so, there are say 2 cycles in the process ok. So, this is the this is one process P 1 say may say P 1 which is involved in 2 cycles. So, it is making a deadlock with this processes as well as it is making a deadlock with these processes.

So, ideally we should be able to see the situation that process P 1 is the dead locking process. So, we should try to remove P 1, but it may so, happen that we remove one process from here. So, as a result this cycle is broken, but the other cycle still remains then again you have to select some other process and again remove that process.

So, it may so, happen that if this process P i is involved in deadlock with multiple cycles then for each cycle one process has to be rolled back so that can happen. So, this is not a very easy thing to do to determine which process is the best one to be rolled back. So, if detection algorithm is invoked arbitrarily there may be many cycles in the resource graph and so, we would not be able to tell which of the many deadlock processes caused the deadlock. So, that is here this process P 1 as we are talking about. So, that makes it difficult of course, we do not have much choice there.

(Refer Slide Time: 15:44)



Recovery from Deadlock: Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  - Priority of the process
  - How long process has computed, and how much longer to completion
  - Resources the process has used
  - Resources process needs to complete
  - How many processes will need to be terminated
  - Is process interactive or batch?

Now, if we want to recover from deadlock. So, the methodology to be adopted is process termination, we have to terminate the process. So, abort all deadlock processes this is the most easy and I should say the robust approach.

So, whichever process comes in the deadlock cycle you remove all those processes or abort although all the deadlocked processes. Another so this is I should say a brute force method and another method maybe that about one process at a time until the deadlock cycle is eliminated. So, you try removing one process and see whether there is the a still deadlock in the system, if there is still deadlock in the system you try removing another sub process. So, this way we can remove one process at a time until the deadlock cycle is eliminated.

Now in which order should we choose to abort? As I said that there may be a multiple number of processes which are involved in the deadlock. Now the first the second suggestion says that you abort one process at a time so, which process to abort. So, certain things we need to consider for example, priority of the process. So, if a process is of high priority then it is advisable that we do not abort that process at the beginning, when other processes low priority process is removing them so, if we are if we could resolve the deadlock then that is better. So, higher priority processes they should not be rolled back.

And how long process has computed and how much longer to completion. Now, you see that it may so happen that this is the total program code that the process has to execute and it is at this point of time. Now you can say that this is the total length of the process and the process is at this point.

So, it is likely that it has got this much amount of code left for execution, but this is a very crude estimation because you see that from this point there may be a jump and it goes back to this point again. So, that can happen, but this may be a measure that if the process is executing for a long time then it is very much likely that process has almost done with it is complete or computation.

So, if we allow the process to continue then possibly it will be finishing off soon and release all the resources and at the same time since it has computed for a long time so, it has done lot of computation. So, if it is rolled back then it will do those computations again so, system resources will be again required.

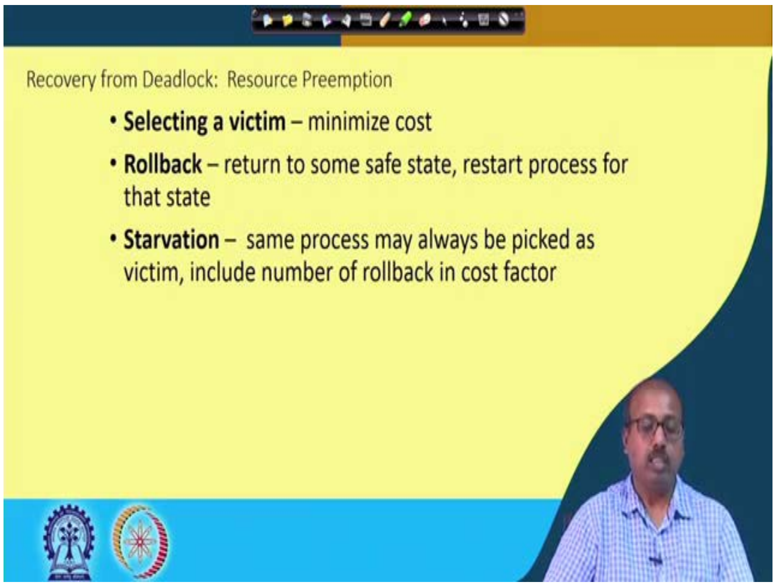
So, that way one positive side for this process the positive logic for this process long process is that it has done almost the computation over and it has compute computed over a long time. However, on the other side you can say that since the process is involved in deadlock and possibly it has grabbed lots of resources over the time and it has interfered with the system operation for quite some time.

So, if we roll back this process then maybe other processes they will find it easy to continue through the system to pass through the system and that way the possibility of deadlock will be less than next time. So there are both positive and negative arguments in favor of this long process and against it then the resources the process has used. So, if the process has used lots of resources then it is likely that the it will finish off soon, the process have issue used only a small number of resources, then maybe again you can say it is a good process it is not requiring lots of resources. So, again there are plus and minus for both the sides.

