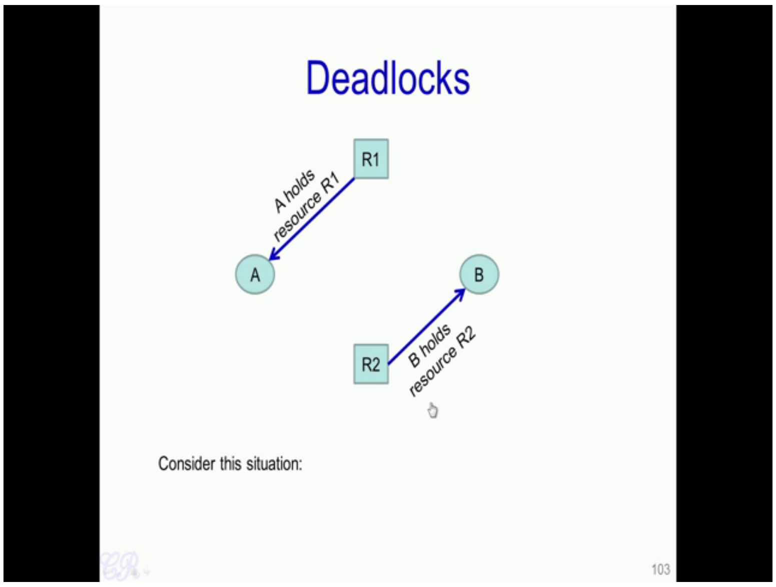
Introduction to Operating Systems

Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week – 07
Lecture – 31
Deadlocks

Hello. In the previous video, we had seen the dining philosophers' problem. And we had seen for the initial nave solutions that we had for the problem it could result in something known as deadlocks. So, in this video, we will look at more in detail about deadlocks, and how they are handled in the system.

(Refer Slide Time: 00:39)



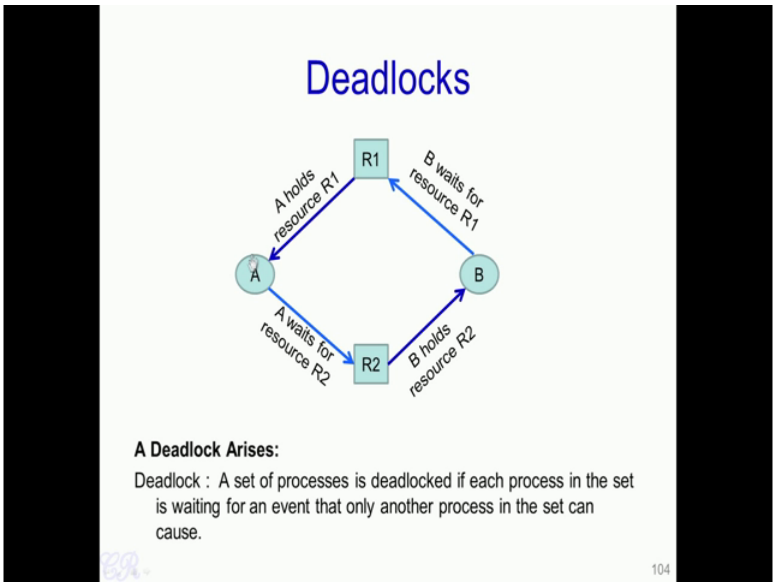
Deadlocks

Consider this situation:

Let us say we have two processes A and B, and we have two resources R 1 and R 2. So, these resources could be anything in the system which have a limited quantity, for example, the resources could be as something as small as a file or stored in the disc or it could be a printer which is used to print or a plotter, a scanner and so on.

So, essentially the arrow from R 1 to A indicates that A is currently holding the resource, for instance, if R 1 is the file; that means, A has currently open the file exclusively and it is doing some operations onto the file. In a similar way, the resource R 2 is held by B, so if it is a printer, for instance, if R 2 is a printer, it means that B is currently using the printer to print some particular document.

(Refer Slide Time: 01:52)



Deadlocks

R1
A holds resource R1
B waits for resource R1
A
B
A waits for resource R2
B holds resource R2
R2

A Deadlock Arises:

Deadlock : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

104

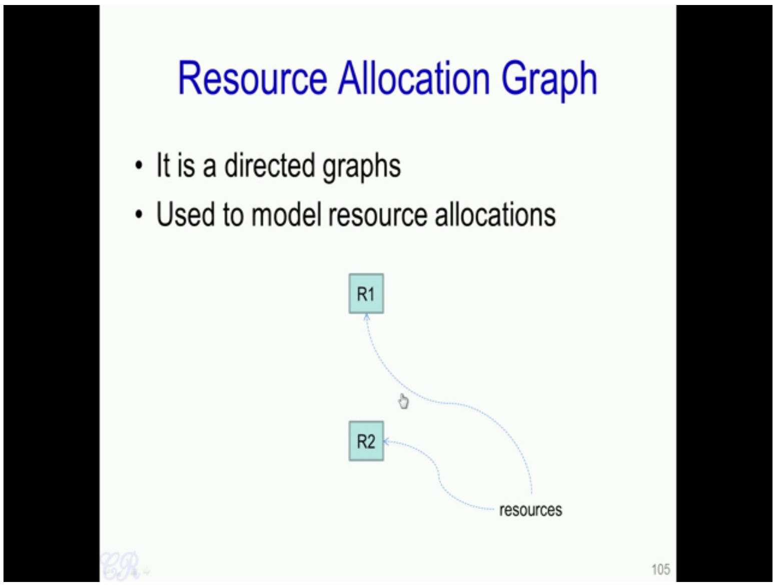
Now, consider this particular scenario where the process A holds the resource R 1, and process B holds the resource R 2; but at the same time, process A is requesting to use R 2. So, essentially the process A is waiting for R 2 to be obtained; and process B is waiting for the resource R 1 to be obtained. So, to take an example, process A is opened the file and is using a particular file which is stored in the disk.

And at the same time, for instance, it wants to print the file to the resource R 2 which we assumed as a printer. Now in a similar way, process B is currently holding this resource that is using this particular resource and it wants to open and utilize this particular resource R 1.

So, what we see that over here, we have a scenario called a deadlock. Essentially a deadlock is a state in the system where each process in the deadlock is waiting for an event that other process in that set can cause. For instance, over here, the process A is waiting for the resource R 2 which is held by B; B in turn is waiting for R 1 which is held by A. We have a set of two processes A and B; and each process, in the set is waiting for the other process to do something. And each process in the set is waiting for the other process to do a particular thing, for example, A is waiting for B to release this particular resource R 2, while B is also waiting for A to release the resource R 1.

So, deadlock like this is a very critical situation that would occur in systems. And when this deadlocks occur it could lead to process A and B in this case waiting for an infinite time continuously waiting without doing any useful work. So, such deadlocks should be analyzed thoroughly. So, in this particular video, we will see how such deadlocks are handled in systems. Now, in order to study deadlocks, we use graphs like this, these are known as resource allocation graphs.

