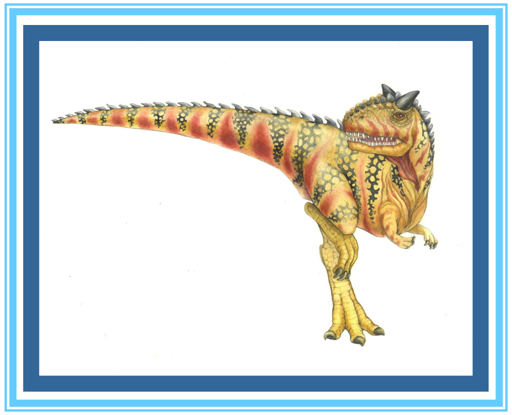
Chapter 7: Deadlocks



Chapter 7: Deadlocks

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

Recap

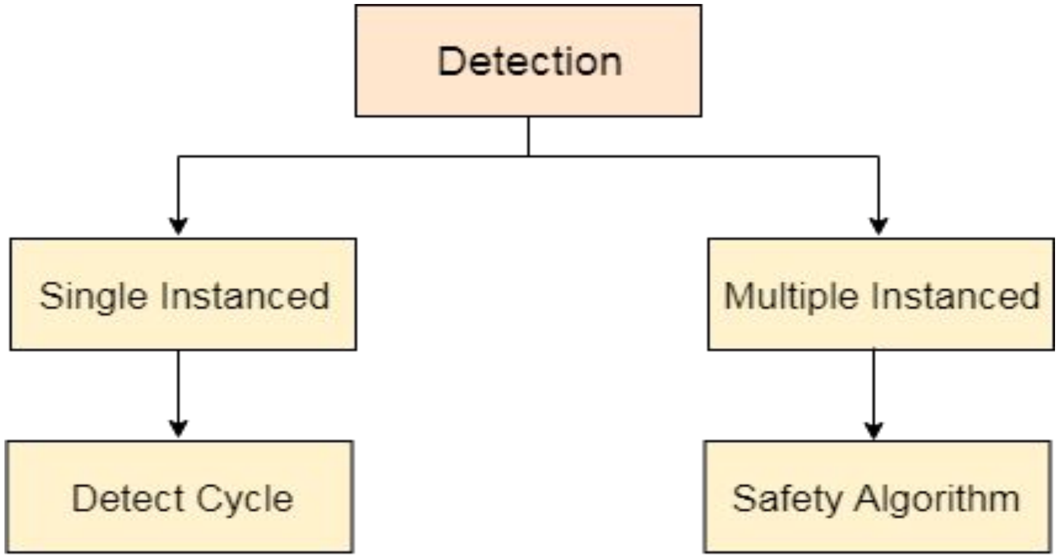
- Deadlock Characteristics
- Resource Allocation Graph
- Deadlock Prevention
- Deadlock Avoidance for Single Recourse Instance
- Deadlock Avoidance for Multiple Recourse Instances (Bankers Algorithm)

Objectives

- Deadlock Detection :
  - For Single instance of resource
  - For Multiple instances of recourse
- Deadlock Recovery

Deadlock Detection

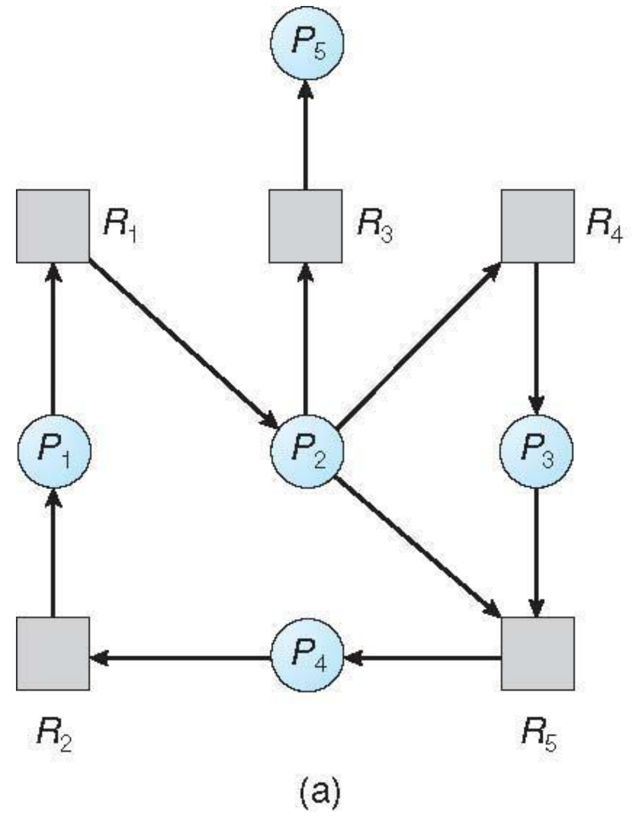
- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme



