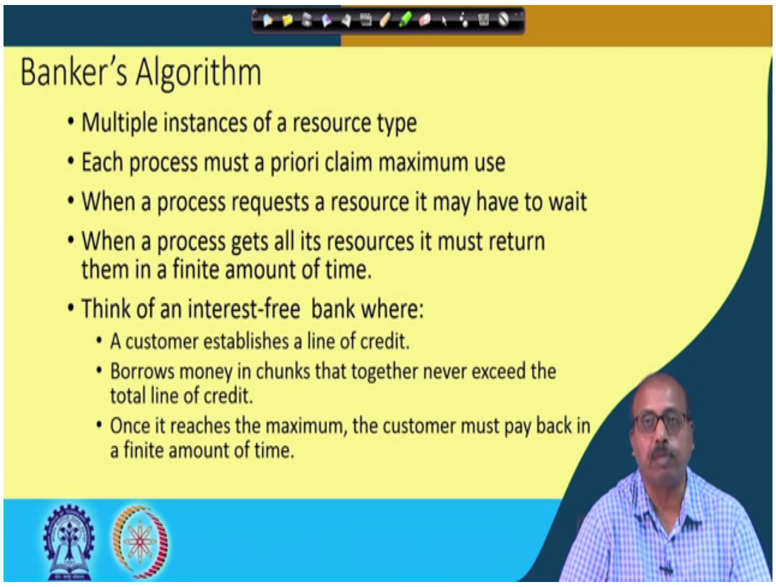
Operating System Fundamentals

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

Lecture - 39
Deadlock (Contd.)

In our last class, we have been discussing on Deadlock avoidance strategies and in that we have seen cycle detection algorithm for single instances resources.

(Refer Slide Time: 00:31)



Banker's Algorithm

- Multiple instances of a resource type
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time.
- Think of an interest-free bank where:
  - A customer establishes a line of credit.
  - Borrows money in chunks that together never exceed the total line of credit.
  - Once it reaches the maximum, the customer must pay back in a finite amount of time.

And for multiple instances so, there is another algorithm known as Banker's algorithm. So, we started discussing on this. So, we will continue on this in today's class. So, in banker's algorithm it is applicable if there are multiple instances of resource types. So, each resource type may have multiple instances and so the requirement is that each process must a priori claim maximum use. So, each process must declare like how many resources it wants at most for each type.

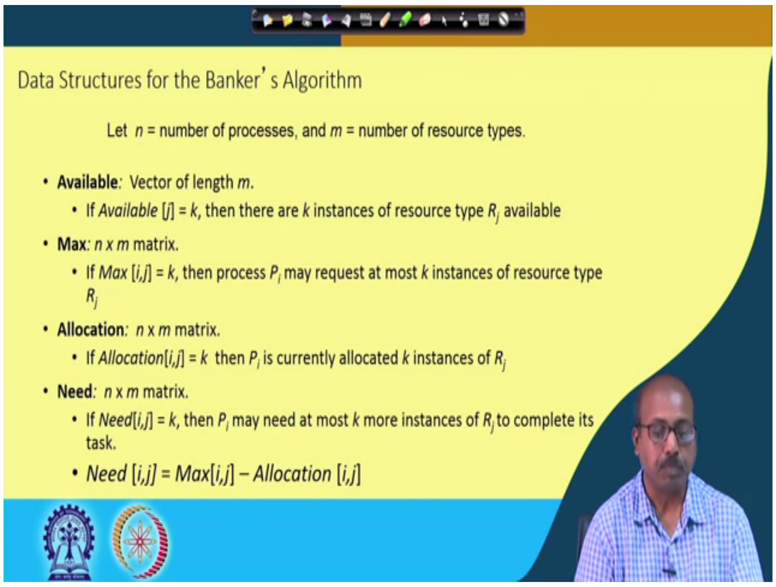
And when a process request a resource it may have to wait. Like, even if the resource is available, so it may not be allocated. So the system will try to decide like if I do this allocation whether there is any possibility of going to a deadlock situation or not. If there is a possibility of going to a deadlock situation, then the resource will not be allocated even if the resource is available.

So, when a process; so another condition that is there is when a process gets all its resources it must return them in a finite amount of time. So, it is not that a process has been given all the resources and still it is just going on doing something without releasing the resource. So, within a finite amount of time the all the resources given to it must be returned. So, it is based on the banking philosophy.

So, think of an interest free bank where a customer establishes a line of credit in which the customer declares that what is the maximum amount of credit that he or she may need. And it borrows money in chunks that together never exceed the line of credit. So, every time the customer needs some money so, it the customer asks for some money, but at no point of time so total money required is more than whatever is the maximum of the credit. So, once it reaches maximum, customer must pay back in a finite amount of time. So, once the total credit, total borrowed money becomes equal to the credit limit, then the customer must return all the money.

So, that is banking philosophy. So, it ensure that the bank itself does not get bankrupt if all the customers they ask for money within their credit limit and then it is unable to satisfy any of the customers, so that should not happen. So, it should be able to satisfy at least one customer, so that customer finishes off the job and returns all the money to the bank with that money the bank will satisfy some other customers and then the whole system will proceed. So, that is why this algorithm is known as banker's algorithm.