Then resources process needs to complete how many resources are needed for completion. So, if that number is high then the possibility that the process will again come into a fall into a deadlock cycle so, is high. So, that process may be that want to abort and how many processes will need to be terminated. So, how many processes we will need to be terminated like if I go like this then how many processes are going to be terminated so that can also be calculated and is process interactive or batch type.

So, if your process is interactive in nature now if we reboot or restart that particular process then naturally the user will again have to enter all the data. So, user will feel bad about the system whereas, the back system so, there we do not have that problem because batch process anyway runs in the background. So, if we run it in the backgrounds then the process will definitely it will not cause much harm as far as the user feelings are concerned so that is what we have got arguments so, those all those points are to be considered.

(Refer Slide Time: 20:36)



Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

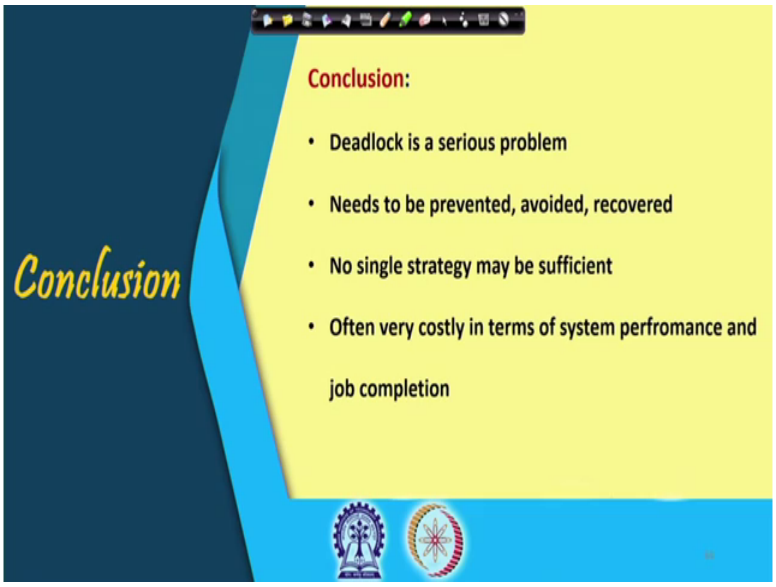
Now how to recover from deadlock so, recovery from deadlock is one possibility is resource preemption. So, we take the resource back from the process maybe the process continues, but the resource is taken back, now due to this taking back of resources so it may so happen that process has to redo some computation again later. So, that is there, but this preemption this resource preemption can be there.

So, naturally the problems or the sub problems that we need to solve is selecting a victim. So, you need to select which process should be which process should be taken as a victim and for taking back the resource, then rollback return to some safe state and restart process for that state. So, basically roll back as I as I was telling that there may be some checkpoints in the program so, we roll back the process till that previous checkpoint for one and from that point the process restarts.

Then starvation same process may always be picked as victim so, include number of rollback in cost factor. So, it was so happen that every time the deadlock is detected the first time the deadlock is detected one process is taken as a victim and that is rolled back. And since this process could not finish so, again the it will come to the system and then again next time it also falls into a deadlock and then again as a victim so, this one is selected.

So, that can happen like the same process is selected as victim again and again so that is that will create starvation for the process. So, we can have some rollback factor number of rollbacks in the cost factor of the process so that this starvation will be less.

(Refer Slide Time: 22:26)



Conclusion

Conclusion:

- Deadlock is a serious problem
- Needs to be prevented, avoided, recovered
- No single strategy may be sufficient
- Often very costly in terms of system perfromance and job completion

So, deadlock is a serious problem definitely and it needs to be prevented, avoided or recovered. There is no single strategy that may be sufficient and it is often very costly in terms of system performance and job completion.

So, this way as we as I said that we can have this deadlock situation occurring in the system, sometimes we may not do anything as I as is done in the say the UNIX operating system like that ostrich algorithm. So, it does not do anything if the deadlock happens then the system administrator will kill some of the processes or reboot some of the reboot the whole system in the worst case and so, we have got this thing ostrich algorithm, but to be more serious about the deadlock we should have this prevention, avoidance or recovery mechanism introduced.

So, and as we can say that no single strategy may be sufficient so, as we have said that prevention. Prevention means we have to violate one of the 4 conditions like mutual exclusion, hold and wait, no preemption and that circular wait. Now, a violating of if these conditions for all types of resources may be difficult. So, for certain resource types so, it may be possible that we violate one of those conditions so as a result it does not go into the deadlock so, deadlock is prevented.

So, if deadlock is not prevented then we can go for the avoidance type of strategy. So, avoidance strategies are fine, but this is over restrictive in some sense, because it is not allowing the system to proceed if there is a even small amount of chance for deadlock occurring in future. So, it is just looking forward and trying to see whether there is any possibility of deadlock that can occur in eventually it may occur and as I say that situation may not occur at all. So, maybe after going to an unsafe state, the system comes back to the safe state so that can happen.