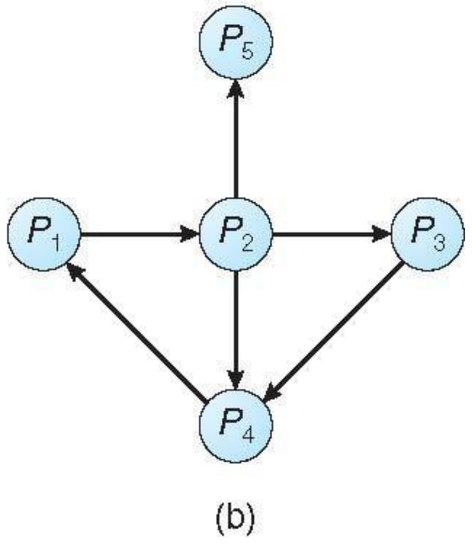
Single Instance of Each Resource Type

- 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

Resource-Allocation Graph and Wait-for Graph



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 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***, $1 \le i \le n$, 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 $<P_0, P_2, P_3, P_4, P_1>$ will result in ***Finish[i] = true*** for all ***i***

Example of Detection Algorithm

Safe Sequence < >

- n = 5 processes P_0 through P_4;
  m = 3 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	

```
Work = Available
 Finish[i] = False
 if Request ≤ Work
 Work = Work + Allocation
 Finish[i] = True
```

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F					

Example of Detection Algorithm

Safe Sequence < P_0 >

- n = 5 processes P_0 through P_4;
  m = 3 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	

Work = 0 0 0
Finish[0] = False
if *Request* ≤ *Work*
0 0 0 ≤ 0 0 0
Condition is TRUE
***Work* = 0 0 0 + 0 1 0= 0 1 0**
Finish[0] = True

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T				

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 resource types:
  A (7 instances), *B* (2 instances), and *C* (6 instances)

Safe Sequence < P_0 >

- Snapshot at time T_0:

	Allocation	*request*	*Available*
	A B C	*A B C*	*A B C*
P_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 1 0
Finish[1] = False
if *Request* ≤ *Work*
2 0 2 ≤ 0 1 0
Condition is FALSE
***Work* = 0 1 0**
Finish[1] = FALSE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F			

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 resource types:

  A (7 instances), *B* (2 instances), and *C* (6 instances)

Safe Sequence < P_0, P_2 >

- Snapshot at time T_0:

	Allocation	*request*	*Available*
	A B C	*A B C*	*A B C*
P_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 1 0
Finish[2] = False
if *Request* ≤ *Work*
0 0 0 ≤ 0 1 0
Condition is TRUE
***Work* = 0 1 0 +3 0 3= 3 1 3**
Finish[2] = TRUE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	T		

Example of Detection Algorithm

Safe Sequence < P_0 , P_2, P_3>

- n = 5 processes P_0 through P_4;
  m = 3 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_1	2 0 0	2 0 2	3 1 3
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 3 1 3
Finish[3] = False
if *Request ≤ Work*
1 0 0 ≤ 3 1 3
Condition is TRUE
***Work* = 3 1 3 + 2 1 1= 5 2 4**
Finish[3] = TRUE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	T	T	

Example of Detection Algorithm

Safe Sequence < P_0, P_2, P_3, P_4 >

- n = 5 processes P_0 through P_4;
  m = 3 resource types:
  A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T_0:

	<u>*Allocation*</u>	<u>*request*</u>	<u>*Available*</u>
	A B C	*A B C*	*A B C*
P_1	2 0 0	2 0 2	5 2 4
P_4	0 0 2	0 0 2	

Work = 5 2 4
Finish[4] = False
if *Request ≤ Work*
0 0 2 ≤ 5 2 4
Condition is TRUE
***Work* = 5 2 4 + 0 0 2= 5 2 6**
Finish[4] = TRUE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	T	T	T

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 resource types:
  A (7 instances), B (2 instances), and C (6 instances)

Safe Sequence < P_0, P_2, P_3, P_4, P_1 >

- Snapshot at time T_0:

	Allocation	*request*	*Available*
	A B C	*A B C*	*A B C*
P_1	2 0 0	2 0 2	5 2 6

Work = 5 2 6
Finish[4] = False
if *Request* ≤ *Work*
2 0 2 ≤ 5 2 6
Condition is TRUE
***Work* = 5 2 6 + 2 0 0= 7 2 6**
Finish[1] = TRUE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	T	T	T	T

Example (Cont.)

- P_2 requests an additional instance of type ***C***

	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?

