**Deadlock Modeling:**

Deadlocks can be described more precisely in terms of Resource allocation graph. It's a set of vertices \( V \) and a set of edges \( E \). \( V \) is partitioned into two types:

\[ P = \{ P_1, P_2, ..., P_n \}, \text{the set consisting of all the processes in the system.} \]
\[ R = \{ R_1, R_2, ..., R_m \}, \text{the set consisting of all resource types in the system.} \]

request edge – directed edge \( P_i \rightarrow R_j \)

assignment edge – directed edge \( R_j \rightarrow P_i \)

---

**Symbols**

- \( P_1 \)
- \( R_1 \)
- \( P_2 \)
- \( R_2 \)

---

a). \( P_1 \) is holding \( R_1 \)

b). \( P_1 \) requests \( R_1 \)

---

- Example of a Resource allocation graph
- Resource allocation graph with Deadlock
- Resource Allocation graph with a Cycle but no Deadlock

---

Deadlock Modeling:

Deadlocks can be described more precisely in terms of Resource allocation graph. It's a set of vertices \( V \) and a set of edges \( E \). \( V \) is partitioned into two types:
Basic Facts:
If graph contains no cycles ⇒ no deadlock.
If graph contains a cycle ⇒
  ● If only one instance per resource type, then deadlock.
  ● If several instances per resource type, possibility of Deadlock

Methods for Handling Deadlock:
Allow system to enter deadlock and then recover
  ● Requires deadlock detection algorithm
  ● Some technique for forcibly preempting resources and/or terminating tasks
Ensure that system will never enter a deadlock
  ● Need to monitor all lock acquisitions
  ● Selectively deny those that might lead to deadlock
Ignore the problem and pretend that deadlocks never occur in the system
  ● Used by most operating systems, including UNIX

Deadlock Prevention
To prevent the system from deadlocks, one of the four discussed conditions that may create a deadlock should be discarded. The methods for those conditions are as follows:

1. Mutual Exclusion:
   In general, we do not have systems with all resources being sharable. Some resources like printers, processing units are non-sharable. So it is not possible to prevent deadlocks by denying mutual exclusion.

2. Hold and Wait:
   One protocol to ensure that hold-and-wait condition never occurs says each process must request and get all of its resources before it begins execution.
   Another protocol is “Each process can request resources only when it does not occupies any resources.”

   The second protocol is better. However, both protocols cause low resource utilization and starvation. Many resources are allocated but most of them are unused for a long period of time. A process that requests several commonly used resources causes many others to wait indefinitely.

3. No Preemption:
   One protocol is “If a process that is holding some resources requests another resource and that resource cannot be allocated to it, then it must release all resources that are currently allocated to it.”
   Another protocol is “When a process requests some resources, if they are available, allocate them. If a resource it requested is not available, then we check whether it is being used or it is allocated to some other process waiting for other resources. If that resource is not being used, then the OS preempts it from the waiting process and allocate it to the requesting process. If that resource is used, the requesting process must wait.” This protocol can be applied to resources whose states can easily be saved and restored (registers, memory space). It cannot be applied to resources like printers.
4. Circular Wait:
One protocol to ensure that the circular wait condition never holds is “Impose a linear ordering of all resource types.” Then, each process can only request resources in an increasing order of priority.

For example, set priorities for r1 = 1, r2 = 2, r3 = 3, and r4 = 4. With these priorities, if process P wants to use r1 and r3, it should first request r1, then r3.

Another protocol is “Whenever a process requests a resource rj, it must have released all resources rk with priority(rk) ≥ priority (rj).

Source: http://dayaramb.files.wordpress.com/2012/02/operating-system-pu.pdf