

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

Lecture - 38
Deadlock (Contd.)

So, next we will be looking into the avoidance policy, Deadlock avoidance policies.

(Refer Slide Time: 00:26)

Deadlock Avoidance

- Ensure that the system will *never* enter a deadlock state
- Requires that the system to have some additional *a priori* information available on possible resource requests.
 - Simplest and most useful model requires that each process declare the **maximum number** of resources of each type that it may need
 - The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
 - Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

	R ₁	R ₂	R ₃
P ₁	3	2	5
P ₂	1	0	4

So, prevention is fine; so prevention is actually trying to do something so that we know that this deadlock will not occur. But, they are a bit synthetic solutions we should say because they are imposing different type of restrictions on the resource type and all. And, it is difficult for a write for writing a process by an application programmer so that these resources are requested in particular sequence.

So, unless so what will happen is that the programmers they will ask for all the all the resources that they need at the very beginning to just to avoid the possibility that I may not be granted decode resources later. So, they will try to put all the resources at the beginning, so that they can as play it safely and release only at the end. So, that actually effects the system resource utilization factor. So, not a very good solution always.

So, next we will be looking to these avoidance policies. So, this deadlock avoidance policies they ensure that the system will never enter into a deadlock state. So, somehow

so, we have we have to guarantee these thing the by means of this resource allocation policies and all that the system will never enter into a deadlock state.

It requires that the system to have some additional a priori information available on possible resource requests. So, it assumes that for every process we know what are the maximum resource requirements of each types of resource for a process. So, for; so maybe I have got I have got say three resource types R 1, R 2 and R 3 in my system and for process 1 may be it so, maximum requirement is 3 instances of R 1, 2 instances of R 2, 5 instances of R 3.

Similarly, for P 2 may be it requires 1 instance of R 2, it does not require any instance of R 1 instance of R 1, no instance of R 2 and 4 instances of R 3. So, these are all the maximum requirements for the processes. Somehow this has to be known and naturally if the system follows this particular policy, then the programmer may be asked to provide at the very beginning what is the maximum resource requirement of different types of resources that is there in the system.

So, simplest and the most useful model requires that each process declare the maximum number of resources of each type that it may need. So, 3 for example, P 1 requires at most 3 instances of R 1 does not mean that P 1 require all of them simultaneously, so all it will require all the three at together. It just tells that I may need at most 3 of R 1, 2 of R 2, 5 of R 5 R 3 and during execution so, it will put the individual requests, but the total request will never be exceeding 3 for R 1, 2 for R 2, and 5 for R 3 simultaneously.

So, this that; so the process has to declare the maximum number of resources of each type that it may need and the deadlock-avoidance algorithm it will dynamically check whether the resource allocation state it will dynamically examine the resource allocation state to ensure that there can never be a circular-wait condition. So, it will check whether there can be a circular-wait or not and they.

So, resource-allocation state is defined by the number of available and allocated resources and the maximum demand for each process. So, this resource-allocation state so, what are what is the currently available and allocated resources that least and the maximum requirement for the processes. So, that will define this resource allocation state.

(Refer Slide Time: 04:26)

Safe State

- System is in **safe state** if there exists a sequence $\langle P_1, P_2, \dots, P_n \rangle$ of ALL the processes in the systems such that for each P_i the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j with $j < i$.
Handwritten notes: R_1, R_2, R_3 and $P_1, P_2, \dots, P_{i-1}, P_{i+1}$
- That is:
 - If P_i resource needs are not immediately available, then P_i can wait until all processes P_j ($j < i$) have finished executing.
 - When they have finished executing they release all their resources and then P_i can obtain the needed resources, execute, return allocated resources, and terminate
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

So, based on this we can formulate some deadlock-avoidance policy. We will define some state to be a safe state and some other state to be unsafe state.

(Refer Slide Time: 04:32)

Basic Facts

- If a system is in safe state \Rightarrow no deadlocks
- If a system is in unsafe state \Rightarrow possibility of deadlock

Handwritten notes: $P_i: 3R_1, 2R_2$ and $1R_1, 1R_2$ / $2R_1, 1R_2$

So, this is a diagram, so that tells if this is the total state space that we have; so some of the states are safe state and some of the states are deadlock deadlocked state. So safe states are such that if a system is in safe state it can never go to deadlock directly. So if so this is the total set of states now it says that so, this is a state here and this is a

deadlock state, then we will never have a transition from this to this. So this is never there.

