Deadlock

Introduction

In a computer system, we have a finite number of resources to be distributed among a number of competing processes.

System resources are classified into:

Physical Resources- It includes printers, tape drivers, memory space, CPU etc.

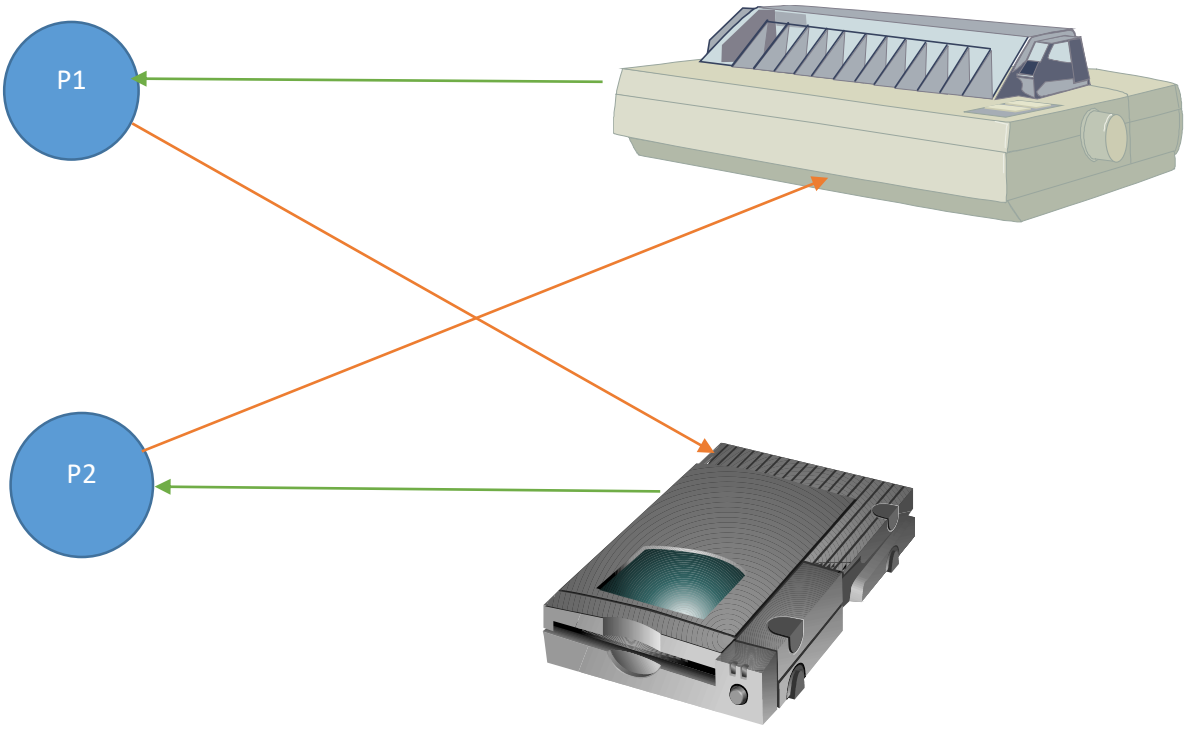
Logical Resources- Includes files and semaphores.

A process must request a resource before using it and release the resource after using it. A process may request as many resources as it requires to carry out its designated task. The number of resources requested cannot exceed the total number of resources available in the system. In a normal operation, a process may utilize a resource in the following sequence:

- Request the resource- If the request cannot be immediately granted, then the requesting process must wait until it can get the resource.
- Use the resource - The requesting process can operate on the resource.
- Release the resource - The process releases the resource after using it.

The OS is responsible for making sure that the requesting process has been allocated the resource. Request and release of resources can be accomplished through the wait and signal operations on semaphores. A system table records whether each resource is free or allocated, and if allocated, to which process. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource. In a multi-programming environment, several processes may compete for a fixed number of resources. A process requests resources and if the resources are not available at that time, it enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called **deadlock.**

Consider a system with one printer and one scanner. Process P1 request the printer and process P2 requests the scanner. Both requests are granted. Now P1 requests the scanner (without giving up the printer) and P2 requests the printer (without giving up the scanner). Neither request can be granted so both processes enter a deadlock situation. It is explained in the following figure.



A deadlock is a situation where a group of processes is permanently blocked as a result of each process having acquired a set of resources needed for its completion and having to wait for release of the remaining resources held by others thus making it impossible for any of the deadlocked processes to proceed. Deadlocks can occur in concurrent environments as a result of the uncontrolled granting of the system resources to the requesting processes.

A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. Each member of the set of deadlock processes is waiting for a resource that can be released only by a deadlock process.

