CSC 4320/6320: Operating Systems

Chapter 08: Deadlocks-III

Spring 2025

Instructor: Dr. Md Mahfuzur Rahman
Department of Computer Science, GSU

Disclaimer

The slides to be used in this course have been created, modified, and adapted from multiple sources:

- *The slides are copyright Silberschatz, Galvin and Gagne, 2018. The slides are authorized for personal use, and for use in conjunction with a course for which Operating System Concepts is the prescribed text. Instructors are free to modify the slides to their taste, as long as the modified slides acknowledge the source and the fact that they have been modified.*

Outline

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Review: Banker’s Algorithm

- Multiple instances of resources
- Each thread must a priori claim maximum use
- When a thread requests a resource, it may have to wait
- When a thread gets all its resources it must return them in a finite amount of time

Review: Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: n x m matrix. If *Max* $[i,j] = k$, then process T_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation$[i,j] = k$ then T_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If *Need*$[i,j] = k$, then T_i may need k more instances of R_j to complete its task

$$Need\,[i,j] = Max[i,j] - Allocation\,[i,j]$$

Review: Safety Algorithm

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

   Work = Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

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

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

   (b) ***Need*** $_i$ $\leq$ ***Work***

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

3. ***Work* = *Work* + *Allocation*** $_i$

   Finish*[*i*] = *true

   go to step 2

4. If ***Finish* [*i*] == *true*** for all ***i***, then the system is in a safe state

Review: Resource-Request Algorithm for Process P_i

$\textbf{\textit{Request}}_i$ = request vector for process T_i. If $\textbf{\textit{Request}}_i[j] = k$ then process T_i wants k instances of resource type R_j

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $Request_i \leq Available$, go to step 3. Otherwise, T_i must wait, since resources are not available
3. Pretend to allocate requested resources to T_i by modifying the state as follows:

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

- If safe $\Rightarrow$ the resources are allocated to T_i
- If unsafe $\Rightarrow$ T_i must wait, and the old resource-allocation state is restored

Review: Example of Banker's Algorithm

- 5 threads T_0 through T_4;
  3 resource types:
  A (10 instances), B (5instances), and C (7 instances)
- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
T_0	0 1 0	7 5 3	3 3 2
T_1	2 0 0	3 2 2	
T_2	3 0 2	9 0 2	
T_3	2 1 1	2 2 2	
T_4	0 0 2	4 3 3	

Review: Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	<u>*Need*</u>
	A B C
T_0	7 4 3
T_1	1 2 2
T_2	6 0 0
T_3	0 1 1
T_4	4 3 1

- The system is in a safe state since the sequence < T_1, T_3, T_4, T_2, T_0> satisfies safety criteria

Review: Example: P_1 Request (1,0,2)

- Suppose now that thread T_1 requests one additional instance of resource type A and two instances of resource type C
- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
T_0	0 1 0	7 4 3	2 3 0
T_1	3 0 2	0 2 0	
T_2	3 0 2	6 0 0	
T_3	2 1 1	0 1 1	
T_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < T_1, T_3, T_4, T_0, T_2> satisfies safety requirement
- Can request for (3,3,0) by T_4 be granted? (Are resources available?)
- Can request for (0,2,0) by T_0 be granted? (Does it keep the system safe?)

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

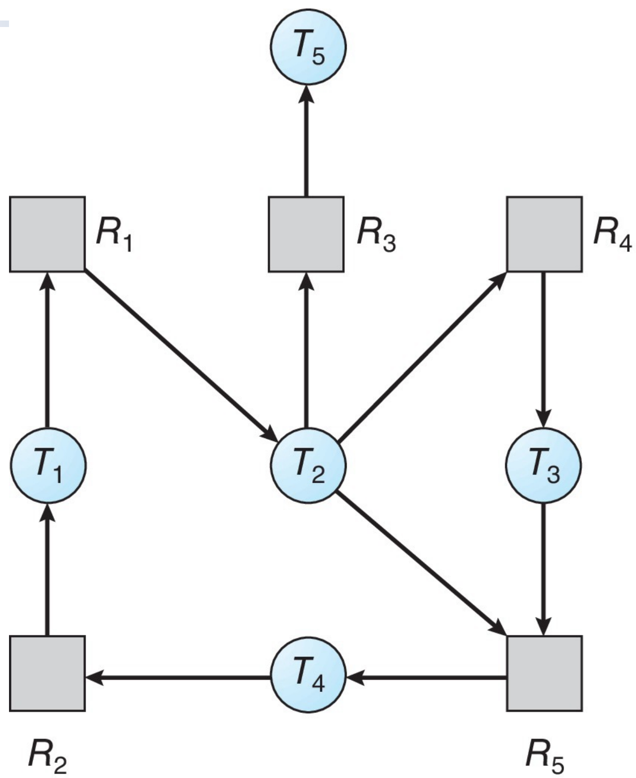
Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are threads
  - $T_i \rightarrow T_j$ if T_i is waiting for T_j