(Refer Slide Time: 03:12)



Data Structures for the Banker's Algorithm

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

- **Available**: Vector of length m.
  - If *Available* $[j] = k$, then there are k instances of resource type R_j available
- **Max**: $n \times m$ matrix.
  - If *Max* $[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: $n \times m$ matrix.
  - If *Allocation*$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: $n \times m$ matrix.
  - If *Need*$[i,j] = k$, then P_i may need at most k more instances of R_j to complete its task.
  - *Need* $[i,j]$ = *Max*$[i,j]$ – *Allocation* $[i,j]$

So, here the system is a considered to be in two different step or three possible states deadlock state and possibility of deadlock and there is no deadlock; so, the there is no possibility of deadlock. And we have seen that the when there is no possibility of any deadlock to occur so that is called as safe state.

When there is a possibility of deadlock from a particular state, so that is called unsafe state, though unsafe state means it is not yet deadlocked. So, it may so happen from the unsafe state the system comes back to the safe state, but there exist a chance that processes may request resources in such a fashion that the system may lead to a deadlock situation.

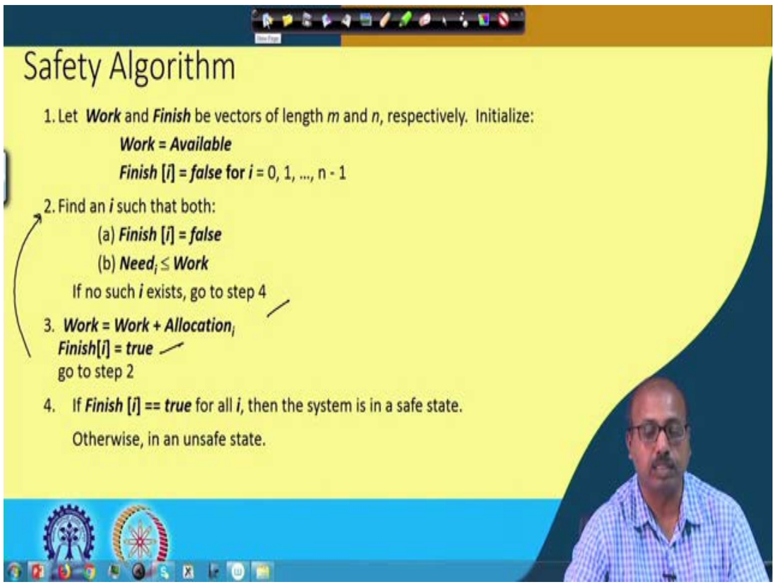
So, banker's algorithm it works for say n number of processes and m number of resource types. So, available array so it is a vector of length m. So, Available j is equal to k if there are k instances of resource type R j Available. So, it tells at any point of time how many resources of each type Available. Then there is a Max array so which is n cross m matrix. Max i, j this is equal to k, means that process i may request at most max k instances of resource type R j.

So, we can understand that Max array is a static array because this is declared by the processes at the very beginning how many resources, how many instances of each resource type the process may need at most, so that is the Max array. And Available so this is dynamic. So, at any point of time as the resources are being allocate to processes. So, these Available entries will decrease and when the resources are returned available entries will again increase.

Then Allocation, so this is also a dynamic array. So, it tells the it n cross m array the entries will change over time. So, Allocation i, j equal to k means process P i is allocated k instances of resource R j at this point of time. So, the Need array is basically a derived matrix. So, Need i, j equal to k, then P i may need at most k more instances of R j to complete the task. So, Need i, j equal to Max i, j minus Allocation i, j. So, Max is the maximum requirement of process i for resource type j and Allocation is currently allocated number of resources of type j to process i.

So, their difference gives us the Need that is the maximum number of more resources that the process i may Need of type j.

(Refer Slide Time: 05:46)



Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:
   Work = Available
   Finish **[*i*]** = ***false*** for *i* = 0, 1, ..., n - 1
2. Find an *i* such that both:
   (a) ***Finish*** **[*i*]** = ***false***
   (b) ***Need***$_i$ ≤ ***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*] == ***true*** for all *i*, then the system is in a safe state.
   Otherwise, in an unsafe state.

So, this safety algorithm, so this is the major part of this banker's algorithm. So, it is works with the two temporary arrays Work and Finish. Work is a vector of they are of vector length m and n respectively. So, initially Work is equal to Available. So, whatever is the current availability of each resource so, that is there in Work. And Finish i; so, Finish i equal to true means the i-th process has been satisfied with all its resources. So, we can consider that the process is over; process is true, it is finished. So, initially since none of the processes have been determined to be able to complete with the Available resources so, we have this Finish i equal to false for all i.

So, the next step is to find an i such that both Finish i equal to false and Need i less or equal Work. So, Work is currently holding the currently available number of resources that is there in the system. So, if the need of the i-th process is less than the currently available resources of each type ok. So, then in that case we can say that whatever resources are requested by process i can be given to it. So, process i will finish off. So, Finish i so we try to determine an i such that Finish i is false and Need i is less or equal Work.