But, there are some states which are unsafe in nature. So there may be transition from safe state to unsafe state and from the unsafe state there is a possibility that it will make a transition to a deadlock state. Of course it is very much possible for that from an unsafe state it again comes back to safe state. But, what we will do we will not allow this type of transitions at all because if the system goes into an unsafe state then there is a possibility that it may go into a deadlock state.

Knowing fully well that in reality in actuality the system may not be making this transition, but it will make a transition coming back to a safe state. But, just to ensure that we will never go to a deadlock state, we will stop this type of transition which will take the system from safe state to some unsafe state .

So, we will elaborate on this slowly so if we just go back define what is a safe state; a system is in a safe state if there exists a sequence P_1, P_2, P_n of all the processes in the systems such that, for each P_i the resources that P_i can still request can be satisfied by currently available resources plus resources held by all the P_j where j less than i . So, let us try to understand what does it mean.

So, at any point of time I have got a number of processes in the system P_1, P_2 , up to P_n and I have got the resources suppose I have got say three types of resources say R_1, R_2 and R_3 . Now, this P_1 to P_n they have got some requirement for R_1, R_2, R_3 . So, they have they are having , so P_1 is holding some amount of R_1 it for its completion it may need some more of R_1 . So that is depicted by the maximum requirement that has been declared by the process P_1 at the beginning.

So, process P_1 for its completion which how many more instances of R_1, R_2 and R_3 are required that is known. Similarly, for process P_2 what is what are the extra R_1, R_2 and R_3 instances needed, so that is known. Now if you can somehow ensure that with the currently available free instances of R_1, R_2, R_3 and so, you with the currently available instances of R_1, R_2, R_3 .

So, if you can satisfy the requirement of P_1 , then I can say that P_1 complete. After that this currently available resources plus the resources held by P_1 they will be released

with that if I am able to satisfy P_2 and so P_2 will finish of and thus the process continue. So, I just then I take all the resources which are free after P_2 has finished, then with that if I am able to finish of the next process.

So, among this processes P_1 to P_n , so if you can find a sequence in which all this processes requests can be satisfied by this requirement. That is so, when I am talking about say P_i so, P_j so, for completion of P_j the resources which are free plus the resources held by sorry for the process P_i ; for the process P_i , so I can say that. So, for so, this for process P_i I have got the resources which are currently available plus the resources which are held by the processes before P_i in this chain.

So, I have got this P_1, P_2, P_i say P_{i-1}, P_i, P_{i+1} etcetera. Now, when I come to this process P_i then all the resources which were available at this point plus all the resource held by this processes so, they are all free now. So, with that if we can satisfy P_i then I can say that P_i can be completed. So, the system will be safe state if I can complete all these processes, so P_1 to P_n . So, if I can complete all the processes in this way when the system is not in a deadlocked state. So, this is a this cannot lead to a deadlock state, so this is a safe state.

So, this actually what it means is that if P_i resource needs are not immediately available, then P_i can wait until all processes P_j, j less than i have finished executing and when they have finished their executing. So, they will release all their resources and then P_i can get all these resources, then execute and return the allocated resources and terminate and when P_i terminates the P_{i+1} can get the needed resources and so on. So, this way it can continue.

So, if we can find a sequence of processes by which all the requirements all the of the processes can be satisfied, then we say that the system is in a safe state. So, with that we can look in to this diagram once more. When the system is in safe state, so, if the system is in any of these state in this subset then all the requests for all the processes can be satisfied always.

