Understanding Operating Systems
Sixth Edition

Chapter 5
Process Management

Learning Objectives

After completing this chapter, you should be able to describe:

- Several causes of system deadlock and livelock
- The difference between preventing and avoiding deadlocks
- How to detect and recover from deadlocks

Learning Objectives (cont'd.)

- The concept of process starvation and how to detect and recover from it
- The concept of a race and how to prevent it
- The difference between deadlock, starvation, and race

Deadlock

- Resource sharing
  - Memory management and processor sharing
- Many programs competing for limited resources
- Lack of **process synchronization** consequences
  - **Deadlock:** "deadly embrace"
    - Two or more jobs placed in HOLD state
    - Jobs waiting for unavailable vital resource
    - System comes to standstill
    - Resolved via external intervention
  - **Starvation**
    - Infinite postponement of job

Deadlock

Deadlock (cont'd.)

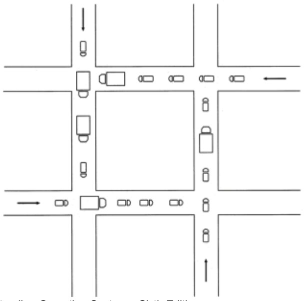
- More serious than starvation
- Affects entire system
  - Affects more than one job
    - Not just a few programs
  - All system resources become unavailable
- Example: traffic jam (Figure 5.1)
- More prevalent in interactive systems
- Real-time systems
  - Deadlocks quickly become critical situations
- No simple and immediate solution

Deadlock (cont'd.)

7

(figure 5.1)

A classic case of traffic deadlock. This is "gridlock," where no vehicles can move forward to clear the traffic jam.



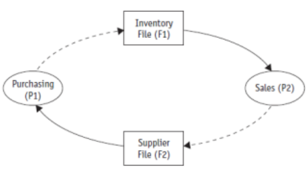
Seven Cases of Deadlock

8

- Nonsharable/nonpreemptable resources
  - Allocated to jobs requiring same type of resources
- Resource types locked by competing jobs
  - File requests
  - Databases
  - Dedicated device allocation
  - Multiple device allocation
  - Spooling
  - Network
  - Disk sharing

Case 1: Deadlocks on File Requests (cont'd.)

9



(figure 5.2)

Case 1. These two processes, shown as circles, are each waiting for a resource, shown as rectangles, that has already been allocated to the other process, thus creating a deadlock.

Case 1: Deadlocks on File Requests

10

- Jobs request and hold files for execution duration
- Example (Figure 5.2)
  - Two programs (P1, P2) and two files (F1, F2)
  - Deadlock sequence
    - P1 has access to F1 and also requires F2
    - P2 has access to F2 and also requires F1
  - Deadlock remains
    - Until one program withdrawn *or*
    - Until one program forcibly removed and file released
  - Other programs requiring F1 or F2
    - Put on hold for duration of situation

Case 2: Deadlocks in Databases

11

- Two processes access and lock database records
- **Locking**
  - Technique
    - One user locks out all other users
    - Users working with database
  - Three locking levels
    - Entire database for duration of request
    - Subsection of database
    - Individual record until request completed

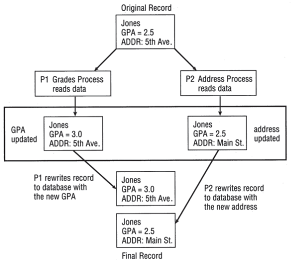
Case 2: Deadlocks in Databases (cont'd.)

12

- Example: two processes (P1 and P2)
  - Each needs to update two records (R1 and R2)
  - Deadlock sequence
    - P1 accesses R1 and locks it
    - P2 accesses R2 and locks it
    - P1 requests R2 but locked by P2
    - P2 requests R1 but locked by P1
- **Race between processes**
  - Results when locking not used
  - Causes incorrect final version of data
  - Depends on process execution order

Case 2: Deadlocks in Databases (cont'd.)

13



(figure 5.3)

Case 2. P1 finishes first and wins the race but its version of the record will soon be overwritten by P2. Regardless of which process wins the race, the final version of the data will be incorrect.

