Understanding Operating Systems Seventh Edition

Chapter 5
Process Management

Learning Objectives

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

- The differences among deadlock, race, and starvation
- Several causes of system deadlock and livelock
- The difference between preventing and avoiding deadlocks
- How to detect and recover from deadlocks
- How to detect and recover from starvation
- The concept of a race and how to prevent it

Introduction

- Resource sharing perspectives
  - Memory management and processor sharing
- Many programs competing for limited resources
- Lack of process synchronization consequences
  - Deadlock: “deadly embrace,” “Catch 22,” “blue screen of death,” etc.
    - Two or more jobs placed in HOLD state
    - Jobs waiting for unavailable vital resource
    - System comes to standstill
    - Unresolved by OS: requires external intervention

Deadlock, Livelock, and Starvation

- Narrow staircase analogy
  - Staircase = system; steps and landings = resources
  - Stairs: only wide enough for one person
  - Landing at each floor: room for two people
- Deadlock: two people meet on the stairs
  - Neither retreats
- Livelock: two people on a landing
  - Each time one takes a step to the side, the other mirrors that step; neither moves forward
- Starvation: people wait on landing for a break
  - Break never comes

Deadlock

- More serious than starvation
- Affects entire system
  - Not just a few programs
  - All system resources become unavailable
- More prevalent in interactive systems
- Real-time systems
  - Deadlocks quickly become critical situations
- OS must prevent or resolve