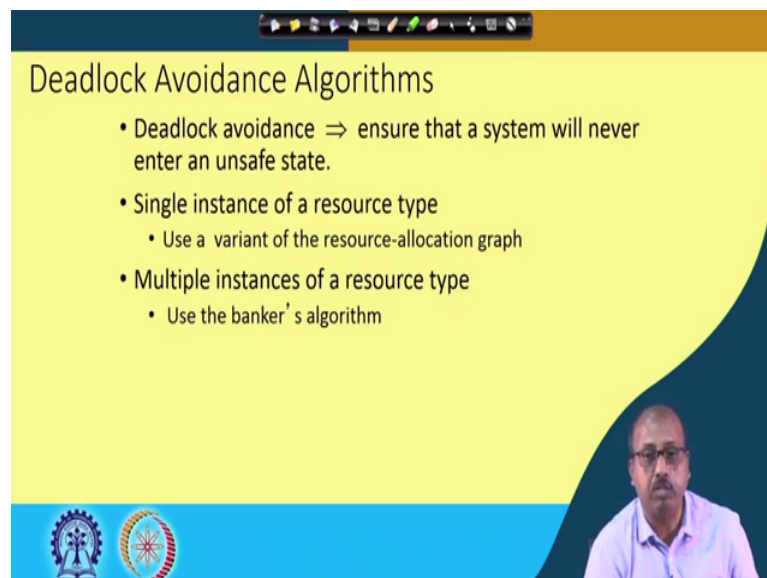
But, if the sub system is in unsafe system, then all the requests cannot be satisfied. Though may be the case that the system the processes they make request for their maximum resources as a result the system goes into deadlock state. But, at the same time

we must understand that a process may not request further resources, but rather release a few resources.

Maybe it may so happen that a process P₁ it requires say 3 of R₁ and 2 of R₂. And, when I am looking into the system code the process code at some point of time may be I am I am at this point, so it has at this point it has got 1 of R₁ and 1 of R₂ with it. So, it can ask for 2 more R₁s and 1 more R₂ before releasing this resource and that may lead to deadlock, but we must understand that may be in between so, it releases this resources also. So, it releases some more resource; so it releases say 1 R₁ and after some time it releases 1 R₂, then it request for 2 R₁, then it request for 1 R₂. So, it may happen like this.

So, it see you see that so, when it releases the resource may be from this unsafe state it can come back to safe state that can happen. But, there is a possibility that without releasing resource the process will request for further resource as a result it the system may go into a deadlock state. So, we will try to avoid this we will try avoid putting the system in unsafe state so that there is potential to go to the deadlock state. So, this is what we will be trying out in our this deadlock avoidance policies.

(Refer Slide Time: 12:15)



The slide is titled "Deadlock Avoidance Algorithms" and features a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two circular logos: one with a gear and a figure, and another with a wheel and a figure. At the bottom right, there is a video inset showing a man with glasses and a light blue shirt speaking. The slide contains the following text:

- Deadlock avoidance \Rightarrow ensure that a system will never enter an unsafe state.
- Single instance of a resource type
 - Use a variant of the resource-allocation graph
- Multiple instances of a resource type
 - Use the banker's algorithm

Now, deadlock avoidance it will ensure that the system will never enter into an unsafe state. Single instance of a resource type, so use a variant of this resource allocation graph and based on that we can formulate and some avoidance policy. If there are multiple

instances of resources, then we can follow something called a banker's algorithm and this banker's algorithm. So, this will be ensuring that this deadlock can be avoided, will not go into this will not go into the unsafe states.

(Refer Slide Time: 12:45)

Resource-Allocation Graph Scheme

- Single instance of a resource type
- Each process must a priori claim maximum resource use
- Use a variant of the resource-allocation graph with claim edges.
- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j ; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system
- A cycle in the graph implies that the system is in unsafe state

So, resource allocation graph based policy. So this is applicable only when we have got single instance of every resource. So multi multiple instances are not there, every resource there is only a single instance. Then the same thing that each process must a priori claim maximum resource use, so the it may every process must tell of the which resources it needs. Since there is a there is always single instance, so telling about how many is of no issue here. So basically we tell what are the resources that are needed by this process.

And, use a variant of the resource allocation graph with something called a claim edge. So, claim edge; so this P_i to R_j it indicates that the process P_j . So, process P_i may request for resource R_j . So, this is this is not P_j , so this is P_i . So, process P_i may request for R_j and represent by a dashed line. So, what I was telling is that may be process P_1 it is maximum, so it requires the resources R_1 , R_2 and R_3 . So, that is declared by the process at the beginning at present may be process P_1 has got the R_1 allocated to it.

So, process P_1 has the potential to ask for the resources R_2 and R_3 . As a result so, we will put the claim edges to this resources R_2 and R_3 . So, we will put a claim edge for R

2 and we will put a claim edge for R 3. So, this is what is told here; so if we will put a claim edge from P_i to R_j indicating that the process may request for R_j and represented by a dashed line.