Case 3: Deadlocks in Dedicated Device Allocation

14

- Limited number of dedicated devices
- Example
  - Two programs (P1, P2)
    - Need two tape drives each
    - Only two tape drives in system
  - Deadlock sequence
    - P1 requests tape drive 1 and gets it
    - P2 requests tape drive 2 and gets it
    - P1 requests tape drive 2 but blocked
    - P2 requests tape drive 1 but blocked

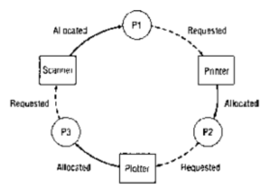
Case 4: Deadlocks in Multiple Device Allocation

15

- Several processes request and hold dedicated devices
- Example (Figure 5.4)
  - Three programs (P1, P2, P3)
  - Three dedicated devices (tape drive, printer, plotter)
  - Deadlock sequence
    - P1 requests and gets tape drive
    - P2 requests and gets printer
    - P3 requests and gets the plotter
    - P1 requests printer but blocked
    - P2 requests plotter but blocked
    - P3 requests tape drive but blocked

Case 4: Deadlocks in Multiple Device Allocation (cont'd.)

16



(figure 5.4)

Case 4. Three processes, shown as circles, are each waiting for a device that has already been allocated to another process, thus creating a deadlock.

Case 5: Deadlocks in Spooling

17

- **Virtual device**
  - Dedicated device made sharable
  - Example
    - Printer: high-speed disk device between printer and CPU
- **Spooling**
  - Process
    - Disk accepts output from several users
    - Acts as temporary storage for output
    - Output resides in disk until printer accepts job data

Case 5: Deadlocks in Spooling (cont'd.)

18

- Deadlock sequence
  - Printer needs all job output before printing begins
    - Spooling system fills disk space area
    - No one job has entire print output in spool area
    - Results in partially completed output for all jobs
    - Results in deadlock

Case 6: Deadlocks in a Network

19

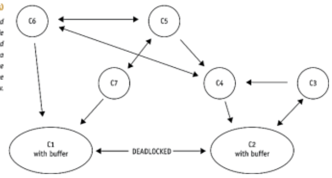
- No network protocols controlling network message flow
- Example (Figure 5.5)
  - Seven computers on network
    - Each on different nodes
  - Direction of arrows
    - Indicates message flow
  - Deadlock sequence
    - All available buffer space fills

Case 6: Deadlocks in a Network (cont'd.)

20

(figure 5.5)

Case 6, deadlocked network flow. Each circle represents a node and each line represents a communication path. The arrows indicate the direction of data flow.



C1: Spool is full with messages sent by C6 and C7 to C2 and cannot receive any more messages. Also cannot send to C2 since C2's spool is full.

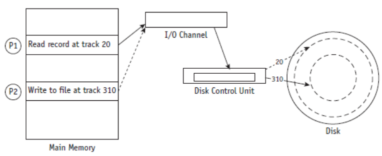
Case 7: Deadlocks in Disk Sharing

21

- Competing processes send conflicting commands
  - Scenario: disk access
- Example (Figure 5.6)
  - Two processes
  - Each process waiting for I/O request
    - One at cylinder 20 and one at cylinder 310
  - Deadlock sequence
    - Neither I/O request satisfied
    - Device puts request on hold while attempting to fulfill other request for each request
  - **Livelock** results

Case 7: Deadlocks in Disk Sharing (cont'd.)

22



(figure 5.6)

Case 7. Two processes are each waiting for an I/O request to be filled: one at track 20 and one at track 310. But by the time the read/write arm reaches one track, a competing command for the other track has been issued, so neither command is satisfied and livelock occurs.

P1: requests read at cylinder 20, and hold while the arm is moving.
P2: requests write at cylinder 310, and the request is accepted since P1 is on hold.
So P2 is on hold while the arm is moving toward cylinder 310.

Conditions for Deadlock

23

- Four conditions simultaneously occurring prior to deadlock or livelock
  - Mutual exclusion
  - Resource holding
  - No preemption
  - Circular wait
- All needed by operating system
  - Must recognize simultaneous occurrence of four conditions
- Resolving deadlock
  - Removal of one condition

Conditions for Deadlock (cont'd.)

24

- **Mutual exclusion**
  - Allowing only one process access to dedicated resource
  - Example: a step can hold only one person
- **Resource holding**
  - Holding resource and not releasing it
  - Waiting for other job to retreat
  - Example: Two people meet on the stairs none retreat
- **No preemption**
  - Lack of temporary reallocation of resources

Conditions for Deadlock (cont'd.)

