**Deadlock Detection**

**Detection algorithm**: An algorithm that examines the state of the system to determine whether a deadlock has occurred.

**Recovery algorithm:** An algorithm to recover from the deadlock.

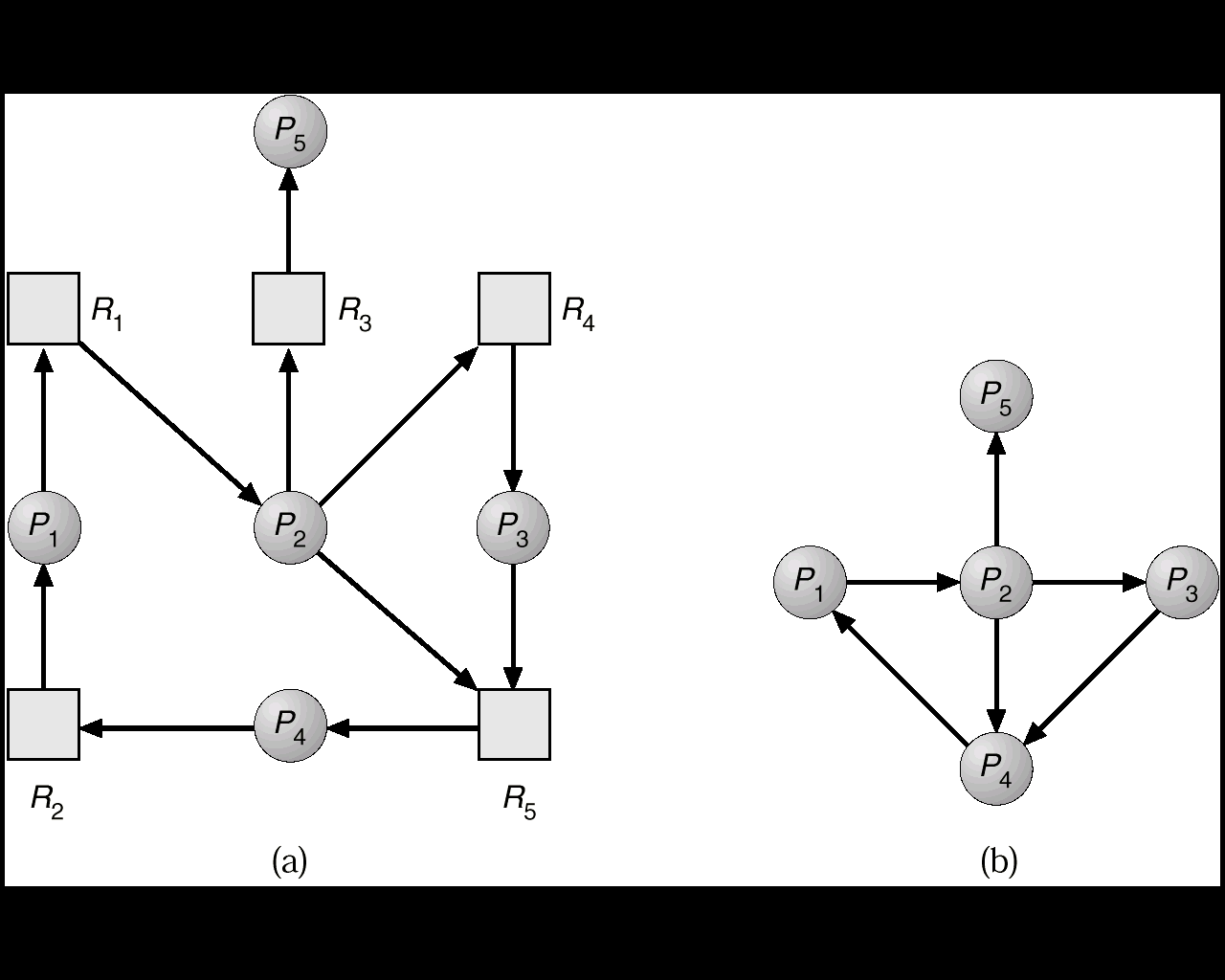
**There are two detection algorithms:**

1. Wait-for graph (Every type has one single instance).
2. Detection-algorithm (similar to safety algorithm) (Every type has any number of instances)

**Wait-for graph:**

* Generate a wait-for graph for its corresponding Resource allocation graph
* Nodes are processes.
* *Pi* → *Pj* if *Pi* is waiting for *Pj.*
* Periodically invoke an algorithm that searches for a wait-cycle in the graph.

Figure below shows the Resource-Allocation Graph and Wait-for Graph.

****

**Detection-algorithm:**

* + ***Available****:* A vector of length *m* indicates the number of available resources of each type.
  + ***Allocation****:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process every row of allocation is a vector.
  + ***Request****:* An *n* x *m* matrix indicates the current request of each process. If *Request* [*i,j*] = *k*, then process *Pi* is requesting *k* more instances of resource type. *Rj*. Every row of request is a vector.
  1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:

1. *Work* = *Available*
2. *For i = 1,2, …, n, if Request i ≠ 0, then   
   Finish[i] = false;*
3. *Otherwise, Finish[i] = true.*
   1. Find an index i such that both:
4. *Finish* [*i*] == *false*
5. *Requesti* ≤ *Work*

If no such *i* exists, go to step 4.

* 1. Work = Work + Allocation i  
     Finish[i] = *true*  
     go to step 2.
  2. If Finish[i] == false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state.
  3. Moreover, if Finish[i] == false, then Pi is deadlocked.

**An example of detection algorithm:**

* Five processes *P*0 through *P*4;three resource types   
  A (7 instances), *B* (2 instances), and *C* (6 instances).
* Snapshot at time *T*0:

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Allocation* | *Request* | *Available* |
|  | *A B C* | *A B C* | *A B C* |
| *P*0 | 0 1 0 | 0 0 0 | 0 0 0 |
| *P*1 | 2 0 0 | 2 0 2 |  |
| *P*2 | 3 0 3 | 0 0 0 |  |
| *P*3 | 2 1 1 | 1 0 0 |  |
| *P*4 | 0 0 2 | 0 0 2 |  |

* Sequence <P0, P2, P3, P4 ,P1> will result in Finish[i] = true for all i.

**Deadlock Recovery:**

* Process Termination
* Resource Preemption

**Deadlock Recovery: Process Termination**

* + Abort all deadlocked processes: Abort one process at a time until the deadlock cycle is eliminated.
  + In which order should we choose to abort?

The order will be chosen according the following:

* Priority of the process.
* How long process has computed, and how much longer to completion.
* Resources the process has used.
* Resources process needs to complete.
* Is process interactive or batch?

**Deadlock Recovery: Resource Preemption**

* Selecting the victim, minimize cost.
* Rollback – return to some safe state, restart process for that state.

**Note**: Starvation – the same process may always be picked as victim.