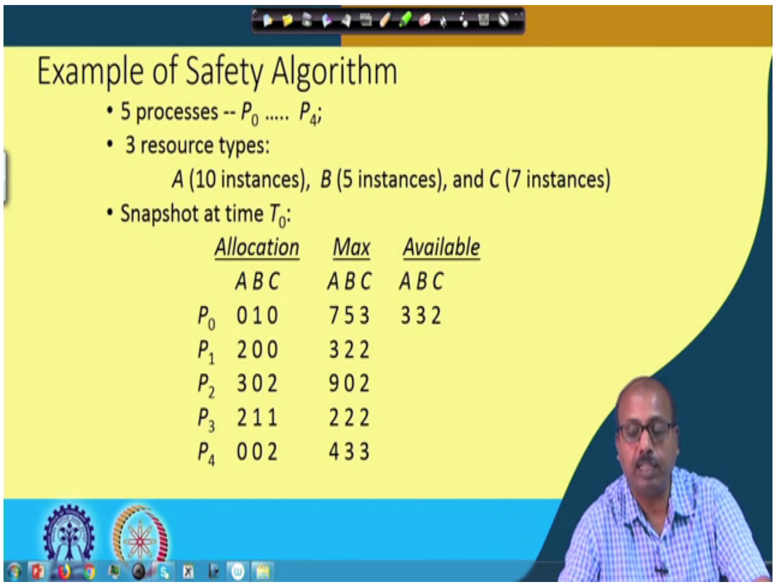
If no such i exists, then there are two possibilities either of the two conditions a or b may become false ok. So, if no such i exists, then we come to step number 4. So, in this case one possibility the Finish i is true for all i. So, we could not find any i for which a Finish i is false. So, all processes could be satisfied with their resource requirements. So, all of them are considered to be completed. So, Finish i equal to true if it is true, if it is the case in that case the system is in a safe state. So, all the with the currently available resources all the maximum request of all the processes can be satisfied. So, naturally so that is there so like we can say that the system is in safe state.

However, the second part so, if we have come to step number 4, following the condition 2 b becoming false; that means, the Need i is for none of the processes their need is less than whatever is currently available. So, all the processes they need more resources then that are available at present to finish off ok. So, may be they can they will ask for all those resources and the system will not be able to allocate those resources to all the processes, none of them will finish off. So, in that case the system is in unsafe state.

So, if we find that the system so if we on the other end if we can find such an i as it is given here so, if we can find such an i, then we come to step number 3 and then this Work array is updated as Work equal to Work plus Allocation i because whatever is currently allocated to process i, so, they once they it is satisfied once its requirements are satisfied then it will return all the resources. So, Work array will have all the currently allocated resources given back and Finish i will become true. So, we can consider that the i-th process is done.

And go to step number 2 again to see whether we can satisfy other processes with the currently available resources. So, in this way if we can satisfy all the processes that means, the system is deadlock free. So, this is the basic idea that we have in this safety algorithm.

(Refer Slide Time: 09:34)



Example of Safety Algorithm

- 5 processes -- P_0 ..... P_4;
- 3 resource types:
  A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0:

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

Now this safety algorithm, we can have an example to before we proceed further. Suppose, we have got 5 processes P 0 to P 4 and this there are 3 types of resources: A has got 10 instances, B has got 5 instances and C has got 7 instances. So, this is the maximum thing, maximum resource that we have.

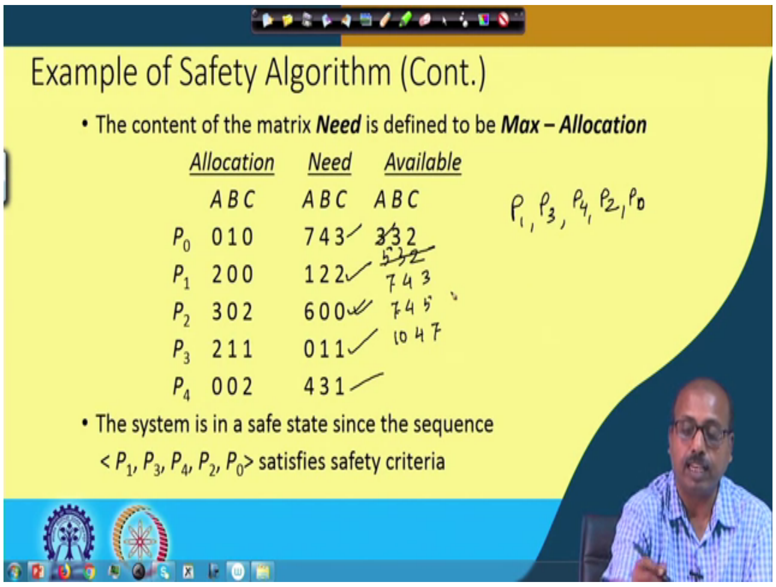
So, basically if we go back to the algorithm, then here this one this away this; so, that is the currently available resources. So, initially the available array will have all these. Now suppose at some point of time T naught, so, this is the situation. The Allocation array, so, for process P 0, it has been allocated no instance of a resource A, 1 instance of resource B and no instance of resource C.