25

- **Circular wait**
  - Each process involved in impasse
    - Waiting voluntarily resource release by another so at least one can continue
  - Example: Each person is waiting for another to voluntarily release the step
- All four required for deadlock occurrence
- Deadlock remains until one condition removed

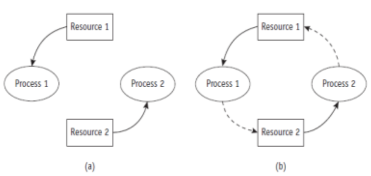
Modeling Deadlocks

26

- **Directed graphs**
  - Circles represent processes
  - Squares represent resources
  - Solid arrow from resource to process
    - Process holding resource
  - Dashed arrow from a process to resource
    - Process waiting for resource
  - Arrow direction indicates flow
  - If there is a cycle in graph
    - Deadlock involving processes and resources

Modeling Deadlocks (cont'd.)

27



(figure 5.7)

In (a), Resource 1 is being held by Process 1 and Resource 2 is held by Process 2 in a system that is not deadlocked. In (b), Process 1 requests Resource 2 but doesn't release Resource 1, and Process 2 does the same — creating a deadlock. (If one process released its resource, the deadlock would be resolved.)

Modeling Deadlocks (cont'd.)

28

- Three graph scenarios to help detect deadlocks
  - System has three processes (P1, P2, P3)
  - System has three resources (R1, R2, R3)
- Scenario one: no deadlock
  - Resources released before next process request
- Scenario two: deadlock
  - Processes waiting for resource held by another
- Scenario three: no deadlock
  - Resources released before deadlock

Modeling Deadlocks (cont'd.)

29

- No deadlock
  - Resources released before next process request

Event	Action
1	P1 requests and is allocated the printer R1.
2	P1 releases the printer R1.
3	P2 requests and is allocated the disk drive R2.
4	P2 releases the disk R2.
5	P3 requests and is allocated the plotter R3.
6	P3 releases the plotter R3.

(table 5.1)

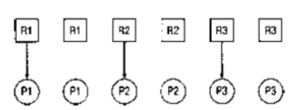
First scenario's sequence of events is shown in the directed graph in Figure 5.8.

Modeling Deadlocks (cont'd.)

30

(figure 5.8)

First scenario. The system will stay free of deadlocks if each resource is released before it is requested by the next process.



Modeling Deadlocks (cont'd.)

31

- Deadlock
  - Processes waiting for resource held by another

(table 5.2)

The second scenario's sequence of events is shown in the two directed graphs shown in Figure 5.9.

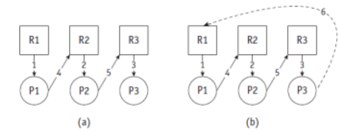
Event	Action
1	P1 requests and is allocated R1.
2	P2 requests and is allocated R2.
3	P3 requests and is allocated R3.
4	P1 requests R2.
5	P2 requests R3.
6	P3 requests R1.

Modeling Deadlocks (cont'd.)

32

(figure 5.9)

Second scenario. The system (a) becomes deadlocked (b) when P3 requests R1. Notice the circular wait.



Modeling Deadlocks (cont'd.)

33

- No deadlock
  - Resources released before deadlock

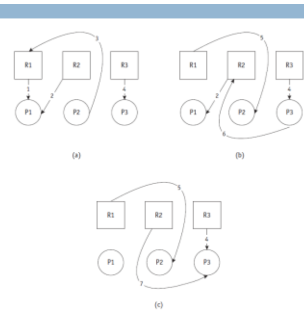
Event	Action
1	P1 requests and is allocated R1.
2	P1 requests and is allocated R2.
3	P2 requests R1.
4	P3 requests and is allocated R3.
5	P1 releases R1, which is allocated to P2.
6	P3 requests R2.
7	P1 releases R2, which is allocated to P3.

(table 5.3)

The third scenario's sequence of events is shown in the directed graph in Figure 5.10.

Modeling Deadlocks (cont'd.)

34



(figure 5.10)

The third scenario. After event 4, the directed graph looks like (a) and P2 is blocked because P1 is holding on to R1. However, event 5 breaks the deadlock and the graph soon looks like (b). Again there is a blocked process, P3, which must wait for the release of R2 in event 7 when the graph looks like (c).

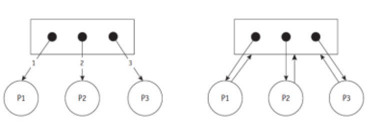
Modeling Deadlocks (cont'd.)

