Chapter 7 – Deadlock and Indefinite Postponement

Outline

Chapter 7 – Deadlock and Indefinite Postponement

Outline (continued)

Objectives

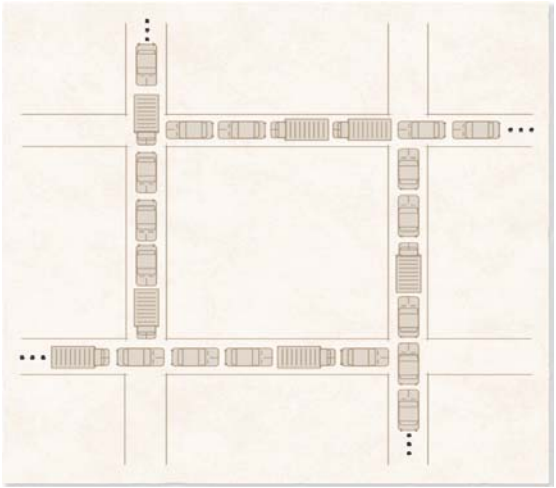
- After reading this chapter, you should understand:
  - the problem of deadlock.
  - the four necessary conditions for deadlock to exist.
  - the problem of indefinite postponement.
  - the notions of deadlock prevention, avoidance, detection and recovery.
  - algorithms for deadlock avoidance and detection.
  - how systems can recover from deadlocks.

7.1 Introduction

- Deadlock
  - A process or thread is waiting for a particular event that will not occur
- System deadlock
  - One or more processes are deadlocked

7.2.1 Traffic Deadlock

Figure 7.1 Traffic deadlock example.

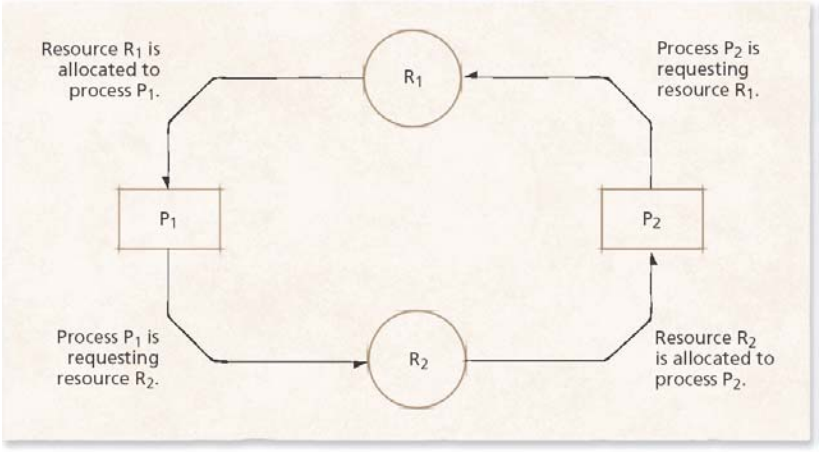


7.2.2 Simple Resource Deadlock

- Most deadlocks develop because of the normal contention for dedicated resources
- Circular wait is characteristic of deadlocked systems

7.2.2 Simple Resource Deadlock

Figure 7.2 Resource deadlock example. This system is deadlocked because each process holds a resource being requested by the other process and neither process is willing to release the resource it holds.



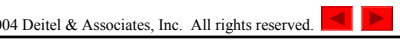
7.2.3 Deadlock in Spooling Systems

- Spooling systems are prone to deadlock
- Common solution
  - Restrain input spoolers so that when the spooling file begins to reach some saturation threshold, the spoolers do not read in more print jobs
- Today's systems
  - Printing begins before the job is completed so that a full spooling file can be emptied even while a job is still executing
  - Same concept has been applied to streaming audio and video

7.2.4 Example: Dining Philosophers

- Problem statement:

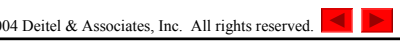
  Five philosophers sit around a circular table. Each leads a simple life alternating between thinking and eating spaghetti. In front of each philosopher is a dish of spaghetti that is constantly replenished by a dedicated wait staff. There are exactly five forks on the table, one between each adjacent pair of philosophers. Eating spaghetti (in the most proper manner) requires that a philosopher use both adjacent forks (simultaneously). Develop a concurrent program free of deadlock and indefinite postponement that models the activities of the philosophers.



7.2.4 Example: Dining Philosophers

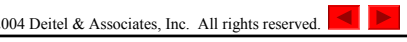
Figure 7.3 Dining philosopher behavior.

```
1  void typicalPhilosopher()
2  {
3     while ( true )
4     {
5        think();
6        eat();
7     } // end while
8
9  } // end typicalPhilospher
```

7.2.4 Example: Dining Philosophers

- Constraints:
  - To prevent philosophers from starving:
    - Free of deadlock
    - Free of indefinite postponement
  - Enforce mutual exclusion
    - Two philosophers cannot use the same fork at once
- The problems of mutual exclusion, deadlock and indefinite postponement lie in the implementation of method eat.