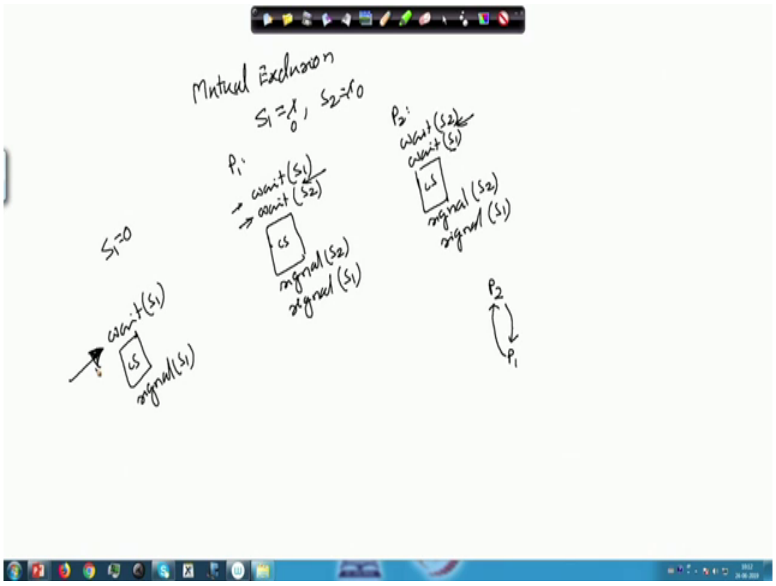
So, this avoidance policy is good, but it is some it is too restrictive and then we have got this detection and recovery based schemes. So, detection recovery based schemes so, they are fine, but again the question is how frequently we should run this detection algorithm and as it is said the detection algorithm will be it will run when the system throughput is pretty low even if the degree of multi programming is pretty high.

So, that is a; that is a indication that there is some processes are in deadlock state. So, none of them can proceed as a result there is a the system throughput has become low. So, then we have to run this deadlock detection algorithm and then some of the processes which are involved in the deadlock situation are detected and they are rolled back so that this deadlock is recovered.

Of course we have got priorities of the processes and other issues to determine like which one should be selected as victim. So, this way most single strategy is sufficient and often it is very costly in terms of system performance and job completion, because system performance is becoming poor and this job completion in terms of number throughput of the system so that is becoming pretty low. So, these are the things that we have to keep in mind.

So, deadlock is definitely a serious problem and you see that while solving many problems like a critical section problem and all there also we have to be careful about this deadlock situation like one typical situation is like this suppose we are trying to solve a mutual exclusion problem.

(Refer Slide Time: 26:19)



So, we are trying to solve a mutual exclusion critical section problem and then in this critical section problem so, we are solving using semaphore. So, we have got semaphore say a S o1ne which is initialized to one and we have got another semaphore S 2 which is also initialized to 1. Now the process P 1 that we design is like this so, this is wait for S 1, then we have got wait for S 2 and then we have got the critical section code. So, this is the critical section code that we have and after that we have got signal S 2 and signal S 1 whereas, process P 2 while writing it so, we write it like this.

So, it is wait on S 2, then wait on S 1, then this is the critical section code and after that we do a signal on S 2 and signal on S 1 ok. So, in this case you see that otherwise the programs are fine the processes will run pretty well, but the difficulty is suppose this process P 1 comes it executes up to this much. So, S 1 has become equal to 0 and this one the process P 1 gets descheduled and process P 2 comes P 2 executes up to this much.

So, this is S 2 becomes equal to 0, now after that when the process P 2 it will execute a wait on S 1. So, this process P 2 is waiting for this semaphore S 1, but S 1 is already 0 and S 1 is currently with process P 1. So, I can say this process P 2 it is waiting for process P 1. So, after some time process P 1 will come into the system and process P 1 will execute the next line wait on S 2 and S 2 is currently equal to 0 and S 2 is held by P

2. So, P 1 is actually waiting for P 2. So, this is the wait for graph for this particular situation. So, this is again a deadlock condition.

So, you see that even if this system the programs that are written by the same programmer or the same application. So, it may so happen these semaphores are used in such a fashion that it creates difficulty, sometimes difficulty comes because while writing this solution to some concurrent programs. So, instead of initializing the semaphore to 1 we initialize the semaphore to 0 for example, suppose we write like wait S 1 and then this is the critical section code and then we have got this signal S 1 and this semaphore has been initialized to 0. So, then at this point itself the process will be waiting forever so, process will never progress.

So, but this is not a deadlock situation because this process is not hampering others, but it may so happen that a number of processes they are becoming they are using a set of semaphores and in such a fashion that they form a in the of they form a cycle in the wait for graph. So, these situations should be avoided and while try trying to develop concurrent applications. So, we should be able to see this situation and we should write it properly that this semaphore usages are proper. So, with this we conclude this discussion on deadlock in the next class we will start with the memory management policies.