OPERATING SYSTEMS

Deadlocks

Chandravva Hebbi

Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

**Deadlock Detection, Algorithm**

Chandravva Hebbi

Department of Computer Science

Deadlock Detection

- ☐ Allow system to enter deadlock state

- ☐ Detection algorithm

- ☐ Recovery scheme

Single Instance of Each Resource Type

- Maintain **wait-for** graph

  - Nodes are processes

  - $P_i \rightarrow P_j$ if P_i is waiting for P_j

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

(a)
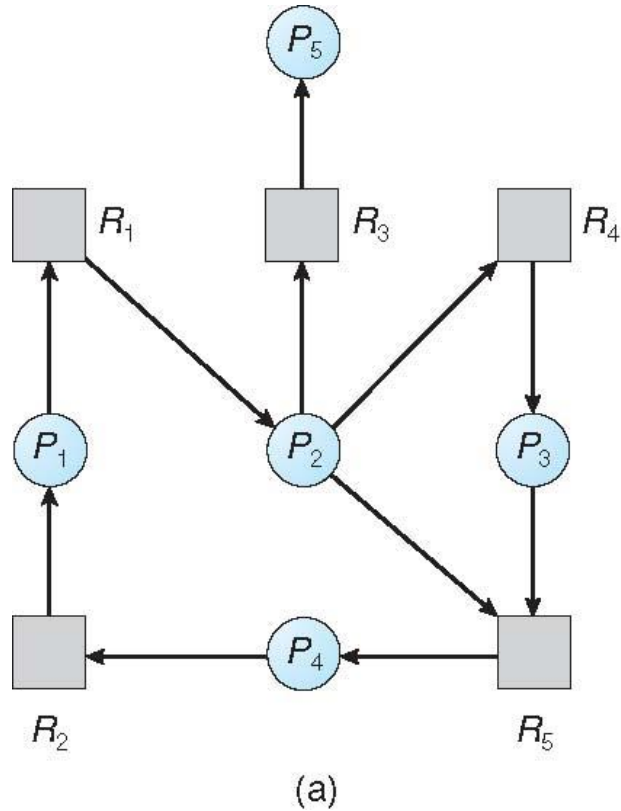
Resource-Allocation Graph

(b)
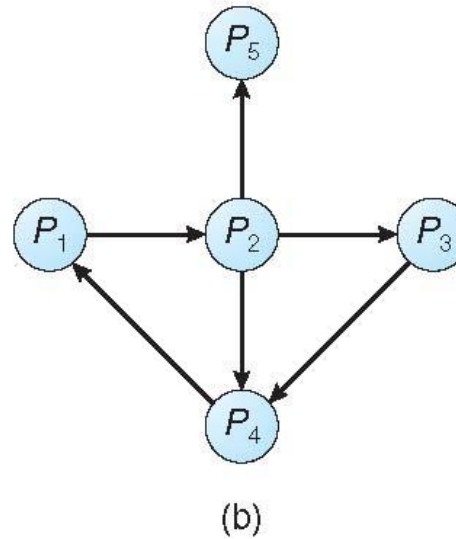
Corresponding wait-for graph

Several Instances of a Resource Type

- **Available***:* A vector of length *m* indicates the number of available resources of each type

- **Allocation***:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process

- **Request***:* An *n* x *m* matrix indicates the current request of each process. If **Request [*i*][*j*] = *k***, then process P_i is requesting *k* more instances of resource type R_j.

Detection Algorithm

1. Let **Work** and **Finish** be vectors of length **m** and **n**, respectively Initialize:

   (a) **Work = Available**

   (b) For **i = 1,2, ..., n**, if **Allocation$_i$ ≠ 0**, then
   Finish[i] **= false**; otherwise, **Finish**[i] **= true**

2. Find an index **i** such that both:

   (a) **Finish**[**i**] **== false**

   (b) **Request$_i$ ≤ Work**

   If no such **i** exists, go to step 4

Detection Algorithm (Cont.)

3. ***Work = Work + Allocation*** $_i$
   Finish*[*i*] = *true
   go to step 2

4. If ***Finish[i] == false***, for some ***i***, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if ***Finish*[*i*] == *false***, then ***P*** $_i$ is deadlocked

Algorithm requires an order of O(*m* x *n*^{2}) operations to detect whether the system is in deadlocked state

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types
  A (7 instances), B (2 instances), and C (6 instances)

- Snapshot at time T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in **Finish[i] = true** for all **i**

Example of Detection Algorithm (Cont.)

☐ P_2 requests an additional instance of type C

$$\underline{Request}$$

	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

☐ State of system?

☐ Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests

☐ Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

Detection-Algorithm Usage

- When, and how often, to invoke depends on:

    - How often a deadlock is likely to occur?

    - How many processes will need to be rolled back?

        ‣ one for each disjoint cycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

THANK YOU

Chandravva Hebbi

Department of Computer Science Engineering

chandravvahebbi@pes.edu