7.2.4 Example: Dining Philosophers

Figure 7.4 Implementation of method `eat`.

```
1  void eat()
2  {
3     pickUpLeftFork();
4     pickUpRightFork();
5     eatForSomeTime();
6     putDownRightFork();
7     putDownLeftFork();
8  } // eat
```


7.3 Related Problem: Indefinite Postponement

- Indefinite postponement
  - Also called indefinite blocking or starvation
  - Occurs due to biases in a system's resource scheduling policies
- Aging
  - Technique that prevents indefinite postponement by increasing process's priority as it waits for resource

7.4 Resource Concepts

- Preemptible resources (e.g. processors and main memory)
  - Can be removed from a process without loss of work
- Nonpreemptible resources (e.g. tape drives and optical scanners)
  - Cannot be removed from the processes to which they are assigned without loss of work
- Reentrant code
  - Cannot be changed while in use
  - May be shared by several processes simultaneously
- Serially reusable code
  - May be changed but is reinitialized each time it is used
  - May be used by only one process at a time

7.5 Four Necessary Conditions for Deadlock

- Mutual exclusion condition
  - Resource may be acquired exclusively by only one process at a time
- Wait-for condition (hold-and-wait condition)
  - Process that has acquired an exclusive resource may hold that resource while the process waits to obtain other resources
- No-preemption condition
  - Once a process has obtained a resource, the system cannot remove it from the process's control until the process has finished using the resource
- Circular-wait condition
  - Two or more processes are locked in a "circular chain" in which each process is waiting for one or more resources that the next process in the chain is holding

7.6 Deadlock Solutions

- Four major areas of interest in deadlock research
  - Deadlock prevention
  - Deadlock avoidance
  - Deadlock detection
  - Deadlock recovery

7.7 Deadlock Prevention

- Deadlock prevention
  - Condition a system to remove any possibility of deadlocks occurring
  - Deadlock cannot occur if any one of the four necessary conditions is denied
  - First condition (mutual exclusion) cannot be broken

7.7.1 Denying the "Wait-For" Condition

- When denying the "wait-for condition"
  - All of the resources a process needs to complete its task must be requested at once
  - This leads to inefficient resource allocation

7.7.2 Denying the “No-Preemption” Condition

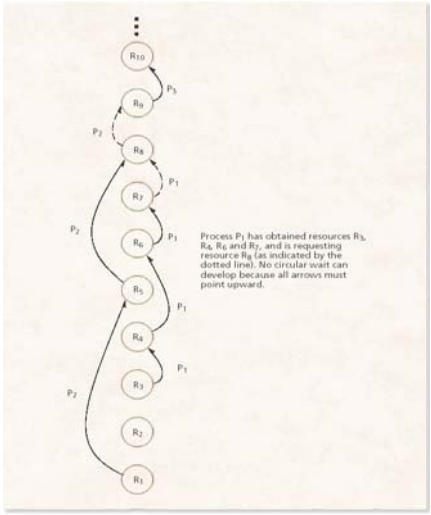
- When denying the “no-preemption” condition
  - Processes may lose work when resources are preempted
  - This can lead to substantial overhead as processes must be restarted

7.7.3 Denying the “Circular-Wait” Condition

- Denying the “circular-wait” condition:
  - Uses a linear ordering of resources to prevent deadlock
  - More efficient resource utilization than the other strategies
- Drawbacks
  - Not as flexible or dynamic as desired
  - Requires the programmer to determine the ordering or resources for each system

7.7.3 Denying the "Circular-Wait" Condition

Figure 7.5 Havender's linear ordering of resources for preventing deadlock.



7.8 Deadlock Avoidance with Dijkstra's Banker's Algorithm

- Banker's Algorithm
  - Impose less stringent conditions than in deadlock prevention in an attempt to get better resource utilization
  - Safe state
    - Operating system can guarantee that all current processes can complete their work within a finite time
  - Unsafe state
    - Does not imply that the system is deadlocked, but that the OS cannot guarantee that all current processes can complete their work within a finite time

7.8 Deadlock Avoidance with Dijkstra's Banker's Algorithm

- Banker's Algorithm (cont.)
  - Requires that resources be allocated to processes only when the allocations result in safe states.
  - It has a number of weaknesses (such as requiring a fixed number of processes and resources) that prevent it from being implemented in real systems

7.8.2 Example of an Unsafe State

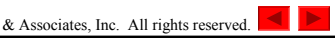
Figure 7.6 Safe state.