(Refer Slide Time: 04:55)

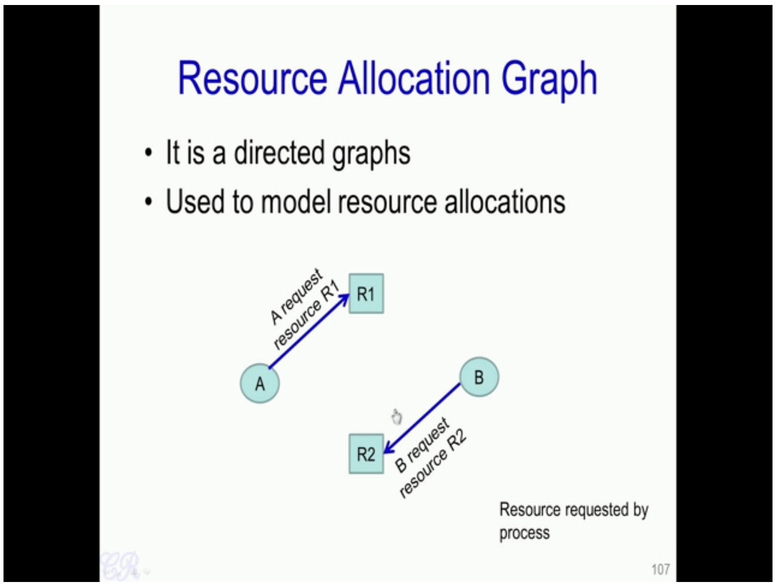


Resource Allocation Graph

- It is a directed graphs
- Used to model resource allocations

Now resource allocation graphs or directed graphs used to model the various resource allocations in the system. And there by determine whether a deadlock has occurred or a deadlock is potentially going to occur and so on. So, in this directed graph, we represent resources by A square. So, instance R 1 and R 2 are resources and they are represented by the square as shown over. Here in a similar way circles that have shown over here are used to represent processes.

(Refer Slide Time: 05:37)

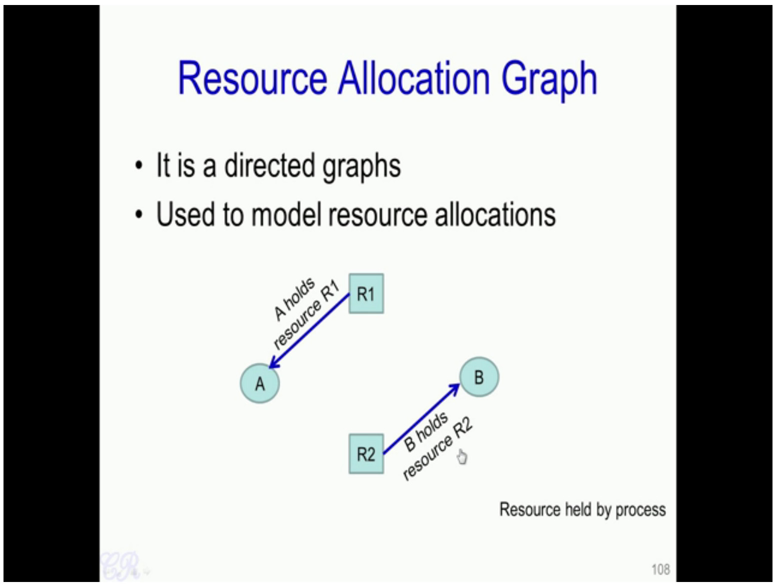


Resource Allocation Graph

- It is a directed graphs
- Used to model resource allocations

And as we seen before arrows from the process to the resource that is directed from the process to the resource would indicate that a request is made for that resource. For example, over here, the arrow from A to R 1 indicates that A is requesting for resource R 1; similarly B, in this case is requesting for resource R 2. So, these requests are made to the operating system; and if possible the operating system will then allocate that resource to the corresponding process.

(Refer Slide Time: 06:22)

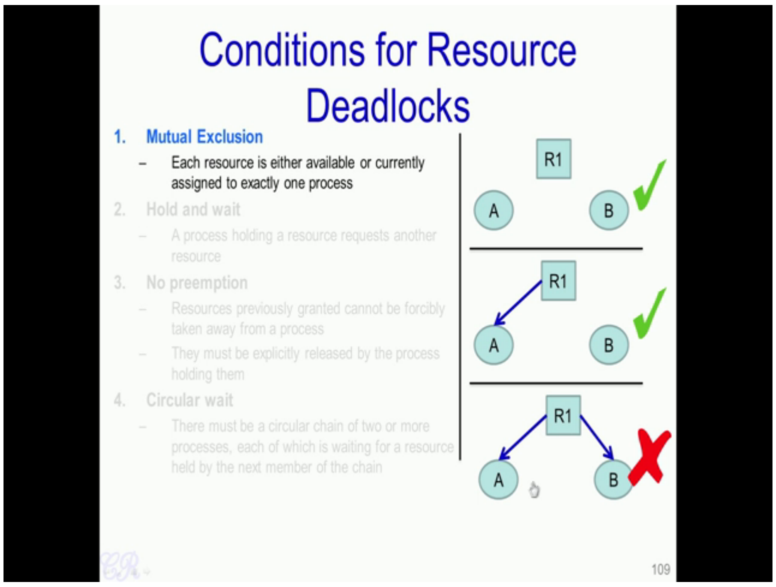


Resource Allocation Graph

- It is a directed graphs
- Used to model resource allocations

So, when that happens the graph will look like this essentially the direction of the arrow has changed. Now the arrow moves from R 1 to A indicating that A holds resource R 1. Similarly, the arrow from R 2 to B indicates that B holds resource R 2. And there are four conditions in order that a deadlock occurs. So, we will now look at each of these conditions for a deadlock.

(Refer Slide Time: 06:59)

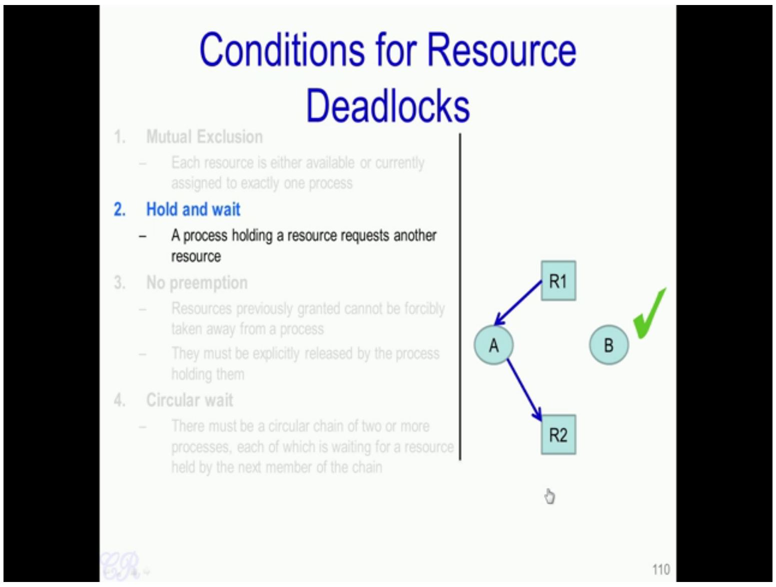


Conditions for Resource Deadlocks

1. **Mutual Exclusion**
   - Each resource is either available or currently assigned to exactly one process
2. Hold and wait
   - A process holding a resource requests another resource
3. No preemption
   - Resources previously granted cannot be forcibly taken away from a process
   - They must be explicitly released by the process holding them
4. Circular wait
   - There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain

So, the first is mutual exclusion. So, what we mean by this is that each resource in the system is either available or currently assigned to exactly one process. So, for instance over here we have resource R 1 which is free. So, it is not assigned to any particular process, so this is fine. While this is also fine where the resource is allocated to exactly one process, but in order that deadlocks happen this kind of scenario should not be present that is the resource cannot be shared between two processes A and B.