Claim edge converts to request edge when a process request a resource. So after sometime P_i actually requests for R_2 . So when it request for R_2 then this edge will become a solid edge ok. So as long it is a potential claim, so this is a dashed edge, but when the process request actually makes the request for the resource then it will be modified by a solid edge.

And, request edge will be converted to an assignment edge when the resource is allocated to the process. So, after some if the system allocates the resources to the process, then this edge will be converted into an allocation edge by removing; so by if this edge will be converted into an allocation edge by putting the arrow on this side. So, this resource is now allocated to process P_1 .

So, when a resource is requested when a resource is released by the process, then the assignment edge is reconverts into a claim edge. So after some time process P_1 may might have released this resource R_2 , but since till the lifetime of P_1 ends. So, it can again ask for resource R_2 . So, in that case so, we will be converting this this assignment edge into again a claim edge. So, this is how this whole thing will work. So, when a resource is released by a process the assignment edge is converted into a claim edge.

And, resources must be claimed a priori in a in the system, so this they must be required at the beginning. A cycle in the graph implies that the system is in unsafe state. So, if you construct a graph like this having this claim edge, request edge and grant edge, allocation edge then the situation that we get from the graph, so that is the if there is a cycle in this graph then it is in unsafe state.

(Refer Slide Time: 16:48)

Resource-Allocation Graph with claim edges

- P1 is holding resource R1 and has a claim on R2
- P2 is requesting R1 and has a claim on R2
- No cycle. So system is in a safe state.

$P_1: R_1, R_2$
 $P_2: R_1, R_2$

The diagram illustrates a Resource-Allocation Graph (RAG) with claim edges. It features two processes, P1 and P2, and two resources, R1 and R2. P1 is holding resource R1 (indicated by a solid arrow from R1 to P1) and has a claim on R2 (indicated by a dashed arrow from P1 to R2). P2 is requesting R1 (indicated by a dashed arrow from P2 to R1) and has a claim on R2 (indicated by a dashed arrow from P2 to R2). The graph shows no cycle, indicating the system is in a safe state.

So, this is what we will try to make; so this is like this. So, resource allocation graph with claim edges. So, P 1 is holding, so, there are two resources R 1 and R 2 in the system. So, P 1 can ask for both R 1 and R 2; so we have got two processes in the system P 1 and P 2. So, P 1 can request for the resources R 1 and R 2 and P 2 can P 2 also can request for the resources R 1 and R 2.

Now, P 1 is holding resource R 1, so we have got an allocation edge from R 1 to P 1 and it has a claim on R 2. So, since P 1 has potential to ask for R 2, so there is a claim edge from P 1 to R 2. Now, P 2 is requesting for R 1 and it has got a claim on R 2. So, it is a it is at present P 2 is requesting for R 1 and it is it has got a claim for R 2; so this is the situation.

Now, there is no cycle in this graph. So naturally this is this system is safe. So, why the system is safe? You can understand that if even if say P 2 in this situation so, if this P 1 is given R 2. So, if P 1 and P 2 both of them come up with their requests, so I can give this R 2 to P 1. So, P 1 will finish off and then this R 1 and R 2 be given to P 2 and P 2 will also finish off. So, nonexistence of cycle in this particular resource allocation graph with claim edges so that means, the system is in safe state. So, it cannot go to a deadlock state.

(Refer Slide Time: 18:27)

Example of a Resource Allocation Graph

- The graph of last slide with a claim edge from P2 to R2 is changing to an assignment edge.
- There is a cycle in the graph → unsafe state.
- Is there a deadlock?

The diagram shows a Resource Allocation Graph (RAG) with two processes, P₁ and P₂, and two resource types, R₁ and R₂. P₁ is represented by a blue circle, and P₂ is a blue circle. R₁ and R₂ are represented by grey rectangles. There are two solid arrows pointing from R₁ to P₁ and two solid arrows pointing from R₂ to P₂. A dashed arrow points from P₁ to R₂, representing a claim edge. A circular arrow indicates a cycle: P₁ -> R₁ -> P₂ -> R₂ -> P₁.

Now, going further; so consider the case where claim edge from P 2 to R 2 is changing to an assignment stage. Suppose, we do it like this that P 2 has requested for this R 2; P 2 has requested for this R 2 and it has granted to it; so R 2 was free. So, when this edge was not there; so, if we look in to the previous slide, look into the previous slide then R 1 is given to P 1 and R 2 is free.

