

MODULE – 3

SYSTEM MODELING

System modeling is the interdisciplinary study of the use of models to conceptualize and construct systems in business and IT development.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- Request. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- Use. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- Release. The process releases the resource.

DEADLOCK CHARACTERIZATION

Necessary Conditions

1. Mutual exclusion. At least one resource must be held in a non sharable mode; that is, only one process at a time can use the resource. The process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and wait. A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No preemption. Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait. A set $\{ P_0, P_1, \dots, P_{n-1} \}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 , \dots , P_{n-1} is waiting for a resource held by P_0 and P_{n-1} is waiting for a resource held by P_0 .

Resource-Allocation Graph: Deadlocks can be described more precisely in terms of a directed graph called a graph.

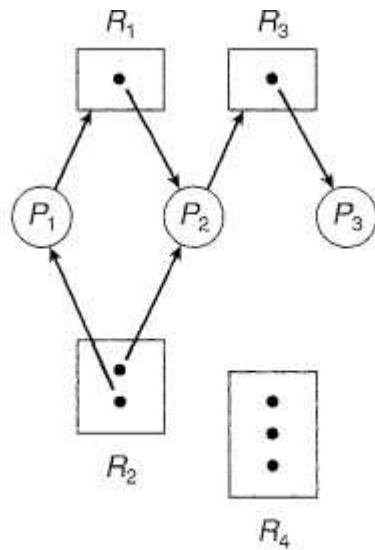


Figure 7.2 Resource-allocation graph.

The sets P , R and E :

$P == \{P_1, P_2, P_3\}$

$R == \{R_1, R_2, R_3, \}$

$E == \{P_1 \rightarrow R_1, R_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$

Resource instances:

One instance of resource type R_1

Two instances of resource type R_2

One instance of resource type R_3

Three instances of resource type R_4

Process states: Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1 . Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3 . Process P_3 is holding an instance of R_3 .

Resource-allocation graph with a deadlock.

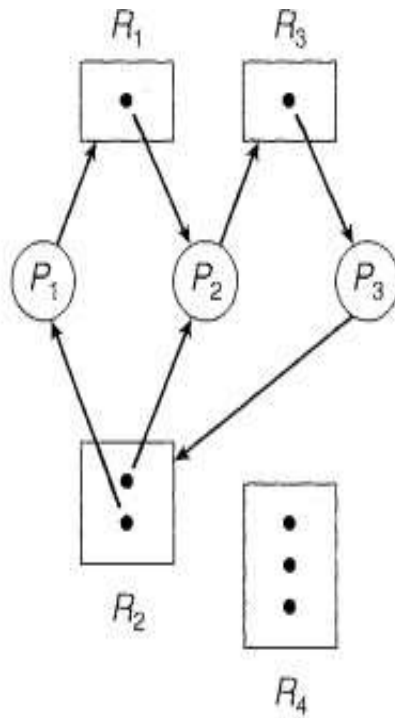


Figure 7.3 Resource-allocation graph with a deadlock.

Resource-allocation graph with a cycle but no deadlock.

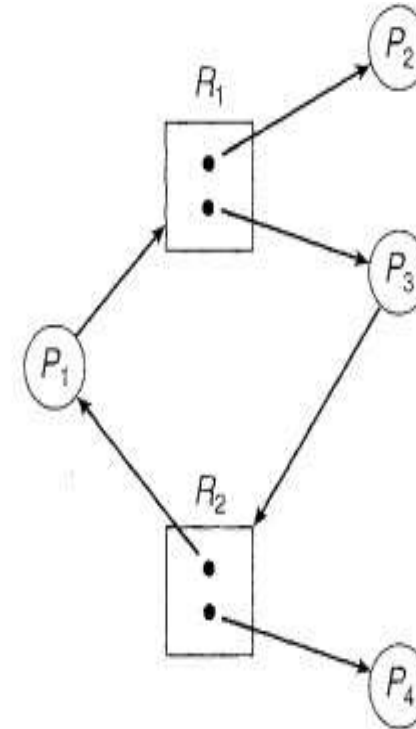


Figure 7.4 Resource-allocation graph with a cycle but no deadlock.

METHODS OF HANDLING DEADLOCKS

Deadlock prevention:

- 1. Mutual exclusion.** At least one resource must be held in a non sharable mode; that is, only one process at a time can use the resource. If a process requests that resource, the requesting process must be delayed until the resource has been released.
- 2. Hold and wait.** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
- 3. No preemption.** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- 4. Circular wait.** A set { P₀ , P₁, ... , P_{n-1} } of waiting processes must exist such that P₀ is waiting for a resource held by P₁, P₁ is waiting for a resource held by P₂, ... , P_{n-1} is waiting for a resource held by P₀, and P₀ is waiting for a resource held by P₀.

Deadlock Avoidance: The deadlock Avoidance method is used by the operating system in order to check whether the system is in a safe state or in an unsafe state and in order to avoid the deadlocks, the process must need to tell the operating system about the maximum number of resources a process can request in order to complete its execution.

Safe State

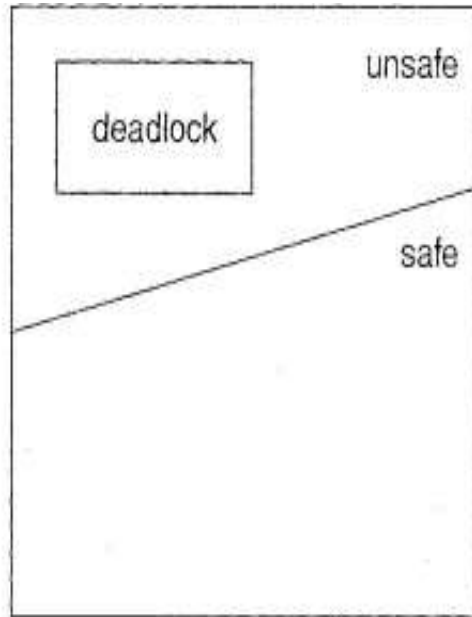


Figure 7.5 Safe, unsafe, and deadlocked state spaces.