Similarly, process P 1 it has been allocated 2 instances of process of resource A, 0 and 0 of B and C. So, like that P 2 is having 3 of A and 2 of C like that and this is the maximum requirement. So, this has been declared by the individual processes. So, process 0 has declared that it will require at most 7 A, 5 B and 3 C; P 1 will require at most 3 A, 2 B, 2 C; P 2 will require at most 9 A, 0 B, 2 C so like that.

So Available at present so, there are a 10 instances of A out of that you see 2 plus 3 plus 2. So, 7 instances are already in use. So, 3 instances are available. Similarly, for B 5 instances are maximum. So, we have got 1 plus 1, two already in use, so three are left. And C has got 7 instances out of that 5 are already in use so 2 are available. So, this is the current availability.

Now at this situation we have to judge whether the system is safe or not. So, system will be safe if by using this 3 3 2 of A B C we can satisfy any of these processes to completion and then that with that process will release the resources with that we will proceed with the other processes.

(Refer Slide Time: 11:52)



Example of Safety Algorithm (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3 ✓	3 3 2
P_1	2 0 0	1 2 2 ✓	~~5 3 2~~
			7 4 3
P_2	3 0 2	6 0 0 ✓	7 4 5
P_3	2 1 1	0 1 1 ✓	10 4 7
P_4	0 0 2	4 3 1 ✓	

P_1, P_3, P_4, P_2, P_0

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0 >$ satisfies safety criteria

So, let us see how it goes. So, first we determine the Need array so, which is Max minus Allocation. So, Max minus Allocation if we do so, this is 7 5 3 minus 0 1 0. So, it is 7 4 3. So, that is 7 4 3. For P 2 it is 3 2 2 minus 2 0 0, so it is 1 2. So, P 2 is 1 2 2 so, this is the requirement. So, P 0 can ask for at most at most 7 more instances of A, 4 more instances of B and 3 more instances of C, before releasing the already held one resource type B.

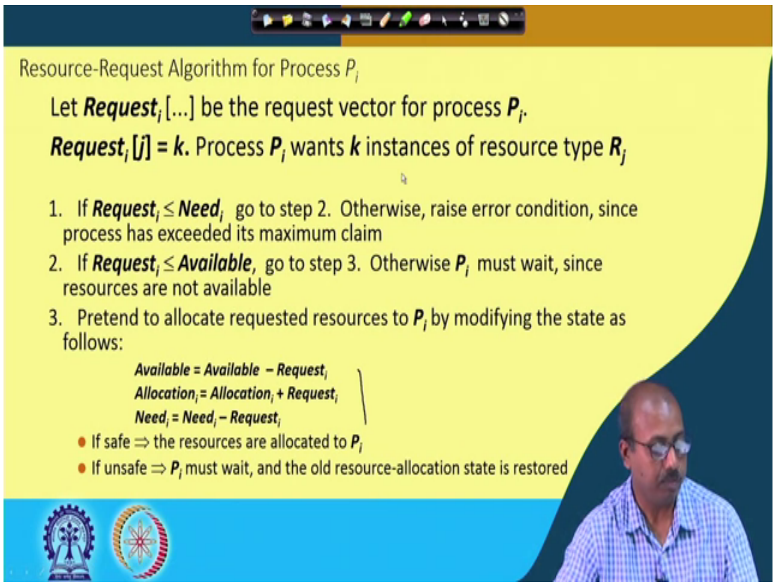
Similarly, P 2 without releasing the resource to a 2 instances of A it can ask for 1 2 2 of A B C like that. So, with that so this is the current availability so, can we satisfy 3 3 2 can we satisfy any of these needs ok? So, you see this P 1; 1 one requires 1 2 2 so with that these with these 3 3 2 availability so, this one can be satisfied. So, we can say that this we can satisfy this one. So, P 1 is done. So, once P 1 is done so this whatever is currently allocated to P 1, so, that will also be available. So, this 3 3 2 it will become this two instances will be available. So, it will become 5 3 2 ok. This two instances will be available.

With this 5 3 2 we can satisfy this P 3. So after P 1 is over so we can give resources to P 3, so that P 3 will complete and this 2 1 1 will be returned by that. So, 2 1 1 is returned so this availability will become 7 4 3. With this 7 4 3 being available, so, we can satisfy this P 4. P 4 is satisfied after this; so, with that this 0 0 2 will be available. So, this will become 7 4 5. Available array will become 7. Actually this Available is copied into Work if you remember. So, that Work array will get updated to 7 4 5. So, this is also done.

Now, after that with this 7 4 5 we can satisfy this P 2. So, 6 0 0, so this one can be satisfied. So, P 2 can be satisfied and this P 2 this after P 2 is over so, this resources will be available. So, you can say it is 10 4 7 that will be available and with this 7 4 10 4 7 we can satisfy P 1 this 7 4 3. So, P 1 can be satisfied. So, P 1 is satisfied at the end. So, at this at this point you see that all the sorry P 0 can be satisfied so this is P 0.

