CS343 Operating Systems

Lecture 11

Introduction to Process Deadlocks

John Jose
Associate Professor
Department of Computer Science & Engineering
Indian Institute of Technology Guwahati

Objectives of Deadlock Management Unit

- To describe deadlocks, which prevent sets of concurrent processes from completing their tasks
- To discuss different methods for preventing or avoiding deadlocks in a computer system

System Model

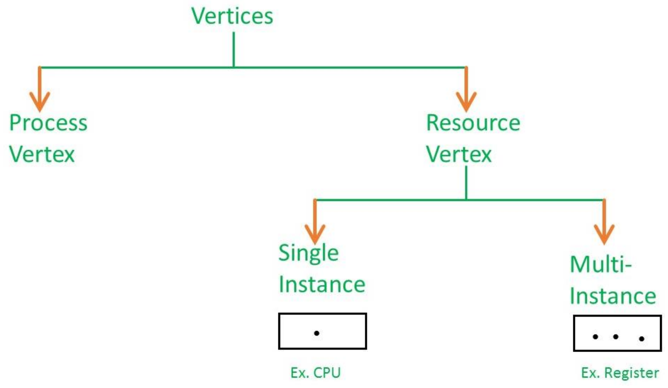
- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$
    - *CPU cycles, memory space, I/O devices*
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
    - **request**
    - **use**
    - **release**

Deadlock Characterization

- Deadlock can arise if the following four conditions hold simultaneously.
- **Mutual exclusion**: Only one process at a time can use a resource
- **Hold and wait**: A process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption**: A resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait**: There exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes 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.

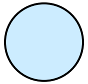
Resource-Allocation Graph

- A set of vertices *V* and a set of edges *E*.
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the active processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