Process	max(P_i) (maximum need)	loan(P_i) (current loan)	claim(P_i) (current claim)
P_1	4	1	3
P_2	6	4	2
P_3	8	5	3
Total resources, t, = 12		Available resources, a, = 2	

7.8.2 Example of an Unsafe State

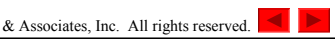
Figure 7.7 Unsafe state.

Process	max(P_i) (maximum need)	loan(P_i) (current loan)	claim(P_i) (current claim)
P_1	10	8	2
P_2	5	2	3
P_3	3	1	2
Total resources, t, = 12		Available resources, a, = 1	



7.8.3 Example of Safe-State-to-Unsafe-State Transition

- Safe-state-to-unsafe-state transition:
  - Suppose the current state of a system is safe, as shown in Fig. 7.6.
  - The current value of *a* is 2.
  - Now suppose that process P_3 requests an additional resource



7.8.3 Example of Safe-State-to-Unsafe-State Transition

Figure 7.8 Safe-state-to-unsafe-state transition.

Process	max(P_i) (maximum need)	loan(P_i) (current loan)	claim(P_i) (current claim)
P_1	4	1	3
P_2	6	4	2
P_3	8	6	2
Total resources, t, = 12		Available resources, a, = 1	

7.8.4 Banker's Algorithm Resource Allocation

- Is the state in the next slide safe?

7.8.4 Banker's Algorithm Resource Allocation

Figure 7.9 State description of three processes.

Process	max(P,)	loan(P,)	claim(P,)
P_1	5	1	4
P_2	3	1	2
P_3	10	5	5
		a = 2	

7.8.4 Banker's Algorithm Resource Allocation

- Answer:
  - There is no guarantee that all of these processes will finish
    - P_2 will be able to finish by using up the two remaining resources
    - Once P_2 is done, there are only three available resources left
    - This is not enough to satisfy either P_1's claim of 4 or P_3's claim of five

7.8.5 Weaknesses in the Banker's Algorithm

- Weaknesses
  - Requires there be a fixed number of resource to allocate
  - Requires the population of processes to be fixed
  - Requires the banker to grant all requests within "finite time"
  - Requires that clients repay all loans within "finite time"
  - Requires processes to state maximum needs in advance

7.9 Deadlock Detection

- Deadlock detection
  - Used in systems in which deadlocks can occur
  - Determines if deadlock has occurred
  - Identifies those processes and resources involved in the deadlock
  - Deadlock detection algorithms can incur significant runtime overhead

7.9.1 Resource-Allocation Graphs

- Resource-allocation graphs
  - Squares
    - Represent processes
  - Large circles
    - Represent classes of identical resources
  - Small circles drawn inside large circles
    - Indicate separate identical resources of each class

7.9.1 Resource-Allocation Graphs

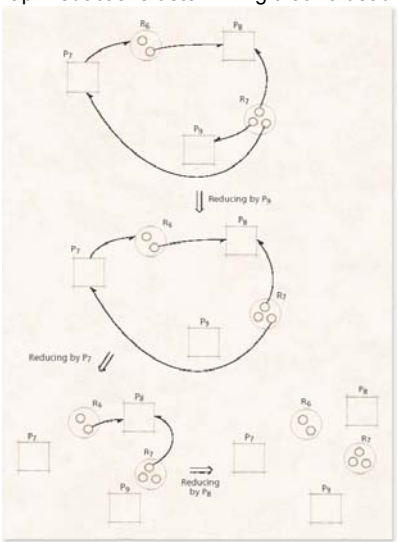
Figure 7.10 Resource-allocation and request graphs.

7.9.2 Reduction of Resource-Allocation Graphs

- Graph reductions
  - If a process's resource requests may be granted, the graph may be reduced by that process
  - If a graph can be reduced by all its processes, there is no deadlock
  - If a graph cannot be reduced by all its processes, the irreducible processes constitute the set of deadlocked processes in the graph

7.9.2 Reduction of Resource-Allocation Graphs

Figure 7.11 Graph reductions determining that no deadlock exists.



7.10 Deadlock Recovery

- Deadlock recovery
  - Clears deadlocks from system so that deadlocked processes may complete their execution and free their resources
- Suspend/resume mechanism
  - Allows system to put a temporary hold on a process
  - Suspended processes can be resumed without loss of work
- Checkpoint/rollback
  - Facilitates suspend/resume capabilities
  - Limits the loss of work to the time the last checkpoint was made

7.11 Deadlock Strategies in Current and Future Systems

- Deadlock is viewed as limited annoyance in personal computer systems
  - Some systems implement basic prevention methods suggested by Havender
  - Some others ignore the problem, because checking deadlocks would reduce systems' performance
- Deadlock continues to be an important research area