(Refer Slide Time: 07:48)

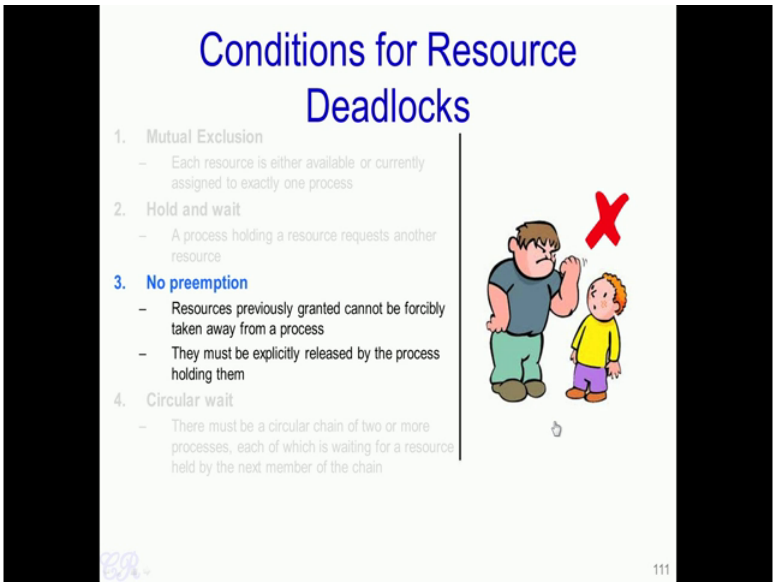


Conditions for Resource Deadlocks

1. Mutual Exclusion
   - Each resource is either available or currently assigned to exactly one process
2. Hold and wait
   - A process holding a resource requests another resource
3. No preemption
   - Resources previously granted cannot be forcibly taken away from a process
   - They must be explicitly released by the process holding them
4. Circular wait
   - There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain

The next condition for a deadlock is hold and wait that is a process holding A resource can request another resource that is in for example, then this case the resource R 1 is held by process A. And while having R 1, A is also requesting for another resource R 2, so it essentially holding R 1 and waiting for R 2.

(Refer Slide Time: 08:22)

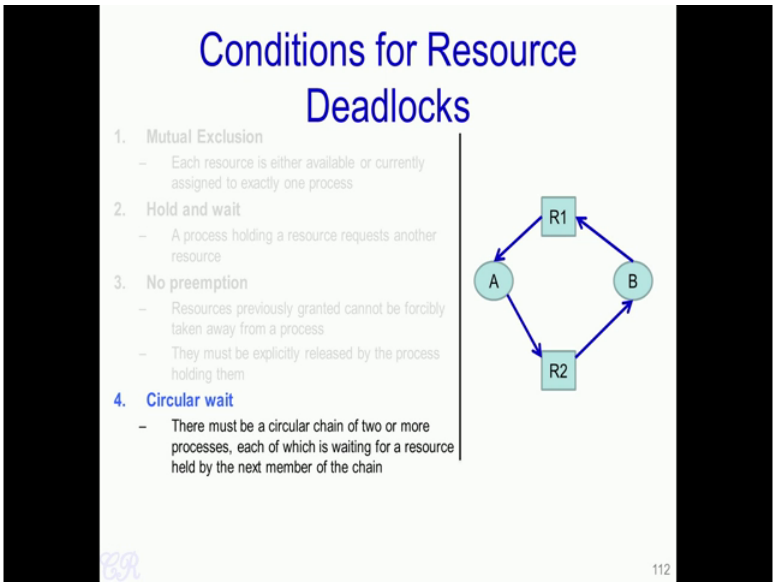


Conditions for Resource Deadlocks

1. Mutual Exclusion
   - Each resource is either available or currently assigned to exactly one process
2. Hold and wait
   - A process holding a resource requests another resource
3. No preemption
   - Resources previously granted cannot be forcibly taken away from a process
   - They must be explicitly released by the process holding them
4. Circular wait
   - There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain

The third condition for a deadlock to happen is no preemption. Essentially, it should not be the case that resources which an operating system previously granted for a particular process is forcibly taken away from that process that is the OS or another entity in the system cannot forcibly remove a resource which has been allocated to a particular process. Instead processes should explicitly release the resource by themselves that is whenever the process once it should release the resource by itself.

(Refer Slide Time: 09:18)



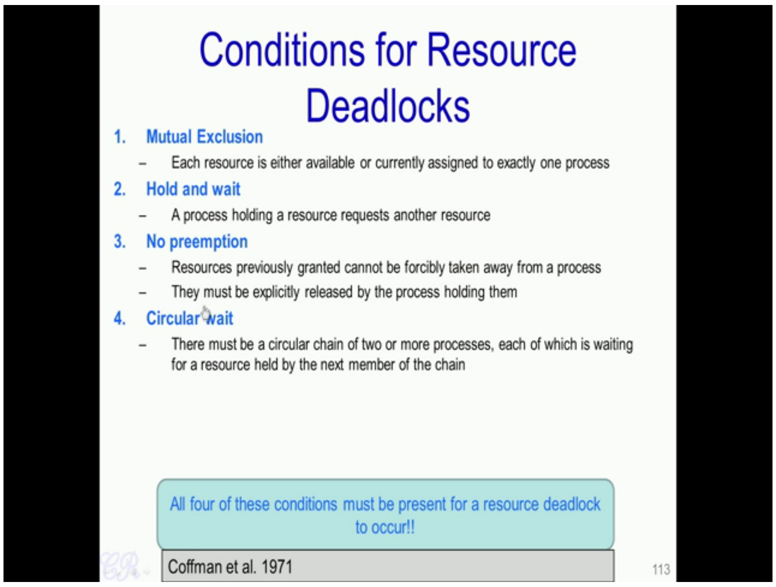
Conditions for Resource Deadlocks

1. Mutual Exclusion
   - Each resource is either available or currently assigned to exactly one process
2. Hold and wait
   - A process holding a resource requests another resource
3. No preemption
   - Resources previously granted cannot be forcibly taken away from a process
   - They must be explicitly released by the process holding them
4. Circular wait
   - There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain

112

So, a fourth requirement is the circular wait. What this means is that there is a circular chain of two or more processes each of which is waiting for a resource held by the next member of the chain. So, we see over here with that we have a circular chain and there is a wait over here because process A is waiting for process B to release resource R 2; and process B is in turn waiting for process A to release the resource R 1. So, we have a circular wait condition over here.

(Refer Slide Time: 10:00)



Conditions for Resource Deadlocks

1. **Mutual Exclusion**
   - Each resource is either available or currently assigned to exactly one process
2. **Hold and wait**
   - A process holding a resource requests another resource
3. **No preemption**
   - Resources previously granted cannot be forcibly taken away from a process
   - They must be explicitly released by the process holding them
4. **Circular wait**
   - There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain

All four of these conditions must be present for a resource deadlock to occur!!

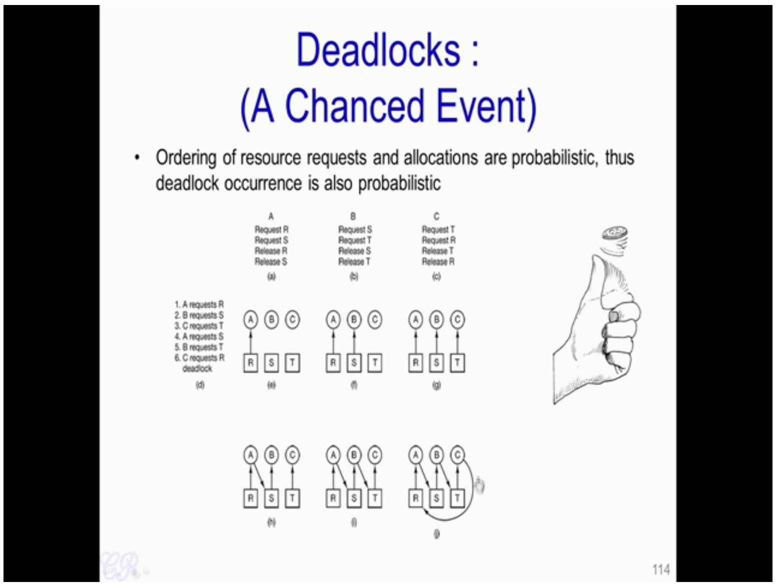
Coffman et al. 1971

113

So, these 4 conditions mutual exclusion, hold and wait, no preemption and circular wait must be present in the system in order that a deadlock could occur. So, if for instance, we were able to build a system where one of these conditions was not present. For example, suppose we build a system where processes cannot hold a particular resource and wait for another resource at the same time. So, such system would never have any deadlocks.