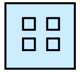


Resource-Allocation Graph

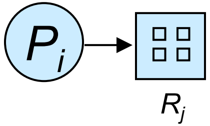
- Process



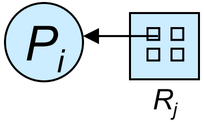
- Resource Type with 4 instances

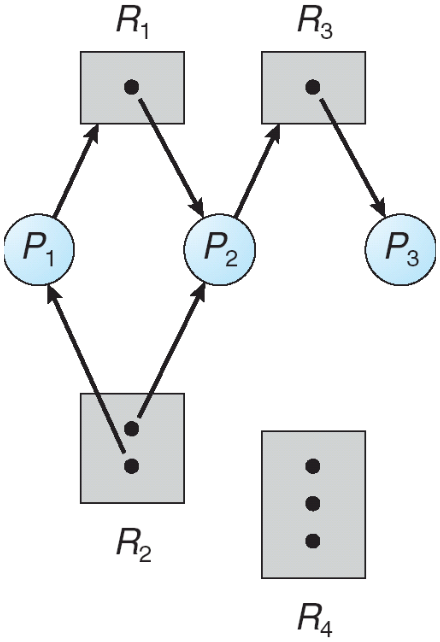


- P_i requests an instance of R_j

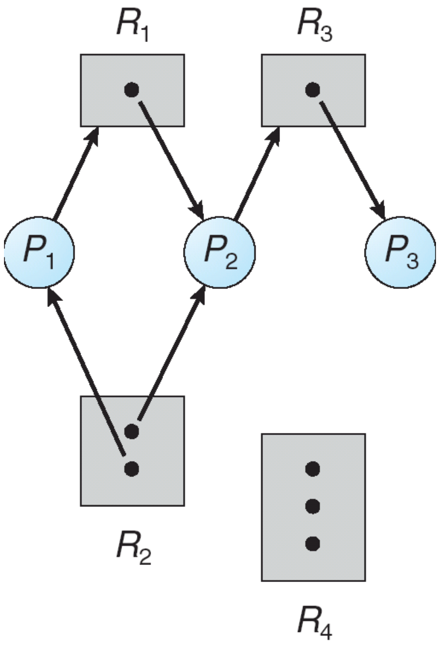


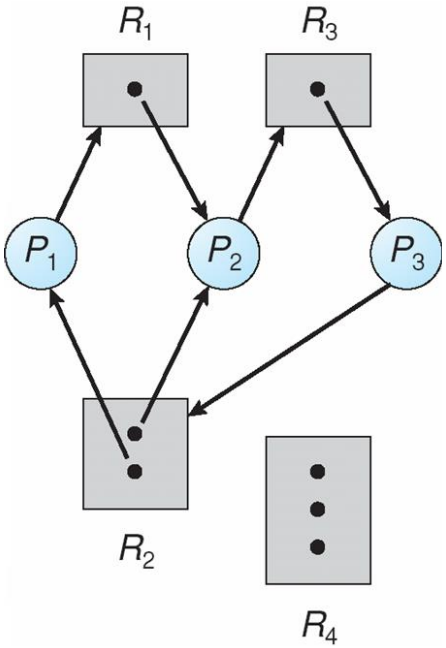
- P_i is holding an instance of R_j



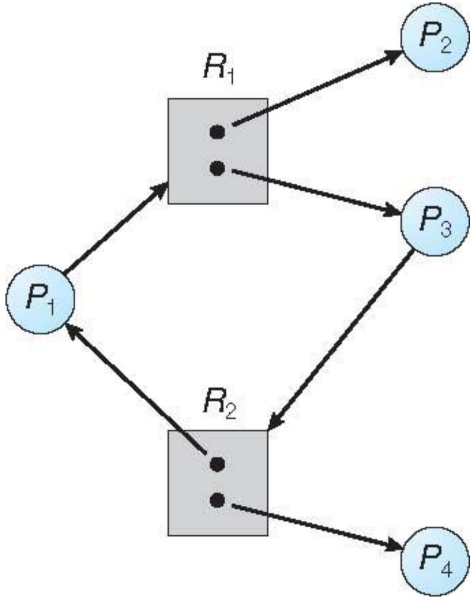


Resource-Allocation Graph





RAG with a deadlock

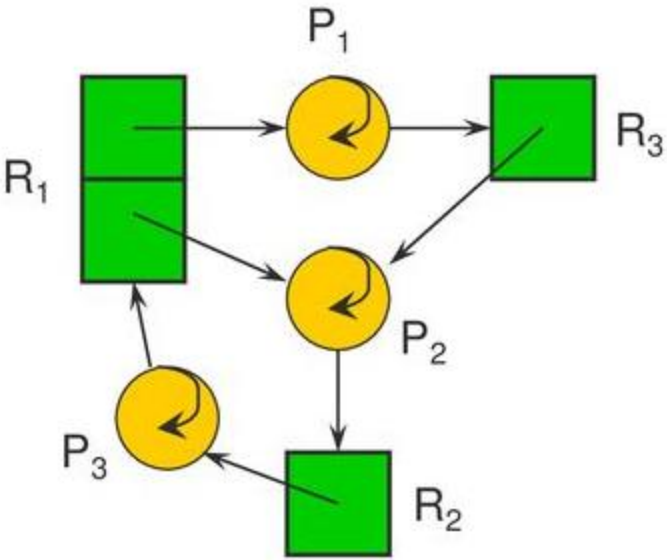


RAG without a deadlock

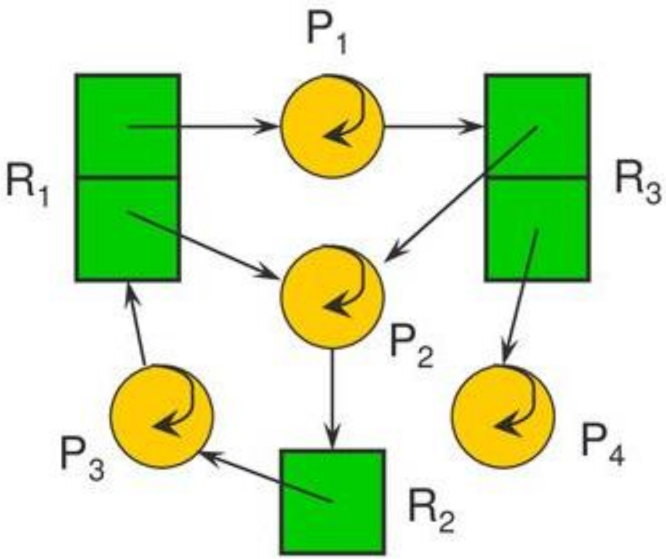
Deadlock detection in RAG

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Deadlock detection in RAG



A cycle...and deadlock!



Same cycle...but no deadlock. Why?

Methods for Handling Deadlocks

- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX
- Ensure that the system will **never** enter a deadlock state:
  - Deadlock prevention
  - Deadlock avoidance
- Allow the system to enter a deadlock state and then recover

Deadlock Prevention

- **Deadlock prevention is done by ensuring that at least one of the necessary 4 conditions for deadlock is not met.**
- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible

Deadlock Prevention

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

- Requires that the system has some additional ***a priori*** information available
- Simplest and most useful model requires that each process declare the ***maximum number*** of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

- System is in **safe state** if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < I$

- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/