So, in this way if you see this all this processes could be satisfied. So, all the for all the processes that Finish array entry will become equal to true. So, this array will this algorithm will terminate declaring that this current system is in this is a safe situation. So, current situation is a safe situation. Similarly, if we cannot find any sequence of processes by which all the process maximum requir ements could be satisfied then we will say that the system is in unsafe state. So, we will proceed with this and let us see how this algorithm can be formulated.

(Refer Slide Time: 15:26)



Resource-Request Algorithm for Process P_i

Let $\textbf{\textit{Request}}_i$ [...] be the request vector for process $\boldsymbol{P_i}$.

$\textbf{\textit{Request}}_i[j] = \boldsymbol{k}$. Process $\boldsymbol{P_i}$ wants $\boldsymbol{k}$ instances of resource type $\boldsymbol{R_j}$

1. If $\textit{Request}_i \leq \textit{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $\textit{Request}_i \leq \textit{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\textit{Available} = \textit{Available} - \textit{Request}_i$$
$$\textit{Allocation}_i = \textit{Allocation}_i + \textit{Request}_i$$
$$\textit{Need}_i = \textit{Need}_i - \textit{Request}_i$$

- If safe $\Rightarrow$ the resources are allocated to P_i
- If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored

So, the Resource-Request algorithm for process P i. So, we do it like this so let Request i be the request vector for process i. So, every process makes a request. So, P i process makes a request and Request i, j equal to k, if P i has asked for k instances of resource type R j.

So, if Request i is less or equal Need i, then only we go to step 2 because if resource i Request i is greater than Need i, that means, there is there is an error. This is an error condition because the process has asked more that it initially declared. So, that is a that should not be the case.

So, Request i less than less or equal Need i, so, this is the condition to be satisfied. Otherwise, raise an error condition the process has exceeded its maximum claim. If Request i is less than Available, then we if Request is more than Available, then naturally the resources are not available, so, process anyway has to wait. So, if Request i is greater than Available, then process has to wait because resources are not available.

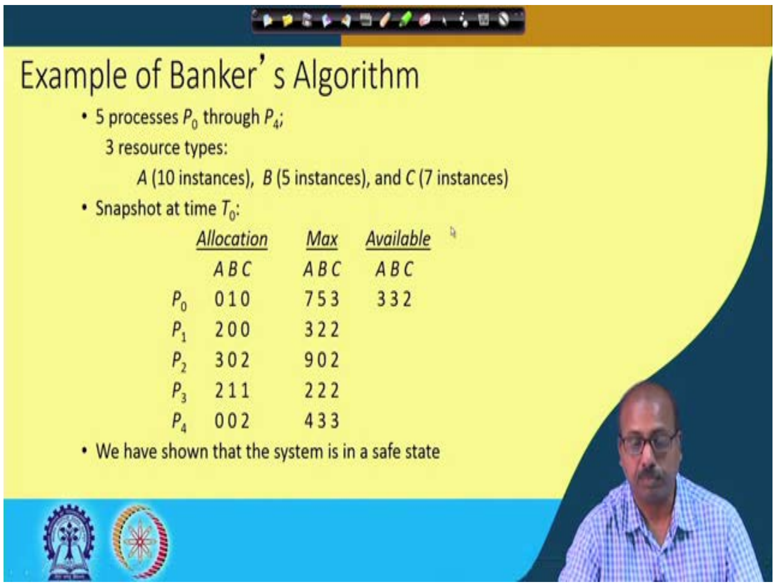
If resources Request is less than Available less or equal Available, then we come to step number 3, we assume or pretend that we have allocated the resources to P i by modifying the state like this; from the available we remove the resources asked by the Request i, so, available array is updated.

Allocation array is updated by allocation with Allocation i we update with this Request i as if all the requested resources have been allocated to P i. So, Allocation i gets updated. And Need i is naturally that is decremented by Request i because Need i is Available minus Allocation. So, that is that Need that is the Maximum minus Allocation. So, that way this Need i is updated by Need i minus Request i.

Now, after that we run the safety algorithm that we have discussed previously and after running the algorithm if we find that it is safe to allocate this the system is in safe state that means, it is safe to allocate the resources requested by process P i. So, in that case the resource the resources requested will be allocated. Otherwise, the if the algorithm declares that the system will go to an unsafe state, then the process P i must wait. So, you see here even if the resources are available only because there is a possibility of a leading to a deadlock situation so, this resources will not be allocated.

So, as old resource allocation state will be restored so, whatever updation we made, this temporary updation that we did for this resource available and these arrays so, they will be; they will be undone this resource Allocation step. So, Available will be Available plus Request I; Allocation i will be Allocation i minus Request I, and Need i will be Need i plus Request i. So, those updations will be done. It is not written here explicitly. So, then we will try with the next request from any other process. So, this way this algorithm continues and it gets this resource allocation done.