35

- Another example
  - Resources of same type
  - Allocated individually or grouped in same process
    - Graph clusters devices into one entity

(figure 5.11)

(a): A fully allocated cluster of resources. There are as many lines coming out of it as there are resources, units, in it. The state of (a) is uncertain because a request for another unit by all three processes would create a deadlock as shown in (b).



Strategies for Handling Deadlocks

37

- **Prevention**
  - Prevent occurrence of one condition
    - Mutual exclusion, resource holding, no preemption, circular wait
- **Avoidance**
  - Avoid deadlock if it becomes probable
- **Detection**
  - Detect deadlock when it occurs
  - Recover gracefully
- **Recovery**
  - Resume system normalcy quickly and gracefully

Strategies for Handling Deadlocks (cont'd.)

38

- **Prevention** eliminates one of four conditions
  - Complication: every resource cannot be eliminated from every condition
  - Mutual exclusion
    - Some resources must allocate exclusively
    - Bypassed if I/O device uses spooling
  - Resource holding
    - Bypassed if jobs request every necessary resource at creation time
    - Multiprogramming degree significantly decreased
    - Idle peripheral devices

Strategies for Handling Deadlocks (cont'd.)

39

- Prevention (cont'd.)
  - No preemption
    - Bypassed if operating system allowed to deallocate resources from jobs
    - Okay if job state easily saved and restored
    - Not accepted to preempt dedicated I/O device or files during modification
  - Circular wait
    - Bypassed if operating system prevents circle formation
    - Use hierarchical ordering scheme
    - Requires jobs to anticipate resource request order
    - Difficult to satisfy all users

Strategies for Handling Deadlocks (cont'd.)

40

- **Avoidance:** use if condition cannot be removed
  - System knows ahead of time
    - Sequence of requests associated with each active process
- Dijkstra's Bankers Algorithm
  - Regulates resources allocation to avoid deadlock
    - No customer granted loan exceeding bank's total capital
    - All customers given maximum credit limit
    - No customer allowed to borrow over limit
    - Sum of all loans will not exceed bank's total capital

Strategies for Handling Deadlocks (cont'd.)

41

(table 5.4)

The bank started with $10,000 and has remaining capital of $4,000 after these loans. Therefore it's in a "safe state."

Customer	Loan amount	Maximum credit	Remaining credit
C1	0	4,000	4,000
C2	2,000	5,000	3,000
C3	4,000	8,000	4,000
Total loaned: $6,000			
Total capital fund: $10,000			

Strategies for Handling Deadlocks (cont'd.)

42

Customer	Loan Amount	Maximum Credit	Remaining Credit
C1	2,000	4,000	2,000
C2	3,000	5,000	2,000
C3	4,000	8,000	4,000
Total loaned: $9,000			
Total capital fund: $10,000			

(table 5.5)

The bank only has remaining capital of $1,000 after these loans and therefore is in an "unsafe state."

Strategies for Handling Deadlocks (cont'd.)

43

Job No.	Devices Allocated	Maximum Required	Remaining Needs
1	0	4	4
2	2	5	3
3	4	8	4
Total number of devices allocated: 6			
Total number of devices in system: 10			

(table 5.6)

Resource assignments after initial allocations. A safe state: Six devices are allocated and four units are still available.

Strategies for Handling Deadlocks (cont'd.)

44

(table 5.7)

Resource assignments after later allocations. An unsafe state: Only one unit is available but every job requires at least two to complete its execution.

Job No.	Devices Allocated	Maximum Required	Remaining Needs
1	2	4	2
2	3	5	2
3	4	8	4
Total number of devices allocated: 9			
Total number of devices in system: 10			

Strategies for Handling Deadlocks (cont'd.)

45

- Operating systems deadlock avoidance assurances
  - Never satisfy request if job state moves from safe to unsafe
    - Identify job with smallest number of remaining resources
    - Number of available resources => number needed for selected job to complete
    - Block request jeopardizing safe state

Strategies for Handling Deadlocks (cont'd.)

46

- Problems with the Banker's Algorithm
  - Jobs must state maximum number needed resources
  - Requires constant number of total resources for each class
  - Number of jobs must remain fixed
  - Possible high overhead cost incurred
  - Resources not well utilized
    - Algorithm assumes worst case
  - Scheduling suffers
    - Result of poor utilization
    - Jobs kept waiting for resource allocation

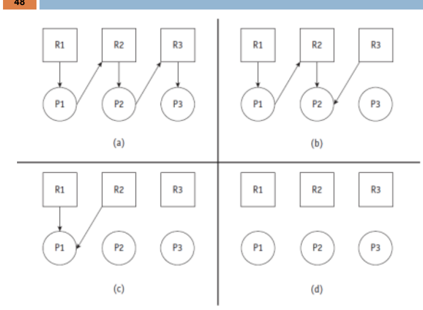
Strategies for Handling Deadlocks (cont'd.)