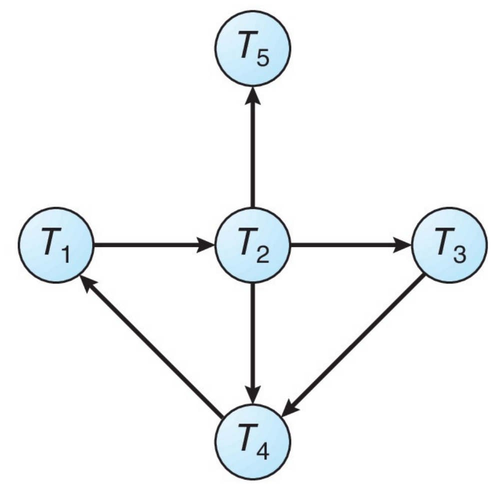
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

- 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

Resource-Allocation Graph and Wait-for Graph



(a)



(b)

Resource-Allocation Graph 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 thread.
- **Request***:* An ***n*** **x** ***m*** matrix indicates the current request of each thread. If ***Request*** **[*i*][*j*] = *k***, then thread T_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***$_i \neq$ **0**, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:
   a) ***Finish*[*i*] == *false***
   b) ***Request***$_i \leq$ ***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 T_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 threads T_0 through T_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*
T_0	0 1 0	0 0 0	0 0 0
T_1	2 0 0	2 0 2	
T_2	3 0 3	0 0 0	
T_3	2 1 1	1 0 0	
T_4	0 0 2	0 0 2	

- Sequence <T_0, T_2, T_3, T_1, T_4> will result in ***Finish[i] = true*** for all ***i***

Example (Cont.)

- T_2 requests an additional instance of type ***C***

	Request		
	A	*B*	*C*
T_0	0	0	0
T_1	2	0	2
T_2	0	0	1
T_3	1	0	0
T_4	0	0	2

- State of system?
  - Can reclaim resources held by thread T_0, but insufficient resources to fulfill other processes' requests
  - Deadlock exists, consisting of processes T_1, T_2, T_3, and T_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 threads “caused” the deadlock.

Recovery from Deadlock: Process Termination

- **Abort all deadlocked threads** (great expense.)
- **Abort one process at a time until the deadlock cycle is eliminated** (increased overhead)
- In which order should we choose to abort?
  - It's an economic decision (which incurs the minimum cost)
  1. Priority of the thread
  2. How long has the thread computed, and how much longer to completion
  3. Resources that the thread has used
  4. Resources that the thread needs to complete
  5. How many threads will need to be terminated
  6. Is the thread interactive or batch?

Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – goal is to minimize cost
- **Rollback** – return to some safe state, restart the thread for that state
- **Starvation** – same thread may always be picked as victim, include number of rollback in cost factor

Questions

1. A system can recover from a deadlock by
A) aborting one process at a time until the deadlock cycle is eliminated.
B) aborting all deadlocked processes.
C) preempting some resources from one or more of the deadlocked threads.
D) All of the above. ✓

2. To recover from a deadlock using resource preemption,
A) the order of resources and processes that need to be preempted must be determined to minimize cost.
B) if a resource is preempted from a process, the process must be rolled back to some safe state and restarted from that state.
C) ensure that starvation does not occur from always preempting resources from the same process.
D) All of the above. ✓

Questions

Q. Suppose that there are 12 resources available to three processes. At time 0, the following data is collected. The table indicates the process, the maximum number of resources needed by the process, and the number of resources currently owned by each process. Which of the following correctly characterizes this state?

Process	Maximum Needs	Currently Owned
P0	10	4
P1	3	2
P2	7	4