Deadlock Characterization

In a deadlock, processes never finish executing, and the system resources are ties up, preventing other jobs from starting.

Necessary Conditions

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

Mutual Exclusion

Hold and Wait

No Preemption

Circular Wait

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 another process requests that resource, the requesting process must be delayed until the resource has been released.

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

No Preemption - Resources already allocated to a process cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

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

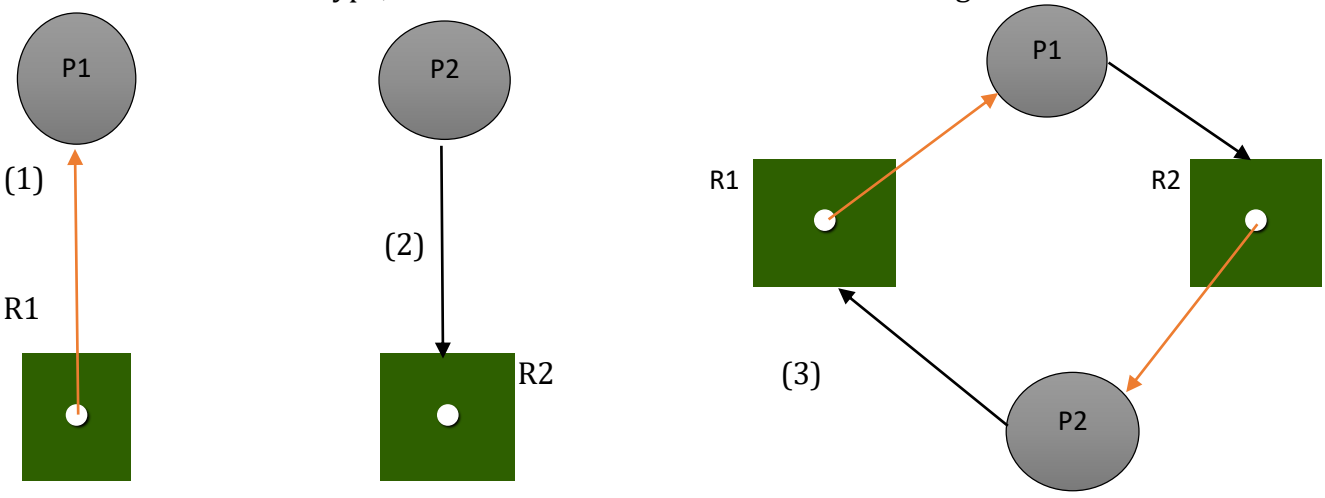
The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

Resource Allocation graph

A graphical representation showing the allocation of resources to the process. Resource-allocation graph is used to describe the deadlock. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all active processes in the system, and $R = \{R_1, R_2, ..., R_m\}$, the set consisting of all resource types in the system. A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$; it signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource. A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$; it signifies that an instance of resource type R_j has been allocated to process P_i.

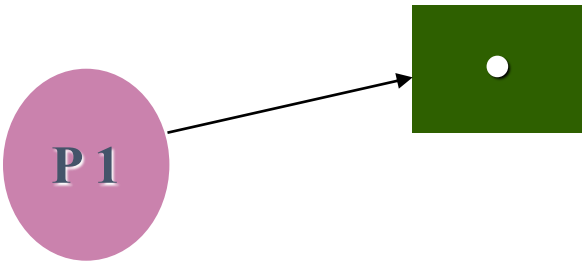
A directed edge $P_i \rightarrow R_j$ is called a **request edge** and a directed edge $R_j \rightarrow P_i$ is called an **assignment edge**.

Process node is represented by circles. Resource node is represented by rectangles. For each instances of a resource type, there is a dot in the resource node rectangle.

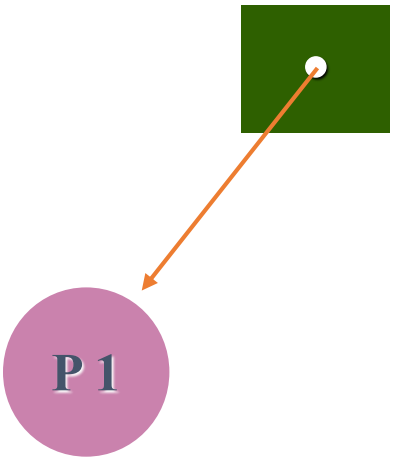


(1) → P1 is holding the resource R1
(2) → P2 is requesting a resource R2
(3) Deadlock situation

A request edge points to only the square, where as an assignment edge must also designate one of the dots in the square. When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource allocation graph.



When this request is fulfilled, the request edge is instantaneously transformed to an assignment edge.



When the process no longer needs access to the resource, it releases the resource, and as a result the assignment edge is deleted.