

---

## Paper Name : Operating System

Prepared by : Debanjali Jana

### **Resource Allocation Graph -**

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

It contains a set of vertices  $V$  and a set of edges  $E$ .

$V$  is partitioned into two types:

- $P = \{P_1, P_2, \dots, P_n\}$ , the set of all the processes in the system
- $R = \{R_1, R_2, \dots, R_m\}$ , the set of all resource types in the system

Types of edges:

1. Assign Edge –

- If a resource is already assigned to a process then it is called assign edge.
- directed edge  $R_j \rightarrow P_i$


2. Request Edge –


- It means in future the process might want some resource to complete the execution, that is called request edge.
- directed edge  $P_i \rightarrow R_j$


So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node

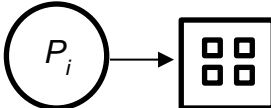
- If graph contains no cycles  $\Rightarrow$  no deadlock

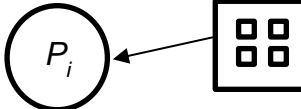
- If graph contains a cycle  $\Rightarrow$ 
  - ❖ If each resource in the cycle provides only one instance, then the processes will be in deadlock. So cycle in single-instance resource type is a sufficient condition for deadlock.
  - ❖ In multi-instance resource type, cycle is not a sufficient condition for deadlock.

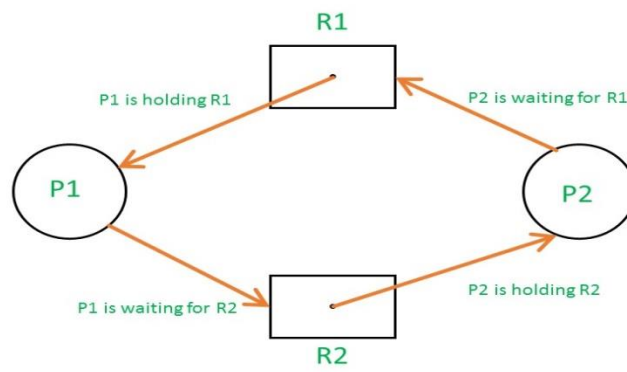
Process 

Resource 

Resource type with 4 instances 

$P_i$  requests instance of  $R_j$  

$P_i$  is holding an instance of  $R_j$  



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

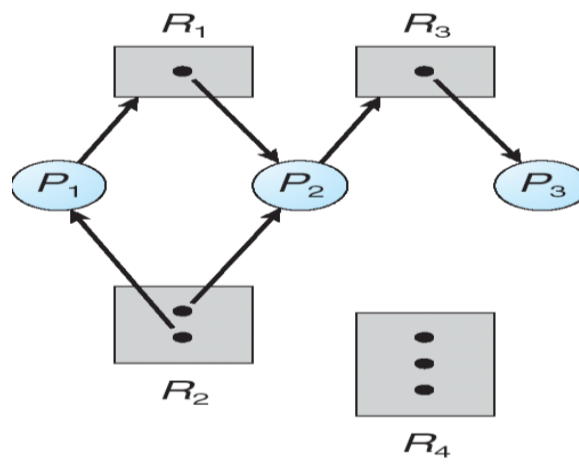


Fig : Example of resource allocation graph

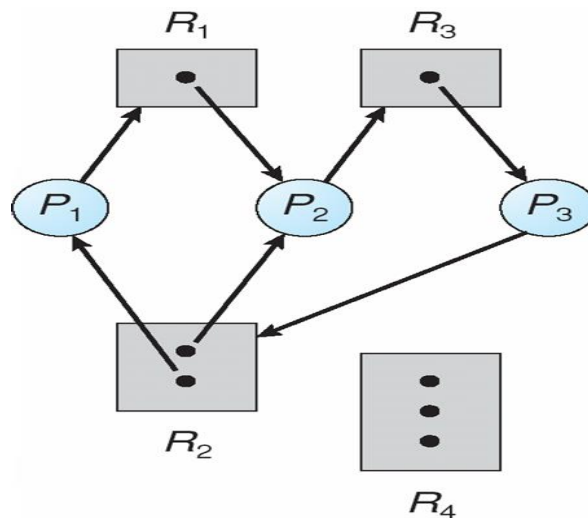


Fig : Example of resource allocation graph with a deadlock

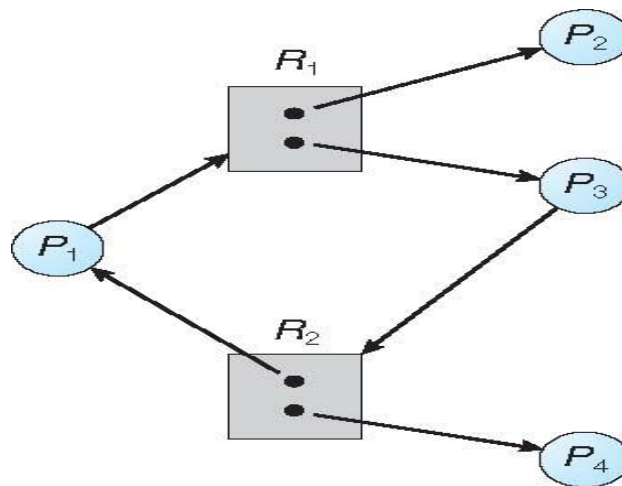


Fig : Example of resource allocation graph with a cycle but no deadlock

## Deadlock Prevention -

We can prevent deadlock by eliminating any of the above four conditions.

### 1. Eliminate Mutual Exclusion

The mutual-exclusion condition must hold for non-shareable resources. For example, a printer cannot be simultaneously shared by several processes. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good example of a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.

In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-shareable.

If access to a resource requires mutual exclusion, then mutual exclusion must be supported by the operating system. Some resources such as files may allow multiple accesses for reads but only exclusive access for writes. Even in this case, deadlock can occur if more than one process requires write permission.

## **2. Eliminate Hold and wait -**

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

a. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.

b. An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

## **3. Eliminate No pre-emption –**

If the resource can be taken away from the process which is causing deadlock then deadlock can be prevented.

## **4. Eliminate Circular Wait -**

To avoid circular wait, resources may be ordered and we can ensure that each process can request resources only in an increasing order of these numbers. The algorithm may itself increase complexity and may also lead to poor resource utilization.

## **Deadlock Avoidance –**

To avoid a deadlock, before allocation a system decides whether it should allocate or not so that it does not enter the deadlock. The request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

A resource allocation state is defined by the number of available and allocated resources, and the maximum requirements of all processes in the system.

A deadlock avoidance algorithm dynamically examines the resource allocation state from time to time to ensure that no circular wait condition exists.

A resource allocation state is safe if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state. Otherwise it is a unsafe state.

A safe state is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e. all the processes can be run to completion). An unsafe state is, of course, a state that is not safe.

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