(Refer Slide Time: 18:43)



Example of Banker's Algorithm

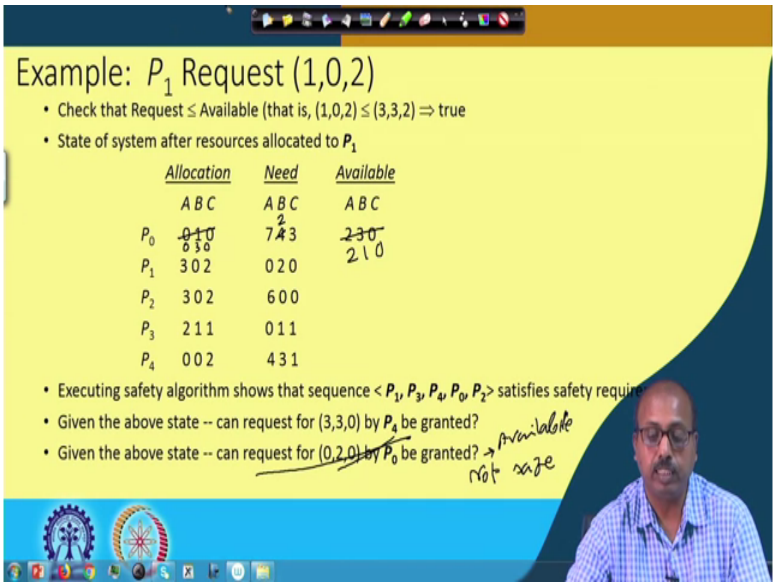
- 5 processes P_0 through P_4;
  3 resource types:
  A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- We have shown that the system is in a safe state

So, it will have a look at this banker’s algorithm so, again we come back to an example that we did preciously. So, suppose there are 5 processes P 0 to P 4 and there are 3 resource types: A of 10 instances, B of 5 instances and C of 7 instances. Snapshot at time T 0 is like this and we have previously analyzed that this state is a safe state. So, system is in safe state. So, there is no possibility of going into a deadlock from this particular state.

(Refer Slide Time: 19:19)



Example: P_1 Request (1,0,2)

- Check that Request ≤ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true
- State of system after resources allocated to P_1

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	~~0 1 0~~ 0 3 0	7 ~~4~~ 2 3	~~2 3 0~~ 2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence $< P_1, P_3, P_4, P_0, P_2>$ satisfies safety requi
- Given the above state -- can request for (3,3,0) by P_4 be granted?
- Given the above state -- can request for (0,2,0) ~~by~~ P_0 be granted? → Available not safe

Now, suppose; now suppose process P 1 makes a request for 1, 0, 2. So, first of all the availability is availability was a 3 3 2. So, the first thing we check whether Request i is less than Need i. So, Request i less than Need i, process P 1 is asking for maximum is 3 2 2 so that check is done, so, it is ok. Then Request is less than Available; the 1 0 2 is less than 3 3 2 so, that is also correct.

So, now we will be trying the trying out that safety algorithm and for that we will be looking into state of the system after resources allocated to P 1. So, we will pretend that this allocation has been done to P 1 and then we try to analyze the system. So, it is 0 1 0; so, this P 1 this P 1 was has requested 1 0 2. So, as a result this allocation is done. So, P 1 was having previously 2 0 0 with that 1 0 2 it becomes 3 0 2 and this Need becomes it is 3 2 2 minus 3 0 2, so, it is 0 2 0. So, this is the Need. And currently Available A B C available was 3 3 2 from there this 1 0 2 is subtracted so it becomes 2 3 0.

Now, with this we have to see whether the system is in safe state or not. So, if we try to run safety algorithm on this then will with this 2 3 0, you can see that first P 1 can be satisfied because P 1 if i give it 0 2 0 resources the 2 instances of B, then P 1 will finish off. And as a result this Available array will become this 3 0 2 will be returned, so, it will become 5 3 2. So, it will become 5 3 2.

So, with this 5 3 2, P 3 can be satisfied. If P 3 is satisfied then this 2 1 1 will be returned. So, this will become 7 then 4 then 3 ok. So, with that this P 4 can be satisfied, this 4 3 1 so this becomes 11 sorry. So, this becomes 7 4 5. So, after P 4 we can satisfy P 0 and this 0 1 0 will now be available. So, this becomes 7 5 5. So, with that we can satisfy P 2 and this 3 0 2 can be satisfied. So, that it becomes 10 5 7. So, you see that this one can be satisfied. So, this request can be satisfied. So, this is in a safe state.

So, at this point of time, so we can see that this particular request by P 1 so, if this request is granted then the system will be in safe state, so, we can allocate the resources to the process P 1. So, this particular request can be satisfied and this is a safe allocation. So, this resource will be allocated.