47

- **Detection:** build directed resource graphs
  - Look for cycles
- Algorithm detecting circularity
  - Executed whenever appropriate
- Detection algorithm
  - Remove process using current resource and not waiting for one
  - Remove process waiting for one resource class
    - Not fully allocated
  - Go back to step 1
    - Repeat steps 1 and 2 until all connecting lines removed

Strategies for Handling Deadlocks (cont'd.)

48



(figure 5.12)

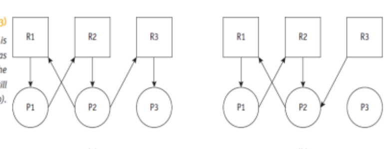
This system is deadlock-free because the graph can be completely reduced, as shown in (d).

Strategies for Handling Deadlocks (cont'd.)

49

(figure 5.13)

Even after this graph (a) is reduced as much as possible (by removing the request from P3), it is still deadlocked (b).



Strategies for Handling Deadlocks (cont'd.)

50

- **Recovery**
  - Deadlock untangled once detected
  - System returns to normal quickly
- All recovery methods have at least one victim
- Recovery methods
  - Terminate every job active in system
    - Restart jobs from beginning
  - Terminate only jobs involved in deadlock
    - Ask users to resubmit jobs
  - Identify jobs involved in deadlock
    - Terminate jobs one at a time

Strategies for Handling Deadlocks (cont'd.)

51

- Recovery methods (cont'd.)
  - Interrupt jobs with record (snapshot) of progress
  - Select nondeadlocked job
    - Preempt its resources
    - Allocate resources to deadlocked process
  - Stop new jobs from entering system
    - Allow nondeadlocked jobs to complete
    - Releases resources when complete
    - No victim

Strategies for Handling Deadlocks (cont'd.)

52

- Factors to consider while selecting a victim
  - Select victim with least-negative effect on the system
  - Most common
    - Job priority under consideration: high-priority jobs usually untouched
    - CPU time used by job: jobs close to completion usually left alone
    - Number of other jobs affected if job selected as victim
    - Jobs modifying data: usually not selected for termination (a database issue)

Starvation

53

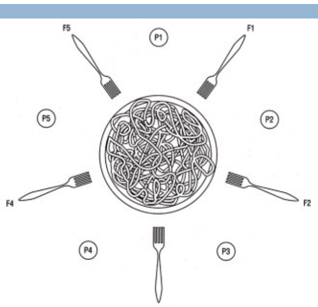
- Job execution prevented
  - Waiting for resources that never become available
  - Results from conservative resource allocation
- Example
  - "The dining philosophers" by Dijkstra
- Starvation avoidance
  - Implement algorithm tracking how long each job waiting for resources (aging)
  - Block new jobs until starving jobs satisfied

Starvation (cont'd.)

54

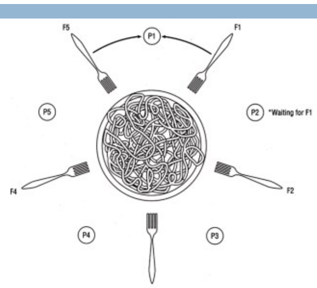
(figure 5.14)

The dining philosophers' table, before the meal begins.



Starvation (cont'd.)

55



(figure 5.15)

Each philosopher must have both forks to begin eating, the one on the right and the one on the left. Unless the resources, the forks, are allocated fairly, some philosophers may starve.

Summary

- Operating system
  - Dynamically allocates resources
  - Avoids deadlock and starvation
- Four methods for dealing with deadlocks
  - Prevention, avoidance, detection, recovery
- Prevention
  - Remove simultaneous occurrence of one or more conditions
  - System will become deadlock-free
  - Prevention algorithms
    - Complex algorithms and high execution overhead

Summary (cont'd.)

- Avoid deadlocks
  - Clearly identify safe and unsafe states
  - Keep reserve resources to guarantee job completion
  - Disadvantage
    - System not fully utilized
- No prevention support
  - System must detect and recover from deadlocks
    - Detection relies on selection of victim