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 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 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Safe Sequence < >

Work = Available
Finish[i] = False
if *Request ≤ Work*
Work = Work + Allocation
Finish[i] = True

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F					

Example of Detection Algorithm

Safe Sequence < P_0 >

- n = 5 processes P_0 through P_4;
  m = 3 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 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 0 0
Finish[0] = False
if *Request* ≤ *Work*
0 0 0 ≤ 0 0 0
Work = Work + Allocation
***work*= 0 0 0 + 0 1 0= 0 1 0**
Finish[0] = True

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T				

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 resource types:
  A (7 instances), *B* (2 instances), and *C* (6 instances)

Safe Sequence <P_0 >

- Snapshot at time T_0:

	Allocation	*request*	*Available*
	A B C	*A B C*	*A B C*
P_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 1 0
Finish[1] = False
if *Request* ≤ *Work*
2 0 2 ≤ 0 1 0
Condition is False
***work*= 0 1 0**
Finish[1] = FALSE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F			

Example of Detection Algorithm

Safe Sequence < P_0 >

- n = 5 processes P_0 through P_4;
  m = 3 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_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 1 0
Finish[2] = False
if *Request* ≤ *Work*
0 0 1 ≤ 0 1 0
Condition is False
***work*= 0 1 0**
Finish[2] = FALSE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	F		

Example of Detection Algorithm

Safe Sequence < P_0 >

- n = 5 processes P_0 through P_4;
  m = 3 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_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Work = 0 1 0
Finish[3] = False
if *Request* ≤ *Work*
1 0 0 ≤ 0 1 0
Condition is False
***work*= 0 1 0**
Finish[3] = FALSE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	F	F	

Example of Detection Algorithm

- n = 5 processes P_0 through P_4;
  m = 3 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_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Safe Sequence < P_0 >

Work = 0 1 0
Finish[4] = False
if *Request* ≤ *Work*
0 0 2 ≤ 0 1 0
Condition is False
***work*= 0 1 0**
Finish[4] = FALSE

Finish[P_i]	P_0	P_1	P_2	P_3	P_4
T/F	T	F	F	F	F

Example (Cont.)

- P_2 requests an additional instance of type ***C***

	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

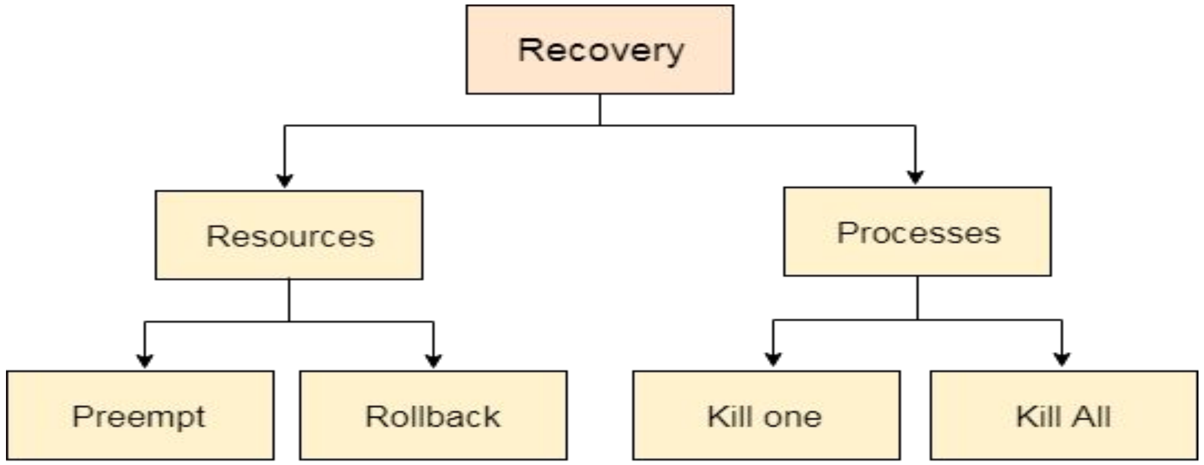
- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

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.

Recovery from Deadlock: Process Termination

- **Abort all the Deadlocked Processes:**
  Aborting all the processes will certainly break the deadlock, but with a great expenses. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.
- **Abort one process at a time until deadlock is eliminated:**
  Abort one deadlocked process at a time, until deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we must run deadlock detection algorithm to check whether any processes are still deadlocked.



Recovery from Deadlock: Process Termination

- In which order should we choose to abort?
- Many factors mat affect which process is chosen, including:
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- **Selecting a victim:**
  We must determine which resources and which processes are to be preempted and the order to minimize the cost.
- **Rollback:**
  We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.
- **Starvation:**
  In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

End of Chapter 7