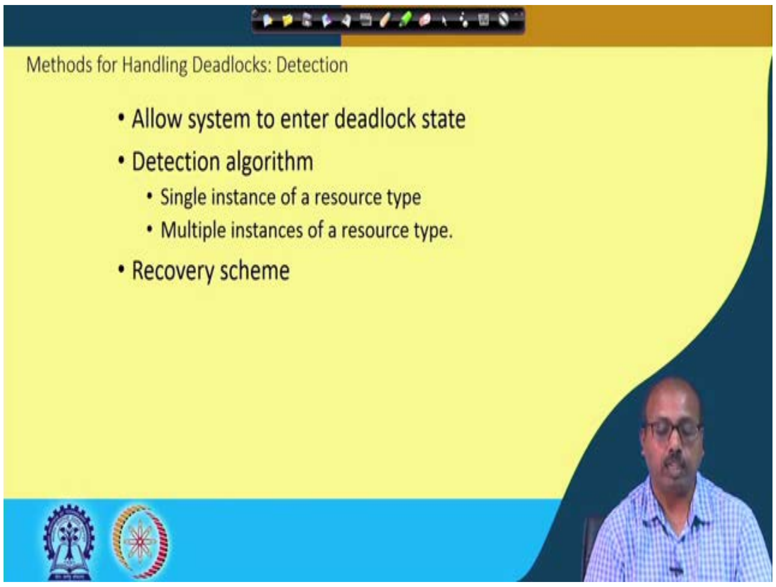
Now, given this state so given the suppose we have allocated 1 0 2 to process P 1 and now process P 4 has requested for 3 3 0. So, P 4 has requested for 3 3 0; so this one if I if I try to see whether this P 4 can be granted or not so, this 3 3 0 if it is there then so it is becomes the Allocation becomes 3 3 2 and this 3 3 0. Of course, the this 3 3 0 cannot be allocated because you see this Available is 2 3 0, so, it will straightway stop at that point because 3 3 0 is not available. So, that way this cannot be done, this cannot be granted.

Now, however, this 0 2 0; suppose P 0 requests for 0 2 0. So, can so this is a not possible because not Available; resource is not Available. Now, what about this one say 0 2 0? So, 0 2 0 if we give P 0, so, this becomes Allocation becomes 0 3 0 ok. So, this maximum requirement will become 7 2 3 and then this 2 3 0, so this will change to 2 1 0. Now, with this 2 1 0 requirement so can we satisfy any of the processes? Like P 0 requires 7 2 3 so it cannot be satisfied; P 1 requires 0 2 0 so it cannot be satisfied; P 2 requires 6 0 0 cannot be satisfied; P 3 requires 0 1 1 so cannot be satisfied and then this P 4 requires 4 3 1, so, none of the processes can be completed.

So, none of them will be returning there is a possibility that none of the processes will return resources before releasing all other resources. So, this is so this one so this particular point so, the resources are available, but the allocation is not safe; Available, but not safe. So, this is not safe.

So, this is a situation that banker’s algorithm it will stop the Allocation. So, this Allocation will also be stopped though the resources are Available. So, this way this banker’s algorithm it does a check that whether there is any possibility of going to a deadlock situation and if there is a possibility, then the resources will not be allocated.

(Refer Slide Time: 25:17)



Methods for Handling Deadlocks: Detection

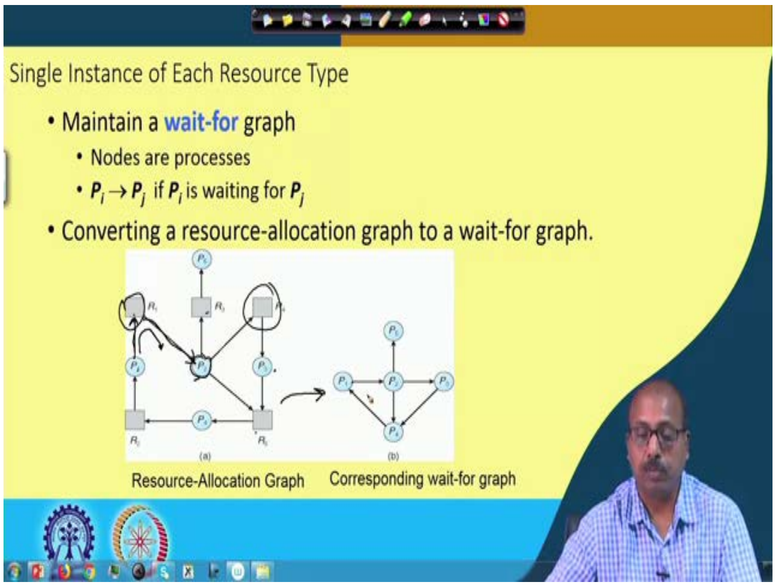
- Allow system to enter deadlock state
- Detection algorithm
  - Single instance of a resource type
  - Multiple instances of a resource type.
- Recovery scheme

So, next we will be looking into the deadlock detection and recovery techniques. So, there are this will allow the system to enter into deadlock state and then some detection algorithm will run. And the detection algorithm will determine whether there is any deadlock or not and if there is a deadlock it will try to recover from the deadlock situation.

So, because this is important because in many cases what happens is that there is there the possibility of occurrence of deadlock is very less. So, as a result when the system's throughput becomes pretty low even if the degree of multiprogramming is high, so we can there is a possibility of deadlock in the system.

So, then this deadlock detection algorithm can run, so it may try to see whether there is really a deadlock in the system or not and if there is a deadlock then it will try to recover from the deadlock. So, recovery means some of the processes that is deadlocked so, it will be; it will be taken off from the say from the system. So, it will be restarted or so and all the resources that are held by that process so, that will be claimed back. So, this is the; this is the way we will do recovery. So, we will come to this recovery scheme and all.

(Refer Slide Time: 26:37)



Single Instance of Each Resource Type

- Maintain a wait-for graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Converting a resource-allocation graph to a wait-for graph.