On the other hand, suppose a system has been developed where all of these things are possible that is there is a mutual exclusion when using resources a process could hold A resource and wait for another one; once allocated, they cannot be forcefully preempted from the resource; and circular wait mechanisms are allowed then deadlocks could potentially occur. So, having all these conditions does not imply that a deadlock has occurred. It only implies that there is a probability of deadlock occurring in the future.

(Refer Slide Time: 11:33)



Deadlocks : (A Chanced Event)

- Ordering of resource requests and allocations are probabilistic, thus deadlock occurrence is also probabilistic

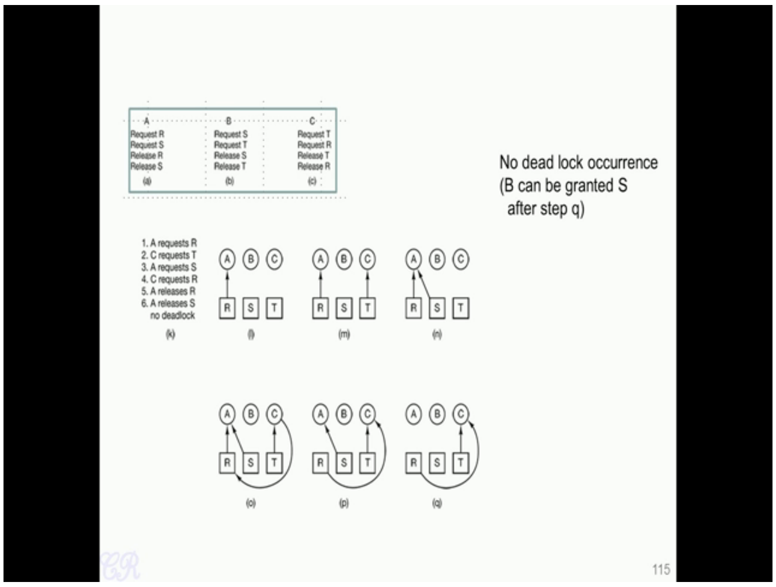
So, this being said a deadlock in a system is a chanced event. Essentially, it depends on several factors such as the way resources are requested by processes, the way allocations are made for these resources, the way de-allocations are made by the by the operating system and so on. So, only if certain order of these requests an allocation happen and only then will a deadlock occur. So, it is a small variation in the request and allocations may cause the deadlock to not occur.

Let us see some examples of this. Let us see we have three processes in the system A, B and C, and there are three resources as well R, S and T. So, each process could request and release resources at sometime during its execution. So obviously, a release can be made by a process only after the request has made. So, these request and release of resources are given to the operating system at various time instance, depending on how A, B and C get scheduled and how they are executed.

Let us consider this particular sequence that A requests R, and then B requests S, C request T; then A requests S, B requests T, and C requests R. So, this is one potential order for how request occur. So, we can use our resource graphs or resource allocation graphs to view this. So, we see that A requests R and the operating system will then allocate the resource R to A, then corresponding to B requests S, the allocation will be of S will be to B. Then C requests T and the OS will allocate T to C.

Then A requests S, so there is a line like this. B requests T there is a line over here; and C requests R. So, we have the four conditions that we have seen in the previous slide at all have all been met. For instance, the circular wait you see is achieved here. R is held by A, while A requests S; S is held by B and at the same time B request T. Now T is held by C, while simultaneously C is requesting for R. So, you see that each process is waiting for other process in this set to release a particular resource. So, we have a deadlocked scenario over here.

(Refer Slide Time: 14:55)



A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock

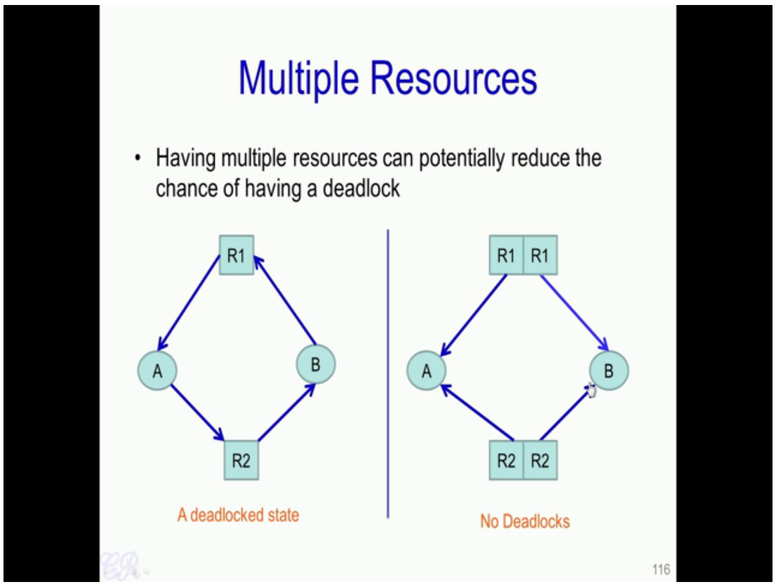
(k)

No dead lock occurrence
(B can be granted S
after step q)

Now, we will see that if the same requests and allocations are done in a slightly different manner then the deadlock will not occur. For example, R is allocated to A, and then T gets allocated to C; then A requests S and gets allocated to A; then C requests R, so we have this over here; now, A releases R, so there is no more line over here or an edge over here then a releases S, and therefore R can be allocated to C.

So, you see that depending on the requests and releases we are able to achieve a situation where each and every request or release can be serviced by the operating system. So, in such a scenario, we have not obtained the deadlock.

(Refer Slide Time: 16:07)



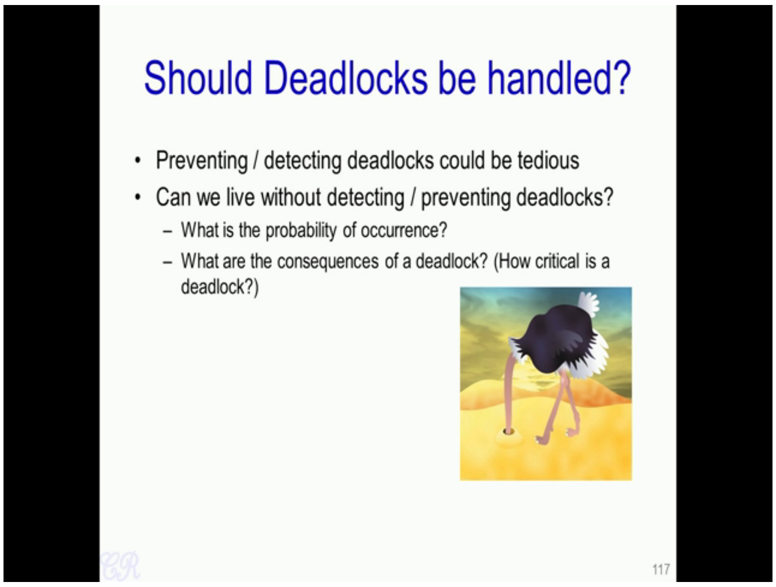
Multiple Resources

- Having multiple resources can potentially reduce the chance of having a deadlock

Now, we have seen that deadlock is indeed a probabilistic event and could occur with some probability. Now one way to reduce the probability is by having multiple resources present in the system. So, we had seen that this was a deadlock state, essentially because A is waiting for resource R 2 to be released by B, and B in turn is waiting for resource R 1. Now one way this can be solved is by having multiple resources, while this particular solution will not always work, and essentially depends on the type of resources, it may help to some extent.

For instance, if we have two resources of exactly the same type then both A is request as well as B is request could be managed that is the resource if we have two types of R 1 or in other words if you have a duplicate of the resource R 1 then that can be given to A as well as B. Similarly, a duplicate of the resource R 2 can be given to A an B simultaneously. So, what this means is that for example, we could have two printers present and a can be allocated one printer while B could be allocated the other printer while this does not completely eradicate deadlocks, it may reduce the likelihood the deadlocks may occur.