Resource-Allocation-Graph Algorithm

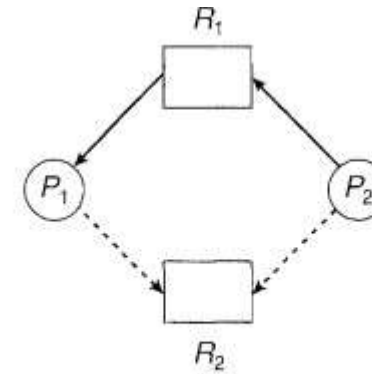


Figure 7.6 Resource-allocation graph for deadlock avoidance.

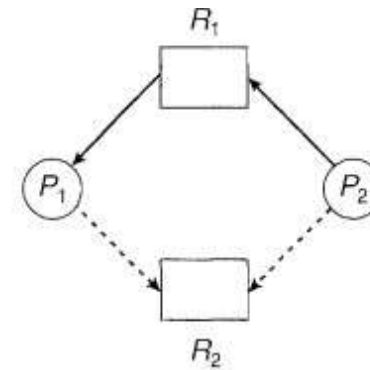


Figure 7.6 Resource-allocation graph for deadlock avoidance.

Deadlock detection

Single Instance Resource Type

If all resources have only a single instance, then we can define a deadlock-detection algorithm that uses a variant of the resource-allocation graph, called a *wait-for* graph.

This graph is obtained from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

More precisely, an edge from P_i to P_j in a wait-for graph implies that process P_i is waiting for process P_j to release a resource that needs.

Several Instances of a Resource Type

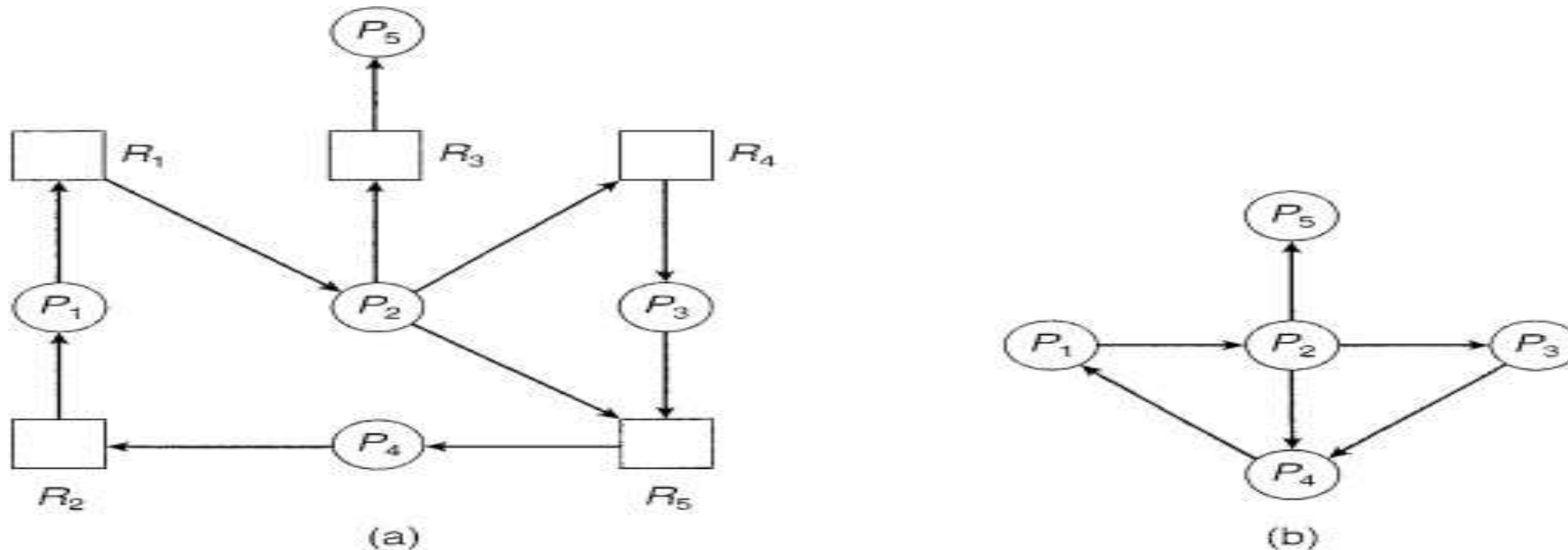


Figure 7.8 (a) Resource-allocation graph. (b) Corresponding wait-for graph.

Detection-Algorithm Usage

If deadlocks occur frequently, then the detection algorithm should be invoked frequently. Resources allocated to deadlocked processes will be idle until the deadlock can be broken. In addition, the number of processes involved in the deadlock cycle may grow. Deadlocks occur only when some process makes a request that cannot be granted immediately. This request may be the final request that completes a chain of waiting processes. In the extreme, then, we can invoke the deadlock detection algorithm every time a request for allocation cannot be granted immediately.

Recovery from deadlock

1. Process Termination

Abort all deadlocked processes. This method clearly will break the deadlock cycle, but at great expense; the deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.

Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

Resource Pre-emption

1. **Selecting a victim.** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed during its execution.
2. **Rollback.** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state.

Since, in general, it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it. Although it is more effective to roll back the process only as far as necessary to break the deadlock, this method requires the system to keep more information about the state of all running processes.

3. **Starvation.** How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?