So, first one for single instance of each resource type. So a for single instance resource type, so this is similar to the graph based algorithm that we have seen for this avoidance of deadlock. So, here we may remember we construct something called a wait-for graph.

So, suppose this is the resource allocation graph that we have. So, in this resource allocation graph so you see that process P 1 it is holding the resource R 2 and requesting for R 1; P 2 is holding R 1 requesting from R 5; P 2 is also holding R 3 sorry P 2 is also asking for R 3 and it is R 3 is currently held by P 5. Similarly, P 2 is also asking for R 4. So, P 2 is asking for R 3, R 4 and R 5 while it is holding R 1. P 3 it is holding R 4 and asking for R 5; P 5 is not asking for anything, but it is holding R 3 and P 4 is holding R 5 and asking for R 2. So, this is the situation that we have.

So, this is the resource allocation graph. Now, from this resource allocation graph we construct another graph which is called wait-for graph ok. So, wait-for graph so this does not have resources, it just have processes in it. So, which process is waiting for which process. Like you see, that P 2 is waiting for the resource R 1 so, P sorry, P 1 is waiting for the resource R 1 which is currently held by P 2. So, in some sense we can say that R 1 will be released by P 2 only after it has finished using. So, P 1 is actually waiting P 2 to release R 1. So, you can say P 1 is waiting for P 2.

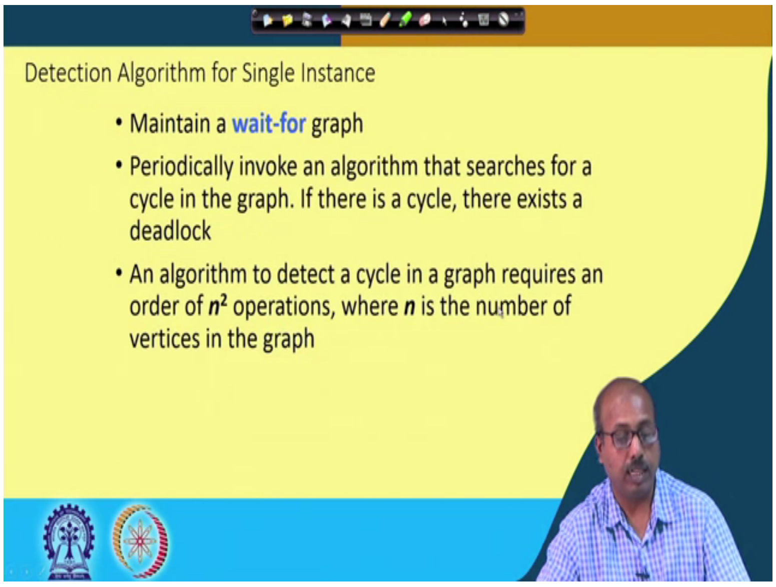
Similarly P 2 is waiting for the resource R 4 and R 4 is currently held by P 3. So, P 2 is actually waiting for P 3. Similarly, P 2 is also waiting for P 5 because resources at R 3 is with P 5. So, P 2 is waiting for P 5 and P 2 is waiting for P 4 because the resource R 5 is currently held by P 4. So, it is waiting for P 4 and P 4 is waiting for P 1 because this resource 2 is held by P 1 and then once P 1 releases it then only P 4 can get it. So, P 4 is waiting for P 1.

So, this way we can construct a wait-for graph and so this in this wait-for graph we do not have any resource node only the processes are the nodes and there is an edge from process P i to process P j, if P i is waiting for process P j. So, this way we construct the wait for graph. So, this strategy is very simple.

So, you start from a process and if you find an edge so going out of that process node and it whichever resource reaches from the resource we again take the edge going out and then ultimately reach a process which is holding it. So, that way if we start with P 1 and search by this path the so you will reduce the process P 2. So, accordingly we can add this edge.

So, this way we can formulate this wait-for graph by looking at each processes and then going through the edges that are going out of the process node. So, this way this resource allocation, this wait-for graph is constructed and once this wait-for graph is constructed so, we can run a cycle detection algorithm.

(Refer Slide Time: 30:12)



Detection Algorithm for Single Instance

- Maintain a wait-for graph
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

So, we maintain a wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, then there exist a deadlock. So, basically this wait-for graph is always maintained in the system and whenever the system throughput is low then the ways may decide that there is a possibility of deadlock so, let me run the deadlock detection algorithm once. So, this is a cycle detection algorithm and cycle detection you know that it uses this some sort of depth-first search algorithm and using that algorithm so, it can try to detect a cycle in the graph and once this if a cycle is detected then there is a deadlock.

So, algorithm detect a cycle in a graph requires an order of n square operation, where n is the number of vertices in the graph. So, that way it is a costly operation. So, you can understand that if there are a large number of processes in the system then this detection deadlock detection will take time. So, it is a costly operation. So, until and unless it is very much necessary, so you should not go for this deadlock detection type of algorithm. But, some systems use it because we have to we do not; we do not use this avoidance or prevention mechanisms and we use this detection technique.

So, we will continue with this in the next class when we discuss on the multi instance situation.