(Refer Slide Time: 18:00)



Should Deadlocks be handled?

- Preventing / detecting deadlocks could be tedious
- Can we live without detecting / preventing deadlocks?
  - What is the probability of occurrence?
  - What are the consequences of a deadlock? (How critical is a deadlock?)

117

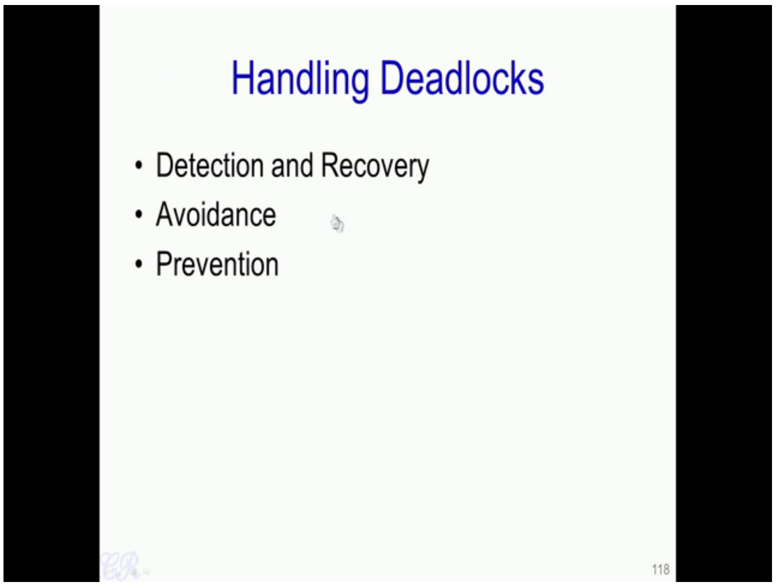
Now, the next question is in a system, which could have deadlocks, should deadlocks be handled. So, this is a debatable question, essentially is not an easy thing to answer, because the cost of having prevention mechanisms or to detect deadlocks is extremely high, and it will cost huge overheads in the operating system.

So, the other aspect was known as the ostrich algorithm is to completely ignore that deadlock could occur and run the system without any prevention or any deadlock detection mechanisms. So, either choice that is either having some deadlock prevention or detection mechanisms or just ignoring the entire aspects of deadlock would need to be made during the OS design time or rather the system design time. So, various things need to be discussed before a decision can be made. Such as what is the probability that a deadlock occurs is it likely that a deadlock will occur every week, or every month or once in 5 years or so on.

Second what is the consequence of a deadlock? Essentially, how critical a deadlock could be? For instance, if a deadlock occurs on my desktop, I could simply reboot the system and it is not going to affect me much. On the other hand, if deadlock occur say in safety critical application like spacecraft or a rocket kind of scenario, then the consequence could be disaster. Therefore, we need to argue about these two aspects essentially if the probability that a deadlock occurs is very frequent then probably the OS would requires some measures in order to handle the deadlock.

On the other hand, if the deadlock occurs very sporadic may be on average once in five years or so then you may not require to have or handle deadlock in the operating system. For now let us assume that we need some mechanism in our operating system to handle deadlocks. So, what can we do about this?

(Refer Slide Time: 20:47)



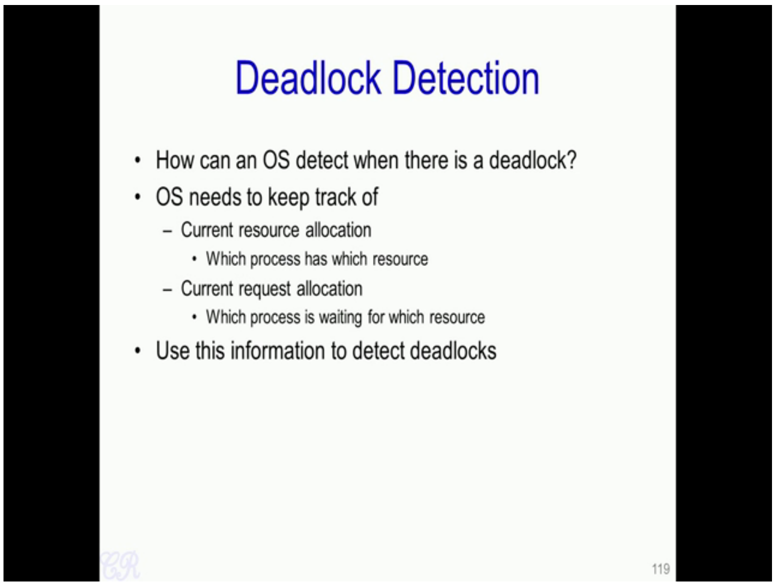
Handling Deadlocks

- Detection and Recovery
- Avoidance
- Prevention

118

Essentially, there are three ways that deadlocks can handle; one is by detection and recovery, second by avoidance and third by prevention.

(Refer Slide Time: 21:06)



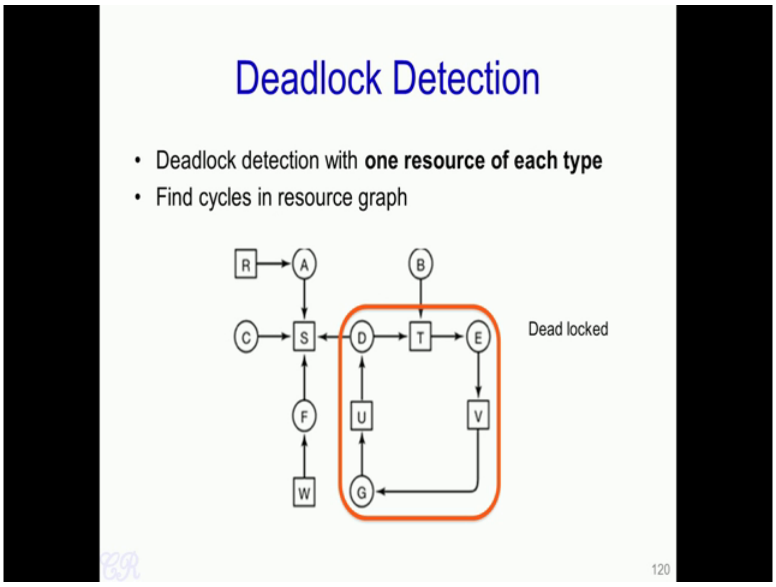
Deadlock Detection

- How can an OS detect when there is a deadlock?
- OS needs to keep track of
  - Current resource allocation
    - Which process has which resource
  - Current request allocation
    - Which process is waiting for which resource
- Use this information to detect deadlocks

119

Let us look at the first case that is detection and recovery. And first how are deadlocks detected. Essentially for the operating system to detect deadlocks, it requires to know the current resource allocation in the system, essentially which process holds which resources. And it also requires knowing the current request allocation by the processes. Essentially which process is waiting for which resources, and the OS will then use this information to detect if the system is in a deadlocked state.