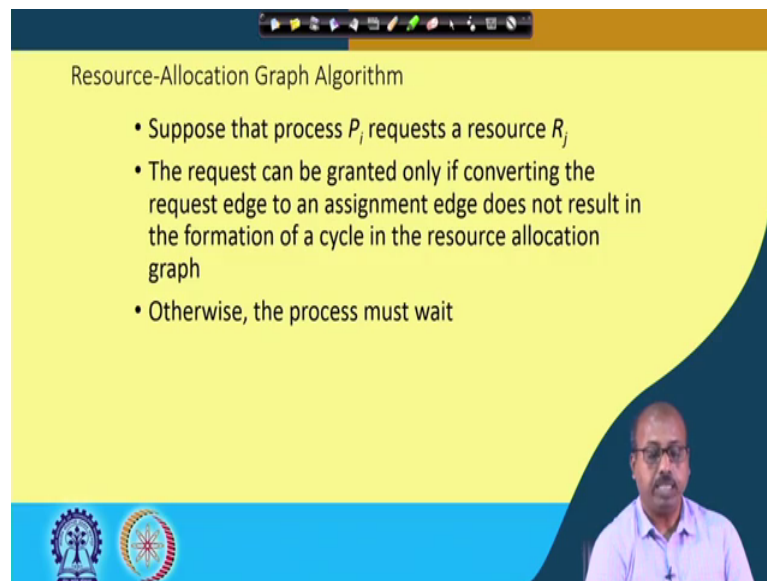
Now, suppose P 2 requests for R 2, so this claim edge converted into a request edge and the system allocates this resource R 2 to P 2. Then what happens? The situation becomes something like this. This R 2 has R 2 is allocated to P 2, so P 2 holds a request for R 1 and there is a claim that P 1 will request for R 2.

Now, the situation is that there is a cycle in this graph, now you see that I cannot give R 1 to P 2; so P 2 cannot proceed. So, this R 1 is currently with P 1. So, P 1 without releasing R 1 may request for R 2 because there is a claim edge here. So, this claim edge at any point of time may get converted into a request edge. So, as a result P 1 may request for R 2 and there will be a cycle coming into this graph; so there will be a cycle coming into this graph. So, that is existing here.

And, if this claim is converted to a request then there will be a deadlock. So, this is an unsafe state. So, this is not a deadlock state at present because till now P 1 has not requested for R 2. So, this is the potential request. So, if P 1 does not request for R 2 at any point of time if P 1 before P 1 instead of P 1 requesting for R 2 if P 1 releases R 1,

then of course, the system will go back to the safe state, because now P 2 can be given both R 1 and R 2 P 2 will finish off then P 1 comes out some other time it request for R 2 then it is getting R 2, so that is possible. But, this is an unsafe state ok. So, it can go to a deadlock state or it can come back to a safe state; so both are possible. Now, there is no deadlock at this point, but it is an unsafe state. So, we can proceed with this.

(Refer Slide Time: 20:51)



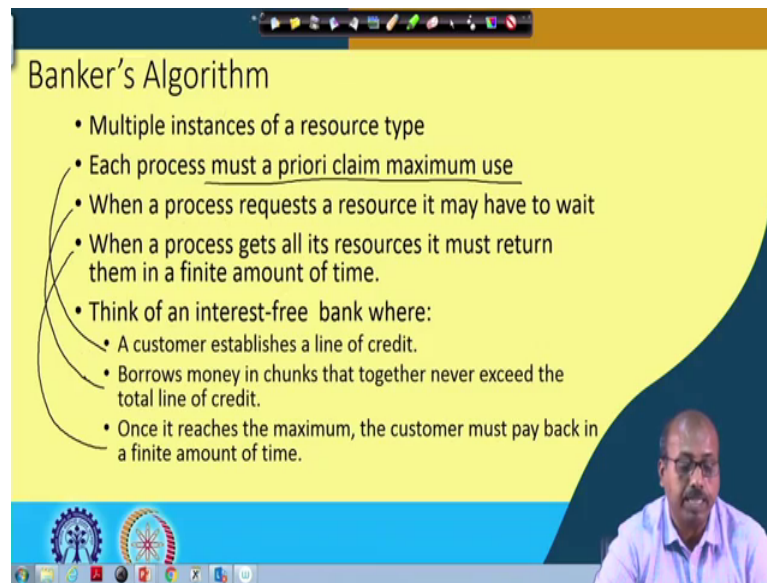
Resource-Allocation Graph Algorithm

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph
- Otherwise, the process must wait

So, resource allocation graph based algorithm is like this suppose that process P_i request for a resource R_j . The request can be granted only if converting the resource edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph; otherwise, the process must wait.

