Operating system concepts

Process Synchronization (deadlocks handling, detection, prevention, avoidance)

Slides Set #12

By Prof K R Chowdhary

JNV University

2023

Semaphores: Used to solve synchronization problems

- In a multiprogramming environment, several processes may compete for a finite number of resources.
- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a **deadlock**.
- Although some applications can identify programs that may deadlock, Operating systems typically do not provide deadlock-prevention facilities,
- Questions:
  - How, you can prevent the occurrence of deadlock?
  - Is there possibility of deadlock in batch type of OS?
  - Is it possible in DOS?

System Model

- A system consists of a finite number of resources to be distributed among a number of competing processes.
- If a process requests an instance of a resource type, the allocation of any instance of the type should satisfy the request.
- Various **synchronization tools** are **mutex locks** and **semaphores**. A lock is typically associated with protecting a specific data structure.
- A process must request a resource before using it and must release the resource after using it.
- *A* **set** *of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.*
- To illustrate a deadlocked state, we can consider a system with *three CD RW drives*.
- Deadlocks may also involve different resource types.

Deadlock Characterization and Necessary Conditions

- In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.
- Conditions of deadlock:
  1. **Mutual exclusion**: At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource.
  2. **Hold and wait**: A process must be holding at least one resource and waiting to acquire additional resources
  3. **No preemption**: Resources cannot be preempted;
  4. **Circular wait**.
- Questions:
  1. What are the conditions of deadlock? Explain each one of them.
  2. What will be the problems, if one or more processes are deadlocked?
  3. If operating system has no provision of deadlock handling, what you will do if you are user of that OS?

Resource-Allocation Graph

- Deadlocks can be described in terms of a directed graph, called a **system resource-allocation graph** $G = (V, E)$.
- A directed edge from process P_i to resource type R_j is denoted by $P_i \to R_j$;
- Pictorially, we represent each process Pi as a circle and each resource type R_j as a rectangle.
- When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource allocation graph.

- The sets P, R, and E: $P = \{P_1, P_2, P_3\}$, $R = \{R_1, R_2, R_3, R_4\}$, $E = \{P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_2, R_2 \to P_1, R_3 \to P_3\}$.

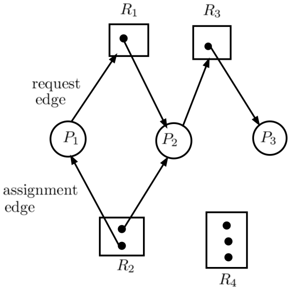

Figure 1: Resource allocation Graph.

Resource-Allocation Graph..

- Resource instances: e.g., One instance of resource type R1
- Process states: Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1. Others...
- Given the definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked.
- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred.

Resource-Allocation Graph..

- Cycle: $P_2 \to R_3 \to P_3 \to R_2 \to P_2$.

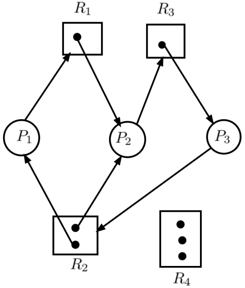

Figure 2: Resource allocation Graph with deadlock.

- Cycle: $P_1 \to R_2 \to P_2 \to R_3 \to P_3 \to R_2 \to P_1$. Processes P_1, P_2, P_3 are deadlocked. The P_2 is waiting for resource R_3 (held by process P_3), ...

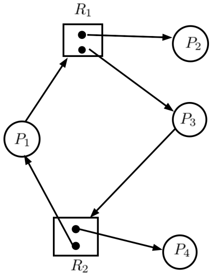

Figure 3: Resource allocation Graph with deadlock.

- There is a cycle: $P_1 \to R_1 \to P_3 \to R_2 \to P_1$, but there will not be dead lock. Why?

Methods for Handling Deadlocks

- We can deal with the deadlock problem in one of three ways:
  1. We can use a protocol to prevent or avoid deadlocks,
  2. We can allow the system to enter a deadlocked state, detect it, and recover.
  3. We can ignore the problem altogether
- To ensure that deadlocks never occur, the system can use either a *deadlock-prevention* or a *deadlock-avoidance* scheme.
- *Deadlock avoidance* requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime.
- The system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover
- Q. What is difference between prevention and avoidance of deadlock?

Deadlock Prevention

For a deadlock to occur, each of the four necessary conditions must hold.

- **1. Mutula Exclusion**. The mutual exclusion condition must hold.
- **2. Hold and Wait**. To ensure that the hold-and-wait condition never occurs in the system
  - An alternative protocol allows a process to request resources only when it has none.
  - To illustrate the difference between these two protocols, we consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.
  - The second method allows the process to request initially only the DVD drive and disk file.
  - Both these protocols have two main disadvantages. First, resource utilization may be low, since resources may be allocated but unused for a long period.
  - Starvation is possible. A process that needs several popular resources may have to wait indefinitely,