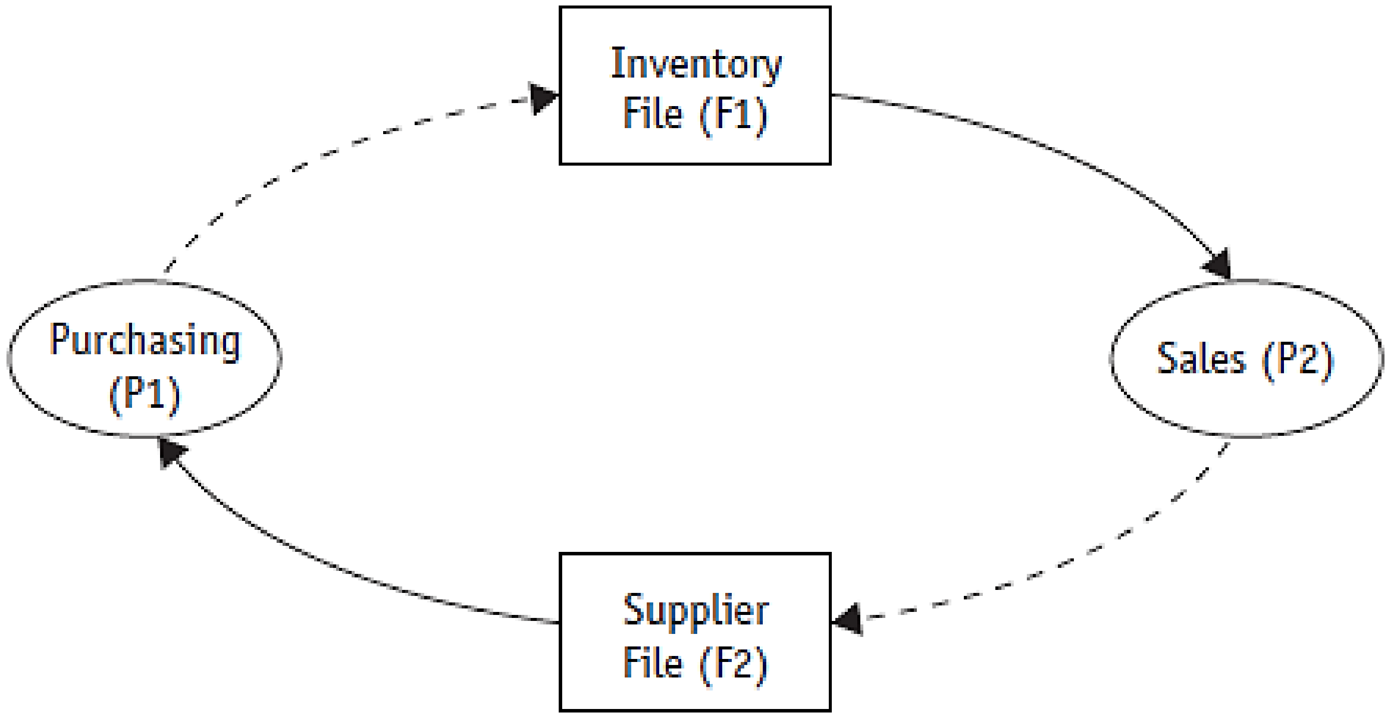
Seven Cases of Deadlock or Livelock

- 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

- 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 is closed *or*
    - One program is forcibly removed and file is released
  - Other programs requiring F1 or F2
    - Put on hold for duration of situation

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



(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.
© Cengage Learning 2014

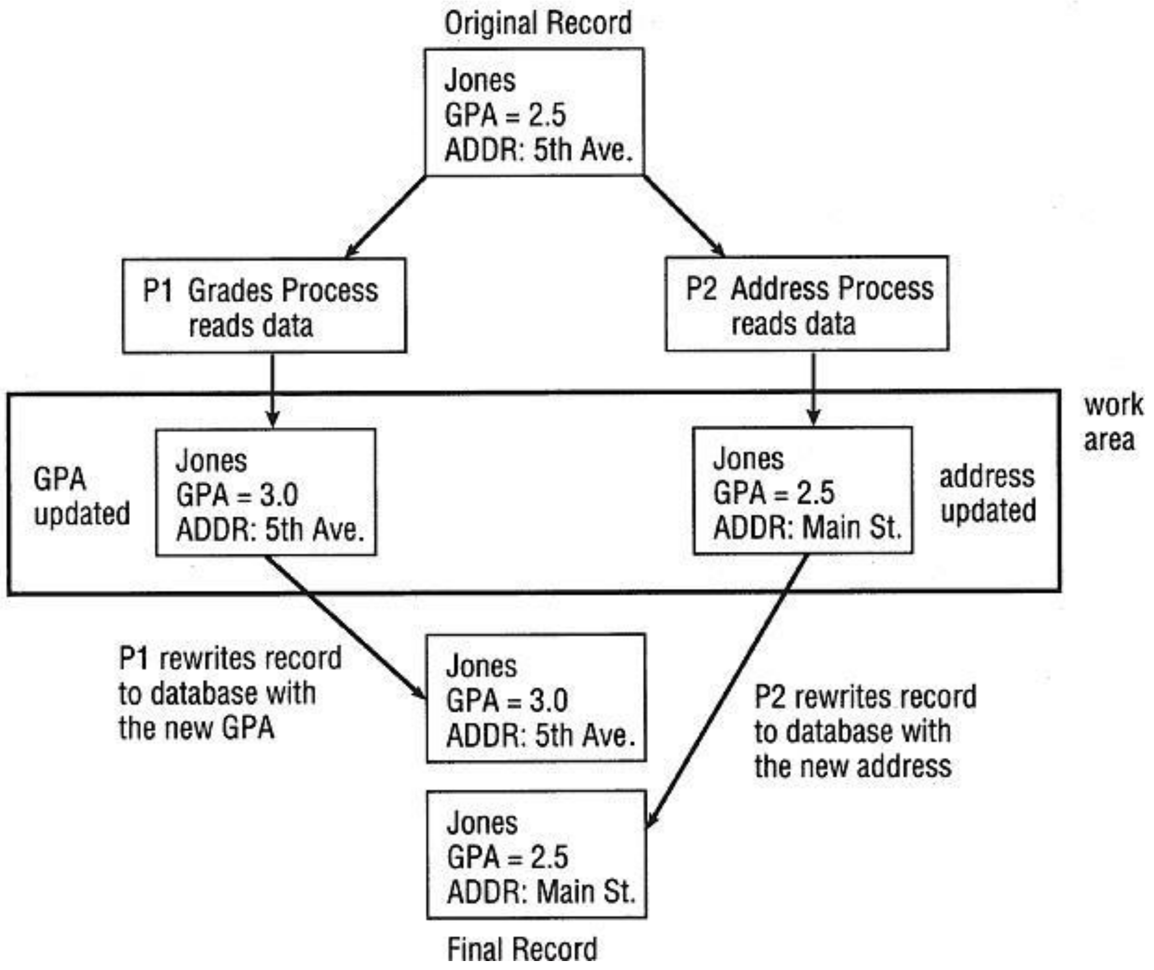
Case 2: Deadlocks in Databases

- Two processes access and lock database records
- Locking
  - Guarantees data integrity
    - One user locks out all other database users
  - Three locking levels
    - Entire database for duration of request
    - Subsection of database
    - Individual record until request completed

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

- 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.)