So, whenever a process is requesting for a resource, so if you convert the resource edge to an assignment edge and see that there is a there is a cycle in the allocation graph then there is a potential for deadlock. So, we have to we have to be careful; so we have that the it is going to an unsafe state. So, we have to stop that allocation, though resource is available, so it will not be allocated just to avoid that case that it it may go to a deadlock state.

(Refer Slide Time: 21:43)



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.

The next algorithm that we will look into is known as banker's algorithm. So, this banker's algorithm, so there are multiple instances of a resource type. So, if there are multiple instances then that cycle detection algorithm does not work. So, we have to go for some more complex algorithm and one such algorithm is the banker's algorithm.

So, each process must a priori claim maximum usage that is fixed as I said that a process must tell how many instances of different resource types that it may needs in its entire lifetime. So, that way each process must a priori claim the maximum usage. When a process requests a resource it may have to wait and when a process gets all its resources it must return them in a finite amount of time.

So, if process request for resource it may have to wait because the resource may not be granted and when the process gets all its resources, then it should not be that it holds the resources infinitely. So, it will use those resources and release all those resources, so that should be done. So, process execution time is finite.

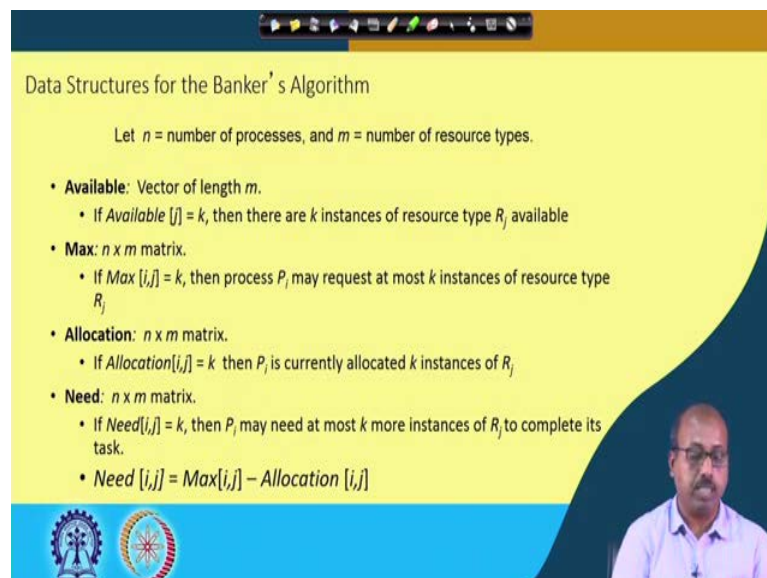
So, this is like an interest-free bank situation, where a customer establishes a line of credit and borrows money in chunks that together never exceed the total line of credit. So, it asks for money now that does not exceed the maximum limit that it has. Once it reaches the maximum the customer must pay back in a finite amount of time. Once it has got you reached the maximum value then the amount must be the entire amount must be returned within a finite amount of time.

So, this is that is why this algorithm is known as banker's algorithm. So, you if you look into the similarity between these two these statements here; so, each process must ah claim a priori the maximum usage. So, this is similar to a customer establishing a line of credit. So, what is the maximum credit that the customer may ask for so, that is there. And, then borrows money in chunks; so, this is basically the requesting for the resource.

So, this is when a process ah request a resource, so then it has to wait. So, that way it has to wait for that. So, here also the customer is asking in chunks. So, that is requesting for the resources and may have to wait because if the bank finds that it may violate the total amount of the money that the bank will have with it to satisfy all the customer. So, that may not be that may be violated.

So, and may be only if some other customer pays back the whole amount, then only this customer will be given the money. So, like that. And, when the process gets all its resources it must finish within finite time and release the resource. So, that is basically the third condition that when the amount reaches the maximum, then the customer must pay back in finite amount of time. So, that is why this algorithm is known as banker's algorithm because it mimics the behavior of a banking system.

