Introduction

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process 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 a deadlock.

A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set.

Example: When two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other.

System Model

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each of which consists of some number of identical instances. Memory space, CPU cycles, files, and I/O devices (such as printers and tape drives) are examples of resource types. If a system has two CPUs, then the resource type CPU has two instances.

A process must request a resource before using it, and must release the resource after using it. A process may request as many resources as it requires carrying out its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system. In other words, a process cannot request three printers if the system has only two. a process may utilize a resource in only the following sequence:

1. **Request:** If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
2. **Use:** The process can operate on the resource.
3. **Release:** The process releases the resource.

A system table records whether each resource is free or allocated, and, if a resource is 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.

Necessary Conditions

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

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 another 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_o, P_1, ..., P_n,) of waiting processes must exist such that P_o 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_o.

Question: Is it possible to have a deadlock involving only one process? Explain your answer.

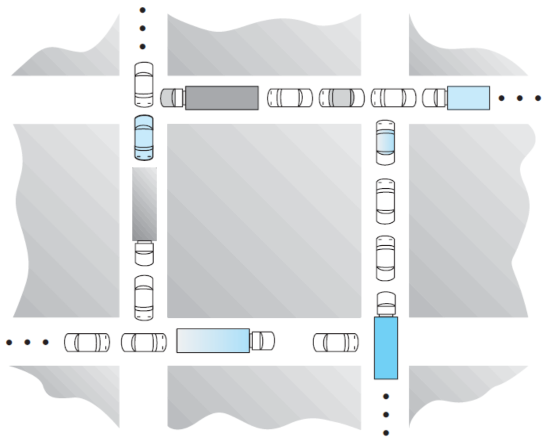
Answer: A deadlock situation can arise if the following four conditions hold simultaneously in a system.

- Mutual Exclusion.
- Hold and Wait.
- No Preemption.
- Circular-wait.

It is not possible to have circular wait with only one process, thus failing a necessary condition for Circular wait. There is no second process to form a circle with the first one. So it is not possible to have a deadlock involving only one process.

Question: Consider the traffic deadlock depicted in Figure

1. Show that the four necessary conditions for deadlock indeed hold in this example.
2. State a simple rule for avoiding deadlocks in this system.



Answer:

1. The four necessary conditions for a deadlock are
   a. Mutual Exclusion
   b. Hold and Wait
   c. No Preemption
   d. Circular wait.

- The mutual exclusion condition holds as only one car can occupy a space in the roadway.
- Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway.
- A car cannot be removed (i.e. preempted) from its position in the roadway.
- Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

2. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E.

G = { V, E }

The set of vertices V is partitioned into two different types of nodes:

- Process (Represented by Circle)
- Resource Types (Represented by rectangle)

The set of edged E is partitioned into two different types of edges:

- **Request Edge:** A directed edge from process Pi to resource type *Rj* is denoted by Pi → Rj, it signifies that process ***Pi*** requested an instance of resource type Rj
- **Assignment Edge:** A directed edge from resource type Rj to process Pi is denoted by Rj → Pi, it signifies that an instance of resource type Rj has been allocated to process Pi.

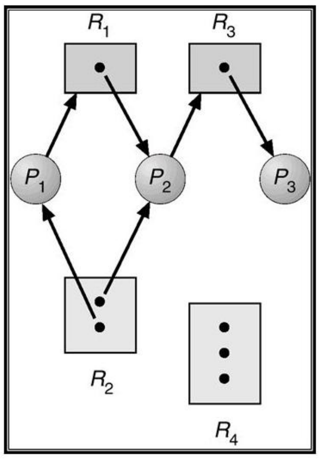


Figure: Resource Allocation Graph

The resource-allocation graph shown in depicts the following situation.

The sets ***P,*** R, and E:

- P = { P1, P2, P3}
- R = { R1, R2, R3, R4 }
- E = { P1→ R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

Resource instances:

- One instance of resource type R1
- Two instances of resource type R2
- One instance of resource type R3
- Three instances of resource type R4

Process states:

- Process *P1* is holding an instance of resource type *R2*, and is waiting for an instance of resource type *R1*.
- Process *P2* is holding an instance of *R1* and *R2*, and is waiting for an instance of resource type *R3*.
- Process *P3* is holding an instance of *R3*.

If the graph contains no cycles, then no deadlocked. If the graph does contain a cycle, then a deadlock may exist. If each resource type has exactly one instance, then deadlock. If each resource type has several instances, then possibility of deadlock.

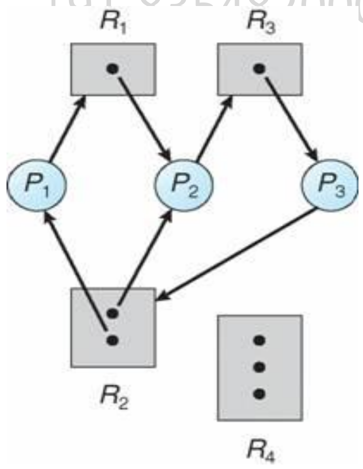


Figure: Resource Allocation Graph with a deadlock

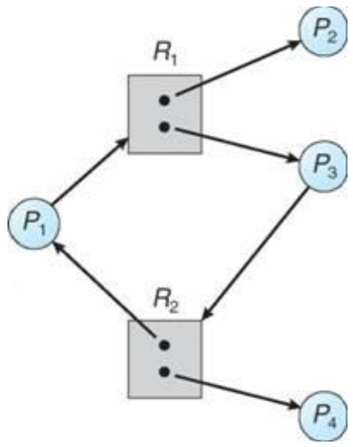


Figure: Resource Allocation Graph with a cycle but no deadlock

Question: A system is having 3 user processes each requiring two units of resource 'R' the minimum number of units of 'R such that no deadlock will occur.
Solution: *Refer you class notebook (Discussed during the lecture)*

Question: The computer system has 6 tape drives, with 'n' processes competing for them. Each process needs 3 tape drives. The maximum value of 'n' for which the system is garmented to be deadlock free.
Solution: *Refer you class notebook (Discussed during the lecture)*