(Refer Slide Time: 21:46)

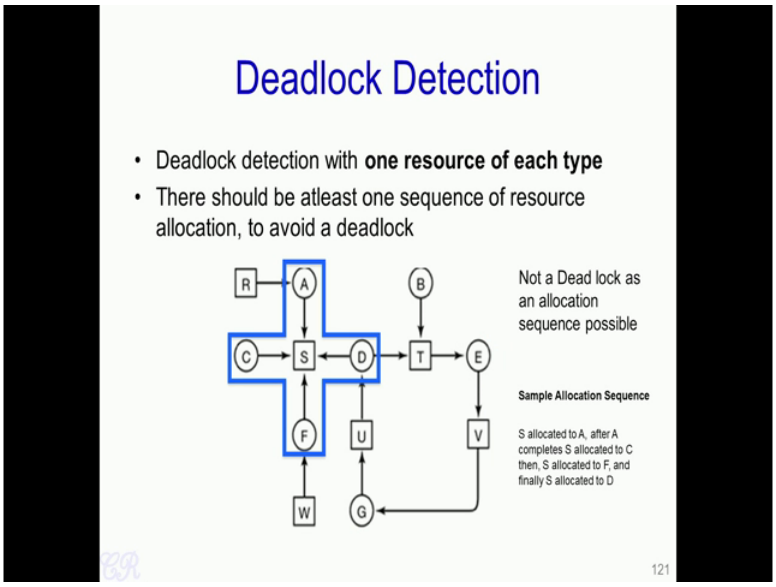


Deadlock Detection

- Deadlock detection with **one resource of each type**
- Find cycles in resource graph

So, the way the detection could work is by finding cycles in the resource allocation graphs. For instance, if this is the resource allocation graph for the various resources in the system, the OS will detect a cycle present. For example, over here, we have cycle between the processes D, E and G then the OS will say that these three processes are indeed in the deadlock state.

(Refer Slide Time: 22:19)



Deadlock Detection

- Deadlock detection with **one resource of each type**
- There should be atleast one sequence of resource allocation, to avoid a deadlock

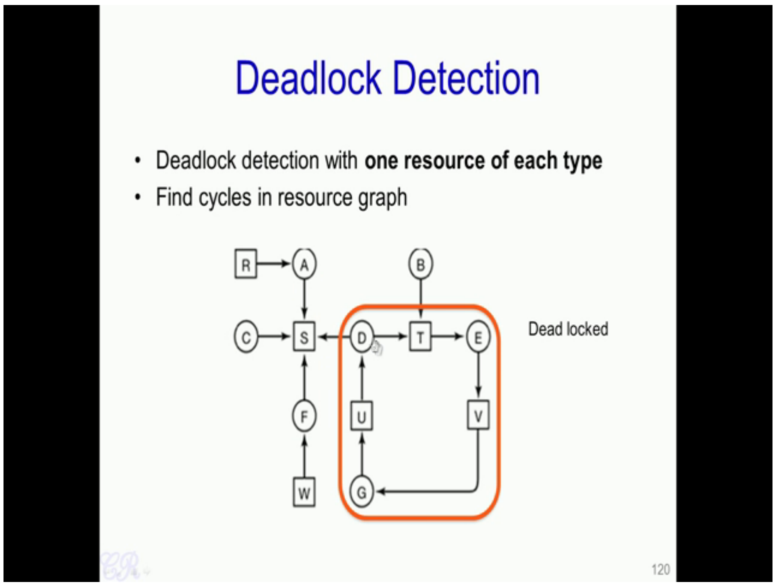
Not a Dead lock as an allocation sequence possible

Sample Allocation Sequence

S allocated to A, after A completes S allocated to C then, S allocated to F, and finally S allocated to D

Now the other aspect is there could be request as shown over here where there is resource S and it is requested simultaneously by A, D, F and C. So, we see that this is not in a deadlock state, because there is some sequence of allocation of S to all these processes. For instance one possible allocation came for S is that first S be allocated to a then A will use the resource S for some time, and then after it complete using S, S can be then allocated to C, and after which S is allocated to F, and then D. So, essentially the allocation of S could be sequential among these four processes. So, this will not have a deadlock.

(Refer Slide Time: 23:18)



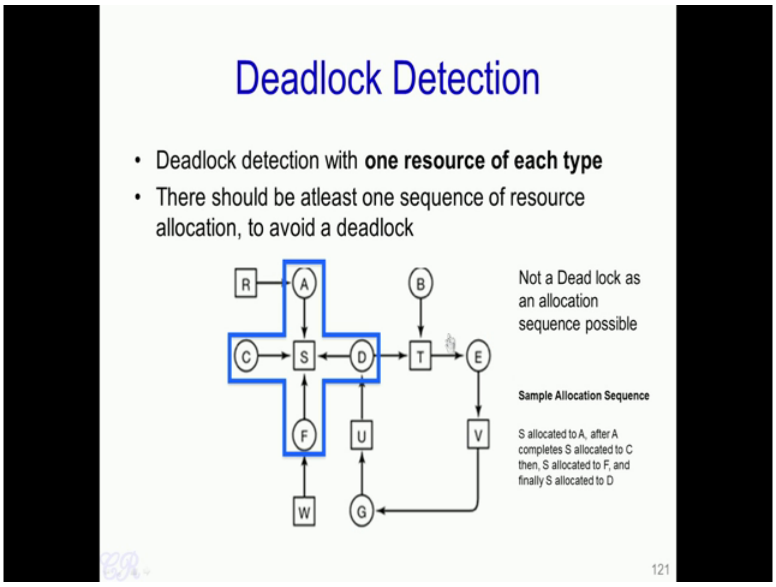
Deadlock Detection

- Deadlock detection with **one resource of each type**
- Find cycles in resource graph

Dead locked

However, the presence of a cycle in the resource allocation graph will indicate that a deadlock is present in the system.

(Refer Slide Time: 23:28)



Deadlock Detection

- Deadlock detection with **one resource of each type**
- There should be atleast one sequence of resource allocation, to avoid a deadlock

Not a Dead lock as an allocation sequence possible

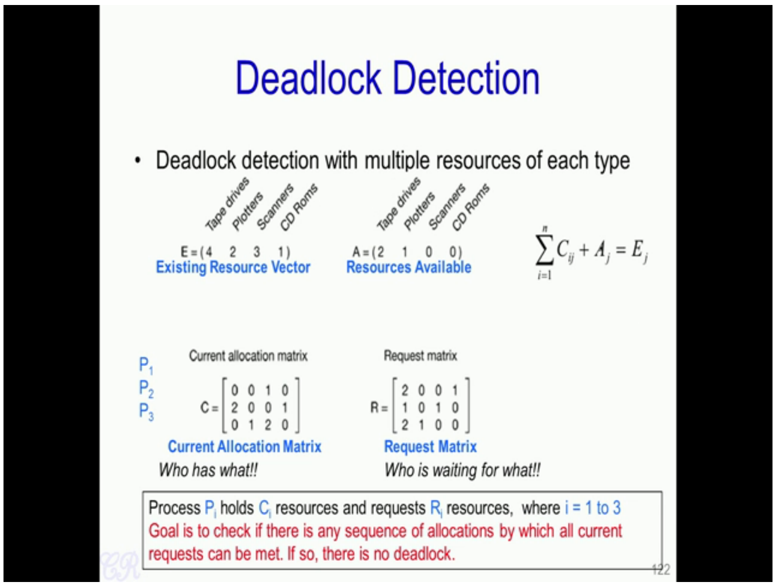
Sample Allocation Sequence

S allocated to A, after A completes S allocated to C then, S allocated to F, and finally S allocated to D

121

This technique of finding cycles in the resource allocation graph would work well with systems where there was one resource of each type. Now suppose we have systems where there multiple resources of each type then another algorithm would be required.

(Refer Slide Time: 23:45)



Deadlock Detection

- Deadlock detection with multiple resources of each type

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ \end{array}$$

E = (4 2 3 1)
Existing Resource Vector

A = (2 1 0 0)
Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{cc} P_1 & \\ P_2 & C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \\ P_3 & \end{array}$$

Current Allocation Matrix

Who has what!!

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Request Matrix

Who is waiting for what!!

Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3
Goal is to check if there is any sequence of allocations by which all current requests can be met. If so, there is no deadlock.