(Refer Slide Time: 24:50)



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.
 - $Need[i,j] = Max[i,j] - Allocation[i,j]$

Now, the data structures that are used in banker's algorithm are like this, we have got n number of processes and m number of resource types. So, we have got an array Available, so it is a vector of length m . If Available j equal to k , then they are k instances

of resource type R_j available. Then we have got a Max n by m , so $Max_{i,j}$ equal to k ; that means, the process P_i request at most k instances of resource type R_j . So, this is basically the process i requesting for resource R_j what is the maximum number of instances that it can ask for.

Allocation is an again an n by m matrix and $Allocation_{i,j}$ is equal to k , then P_i is currently allocated k instances of resource j . So, if $Allocation_{i,j}$ equal to k , then P_i so, that it is currently allocated k instances of resource j . And, $Need$ is another array n by m . So, $Need_{i,j}$ equal to k , means P_i may need at most k more instances of R_j to complete its task.

So, $Need_{i,j}$ is basically a derived matrix, so this is $Max_{i,j}$ minus $Allocation_{i,j}$. So, the maximum requirement for individual processes have been mentioned in the Max array and $Allocation$ is the current allocation for the processes. So, if you subtract this $Allocation$ from Max , so that gives the $Need$ ok. So, $Need$ is basically a what is the extra amount of resources that the processes may ask for without releasing any further resource, so that is the $Need$ array.

(Refer Slide Time: 26:34)

Safety Algorithm

1. Let **Work** and **Finish** be vectors of length m and n , respectively. Initialize:
Work = Available
Finish [i] = false for $i = 0, 1, \dots, n - 1$
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.

The slide features a yellow background with a blue and white wave-like graphic on the right side. At the bottom left, there are two circular logos. At the bottom right, there is a video inset showing a man with glasses and a light blue shirt speaking.

Now, the safety algorithm; so it actually checks whether some allocation will be whether the system is in safe state or unsafe state. So, it works with two temporary arrays, so $Work$ and $Finish$. So, $Work$ and $Finish$ they are vectors of length m and n . Initially $Work$ is equal to $Available$ and $Finish_i$ is false for all the processes. So, if there are n processes

this the Finish array has got n entries in it, it is a Boolean array and it is all initialized to false. And, the Work array, so this is an array of resources available for particular type. So, this is Work is available Work is initialized to the currently available resources.

Now, what the algorithm does is that it tries to find an i such that Finish i is false that is the process is not yet finished, and the Need of the process i is less than the Work. So, whatever is available. So, Work is the currently available one, so Need i with the need of the process i is less than what is available in the working array Work.

Then, we can satisfy that particular process to completion. So, if no such i exists; that means, there are two situation. One possibility is that Finish i is true for all the process for all i . So, in that case all the processes have could be completed with the available resources. So, the system is in a safe state.

Other possibility is that we could not find any process for which the second condition is satisfied that is Need i less or equal or Work, so that condition could be satisfied. So, as a result, so, that is also a situation, when there are there is a the system becomes unsafe actually. So, if none of them are true, then we could find an i such that the process has not yet been finished and its need is less than the currently available one currently available resources.

So, in that case we update the Work array because we assume that this the whatever this process needs so, that can be given from the Work array. So, accordingly this process will be able to finish off and once the process finishes off, all the resources that are currently allocated to process i becomes free. So, as a result my working array is enhanced by having all those resources claimed back from the process.

So, this Work array is enhanced by this Work equal to Work plus Allocation i and Finish i is set to true, so because this process is assumed to be finished. And, then it goes to step 2, it tries to find out another process of which can be finished now with this augmented Work array. So, this way if we can find that all the processes can be finished with the currently available work currently available set of resources in a particular sequence, then we say that the entire system is in a safe state. Otherwise, the system is in unsafe state and we have to avoid coming to an unsafe state.

So, we can utilize this safety algorithm to see like how this allocation decisions can be made whenever a process makes a request for a resource. So, it is first this safety algorithm will be run to see whether it can lead to a potential unsafe state and if we see that it is not leading to an unsafe state the system will remain safe even after this allocation is done, then only we will do that allocation. The resource allocation will be done only at that time; otherwise the process the process has to wait for the process to be free. So, we will continue with this in next class discussing on the allocation algorithm.