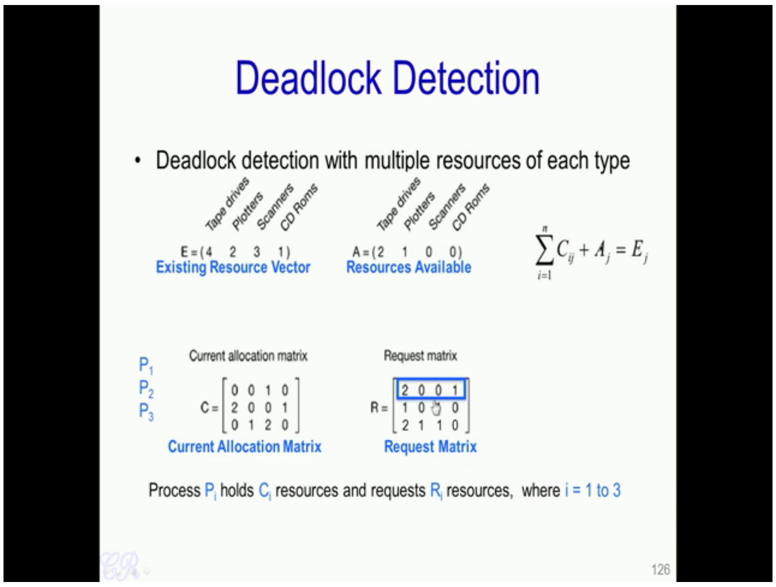
122

Let us give an example of how that would work. Let us say in our system we have four resources tape drives, plotters, scanners and CD ROMs. Further, there are 4 tape drives, 2 plotters, 3 scanners and 1 CD-Rom. Then let us also say that we have three processes executing in the system P 1, P 2, P 3, and this is the current allocation matrix. So, the row 1 is with respect to process P 1. This means that the process P 1 does not have any tape drives, does not have any plotters is allocated 1 scanner and no CD-ROMs. Similarly, process 2 is allocated 2 tape drives and 1 C D Rom. And process 3 is allocated 1 plotter and 2 scanners.

Now, this particular a set A determines the resource available; essentially out of the 4 tape drives that we have if you look into this corresponding column over here in the current allocation matrix we have seen that two is used. So, what remain is 4 minus 2 that is 2 tape drives are free and available to be allocated. So, similarly if you look at plotters out of the 2 plotters 1 is allocated and one is free. Out of the 3 scanners, we see that all 3 scanners are allocated, so it is 0 over here. Similarly, there are no CD-ROMs also available to be allocated.

Now, in addition to this we have a request matrix essentially which process is waiting for what, is represented in this matrix. For example, process p 1 requests 2 more tape drives and 1 CD Rom. So, process P 2 requires 1 tape drives and 1 scanner, and process P 3 requires 2 tape drives and 1 plotter. Now the goal of having such definitions or such representations of the resources allocations and request is to determine if there exists a sequence of allocations of these requests, so that all request can be met; if such a case is possible, then there is no de deadlock present. Let us see if we can allocate these requests by the 3 processes.

(Refer Slide Time: 26:52)



Deadlock Detection

- Deadlock detection with multiple resources of each type

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (4 & 2 & 3 & 1) \end{array}$$

Existing Resource Vector

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (2 & 1 & 0 & 0) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{c} P_1 \\ P_2 \\ P_3 \end{array} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Request matrix

$$R = \begin{bmatrix} \boxed{2 \; 0 \; 0 \; 1} \\ 1 \; 0 \; 1 \; 0 \\ 2 \; 1 \; 1 \; 0 \end{bmatrix}$$

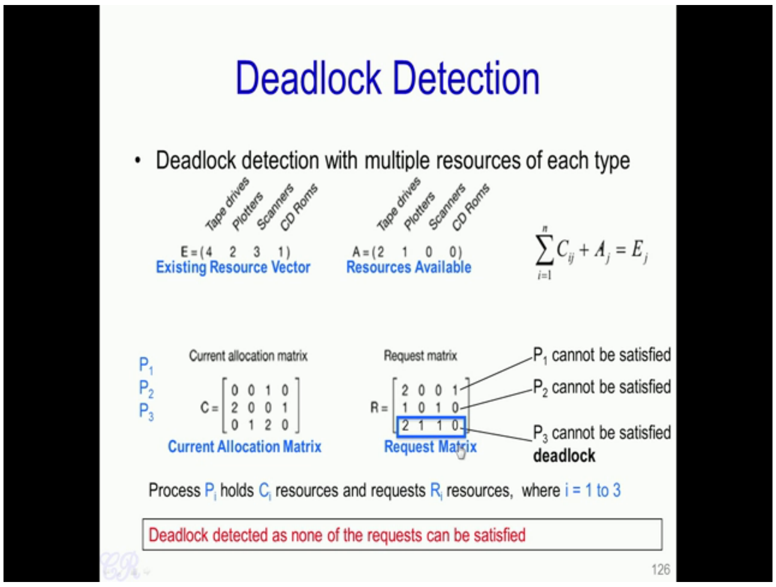
Request Matrix

Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3

126

Let us take a process P 1 and let us see the request by process P 1 can be met. So, it requires 2 tape drives and we see that 2 tape drives are available. So, this is fine and 1 CD-Rom is requested, but there are no CD-Rom is available. So, process P 1 cannot execute. So, the process P 1 cannot be allocated all its resources. So, it cannot continue to execute.

(Refer Slide Time: 27:17)



Deadlock Detection

- Deadlock detection with multiple resources of each type

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (4 & 2 & 3 & 1) \end{array}$$

Existing Resource Vector

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (2 & 1 & 0 & 0) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{c} P_1 \\ P_2 \\ P_3 \end{array} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Request matrix

$$R = \begin{bmatrix} 2 \; 0 \; 0 \; 1 \\ 1 \; 0 \; 1 \; 0 \\ \boxed{2 \; 1 \; 1 \; 0} \end{bmatrix}$$

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 cannot be satisfied

deadlock

Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3

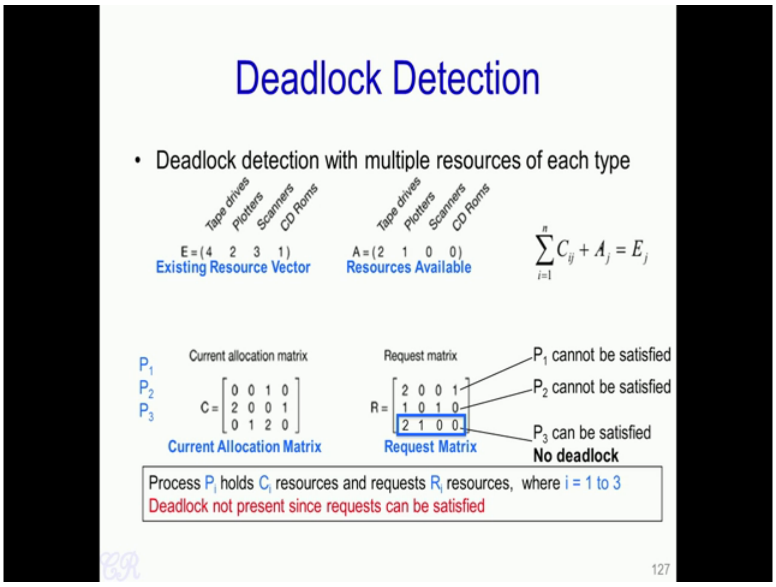
Deadlock detected as none of the requests can be satisfied

126

Let us see about process P 2. So, process P 2 requires 1 tape drive which can be allocated to it, because it is available; and it requires 1 scanner, but 0 are available. So, similarly, process P 2 cannot be satisfied as well, and process P 2 will also need to wait. Now let see the third case where for process 3, there are 2 tape drives which can be met, 1 request for a plotter which can be met and 1 request for a scanner which cannot be met.

So, process P 3 also cannot be satisfied its request. So, P 3 also will wait. So, P, 1 P 2 and P 3 are all waiting for the request, and therefore the state is in a deadlock because none of the requests by any of these processes can be met and therefore, all the processes will need to be waiting.

(Refer Slide Time: 28:34)



Deadlock Detection

- Deadlock detection with multiple resources of each type

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (4 & 2 & 3 & 1) \end{array}$$

Existing Resource Vector

$$A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (2 & 1 & 0 & 0) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{c} P_1 \\ P_2 \\ P_3 \end{array} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \boxed{2 \ 1 \ 0 \ 0} \end{bmatrix}$$

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 can be satisfied

No deadlock

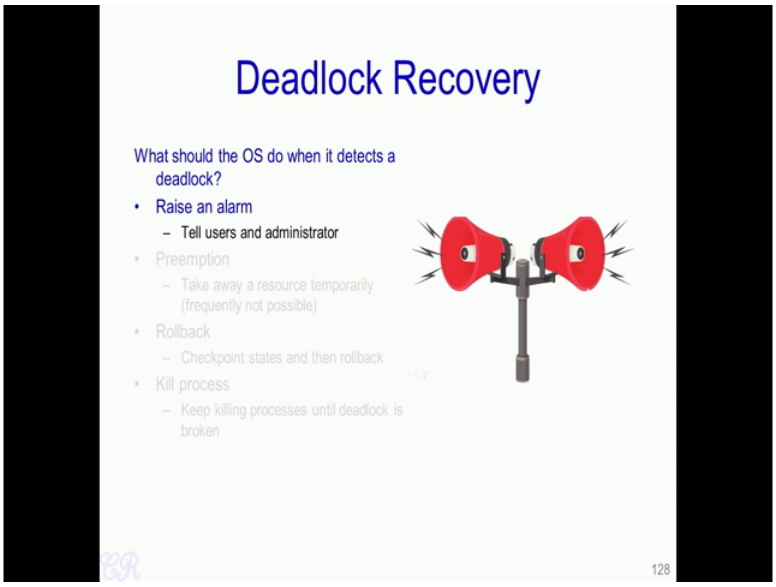
Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3

Deadlock not present since requests can be satisfied

127

If you look at another example where there is a small change. We have to just reduce the number of scanners required by a process P 3. So, in such a case, we see that both the tape drives are requested by process P 3 can be met by the system the single plotter can be also met and there are no scanners and no CD-ROMs that are required. And therefore, we see that all the request can be allocated to process P 3, therefore P 3 is can be satisfied and we do not have a deadlock over here. So, in this way, deadlocks can be detected by the operating system, based on the current allocation matrix and the request matrix.

(Refer Slide Time: 29:32)



Deadlock Recovery

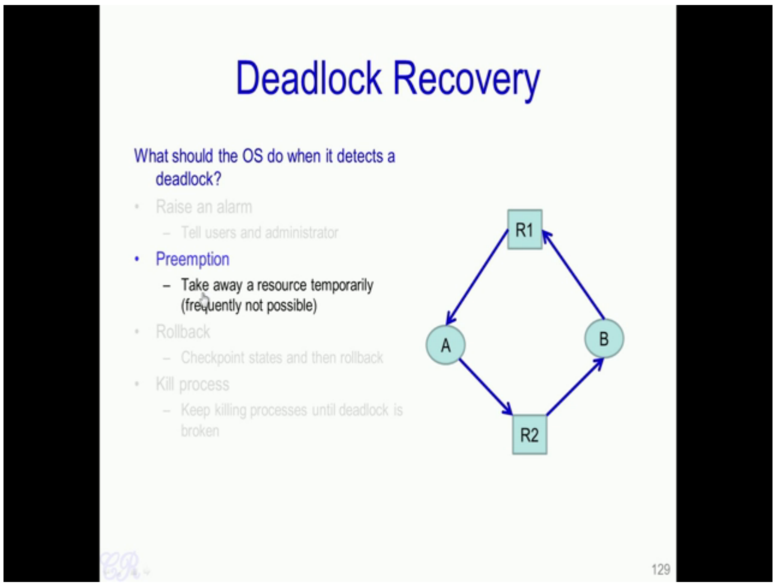
What should the OS do when it detects a deadlock?

- Raise an alarm
  - Tell users and administrator
- Preemption
  - Take away a resource temporarily (frequently not possible)
- Rollback
  - Checkpoint states and then rollback
- Kill process
  - Keep killing processes until deadlock is broken

128

So, once the deadlocks are detected, what next should the operating system do. So, there could be various things that the OS could do. So, one thing is that it could raise an alarm that is telling users and administrator that indeed deadlocks has been detected.

(Refer Slide Time: 29:51)



Deadlock Recovery

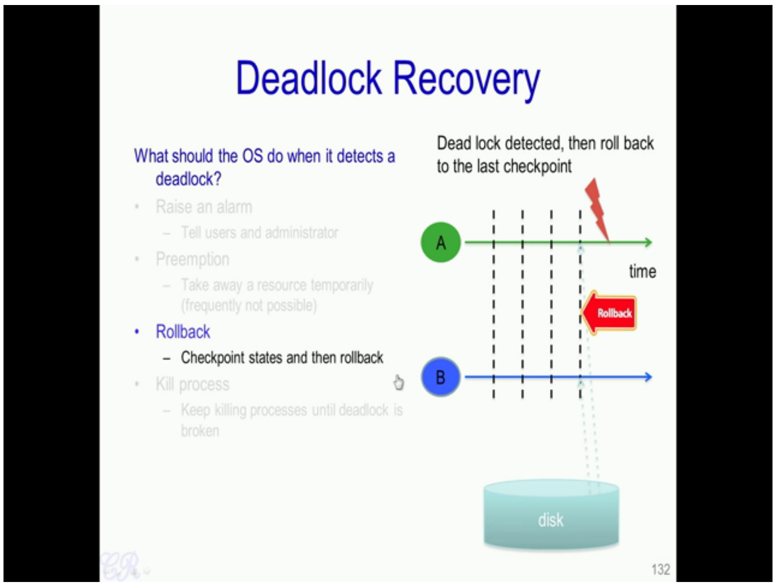
What should the OS do when it detects a deadlock?

- Raise an alarm
  - Tell users and administrator
- Preemption
  - Take away a resource temporarily (frequently not possible)
- Rollback
  - Checkpoint states and then rollback
- Kill process
  - Keep killing processes until deadlock is broken

129

Then a second way is to force a preemption that is you force a particular resource to be taken away from a process and given to another process. For instance, it could be like R 2 was a printer and it is currently held by B. So, what could be done was that the printer could be forced to be taken away from B, while it is allocated to A for some time. And thus the way the deadlock will be broken this is shown over here. So, B no longer has other resource R 2, but R 2 is giving to A.

(Refer Slide Time: 30:42)



Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
  - Tell users and administrator
- Preemption
  - Take away a resource temporarily (frequently not possible)
- Rollback
  - Checkpoint states and then rollback
- Kill process
  - Keep killing processes until deadlock is broken

Dead lock detected, then roll back to the last checkpoint
A
time
Rollback
B
disk
132

Then a third method is by using A technique known as rollback. So, with rollback, both processes A and B as they execute will be check pointed. So, by check point, we mean that the state of the process gets stored on to the disk, the process execute for some time and then the entire state of the process is execute gets stored on to the disk. Now storing this sate of the process on the disk will allow the processes to execute from the point where it has been check pointed, now the checkpoint state. So, as time progresses more check points are taken periodically are shown over here.

Now, let us say a deadlock is detected after sometime then what could happen is that the system could rollback to the last non deadlock state that is over here. So, how the rollback occurs would be by loading the state of process A and B in this particular example to the last known state are shown over here.

Now process A and B will continue to execute from this state. And the deadlock may not occur again essentially we have seen that since deadlock is a probabilistic event by modifying the ordering in which the allocations are made we could prevent this particular deadlock. A fourth way is to kill processes. Essentially, if process A and B are in A deadlock state, killing one of these processes would break the deadlock. So, typically the lower important or the less priority process would be killed.

Thank you.