(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.
© Cengage Learning 2014

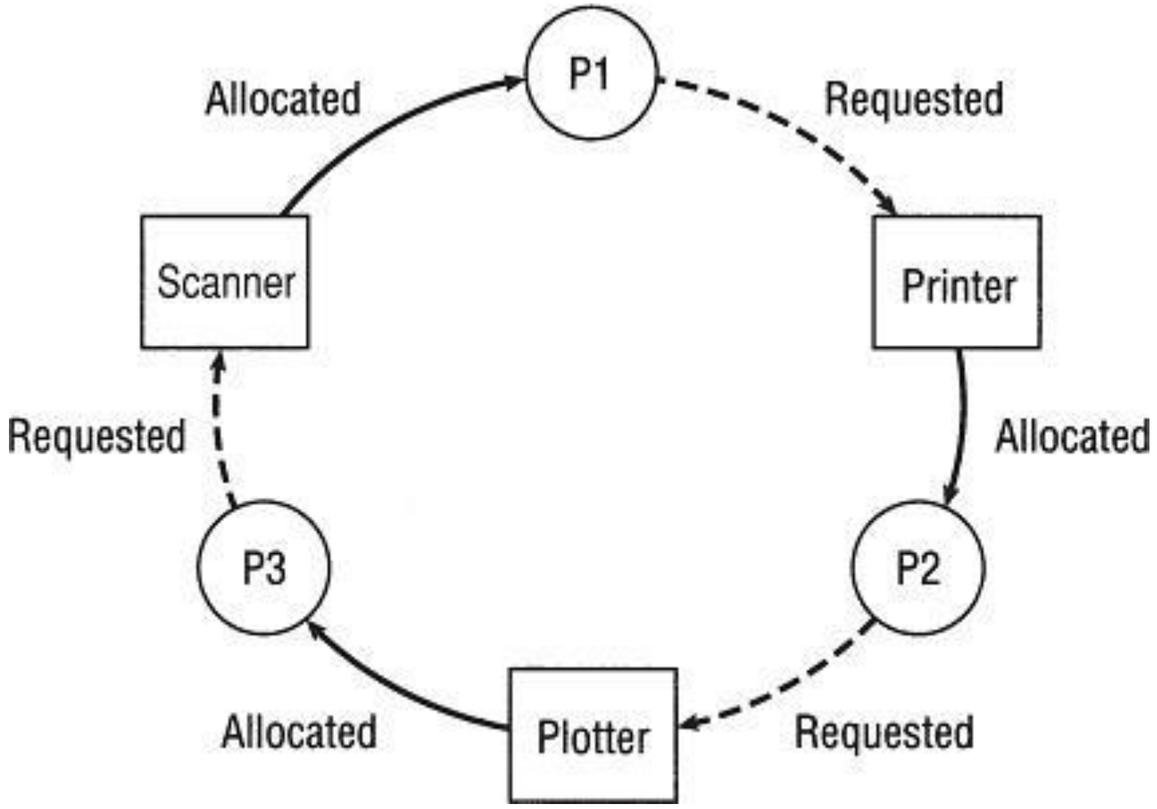
Case 3: Deadlocks in Dedicated Device Allocation

- Limited number of dedicated devices
- Example
  - Two administrators each running education programs with processes P1 and P2, respectively
    - Need two audio recorders each
    - Only two audio recorders (R1 and R2) available
  - Deadlock sequence
    - P1 requests tape R1 and gets it
    - P2 requests tape R2 and gets it
    - P1 requests tape R2 but blocked
    - P2 requests tape R1 but blocked

Case 4: Deadlocks in Multiple Device Allocation

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

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



(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.
© Cengage Learning 2014

Case 5: Deadlocks in Spooling

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

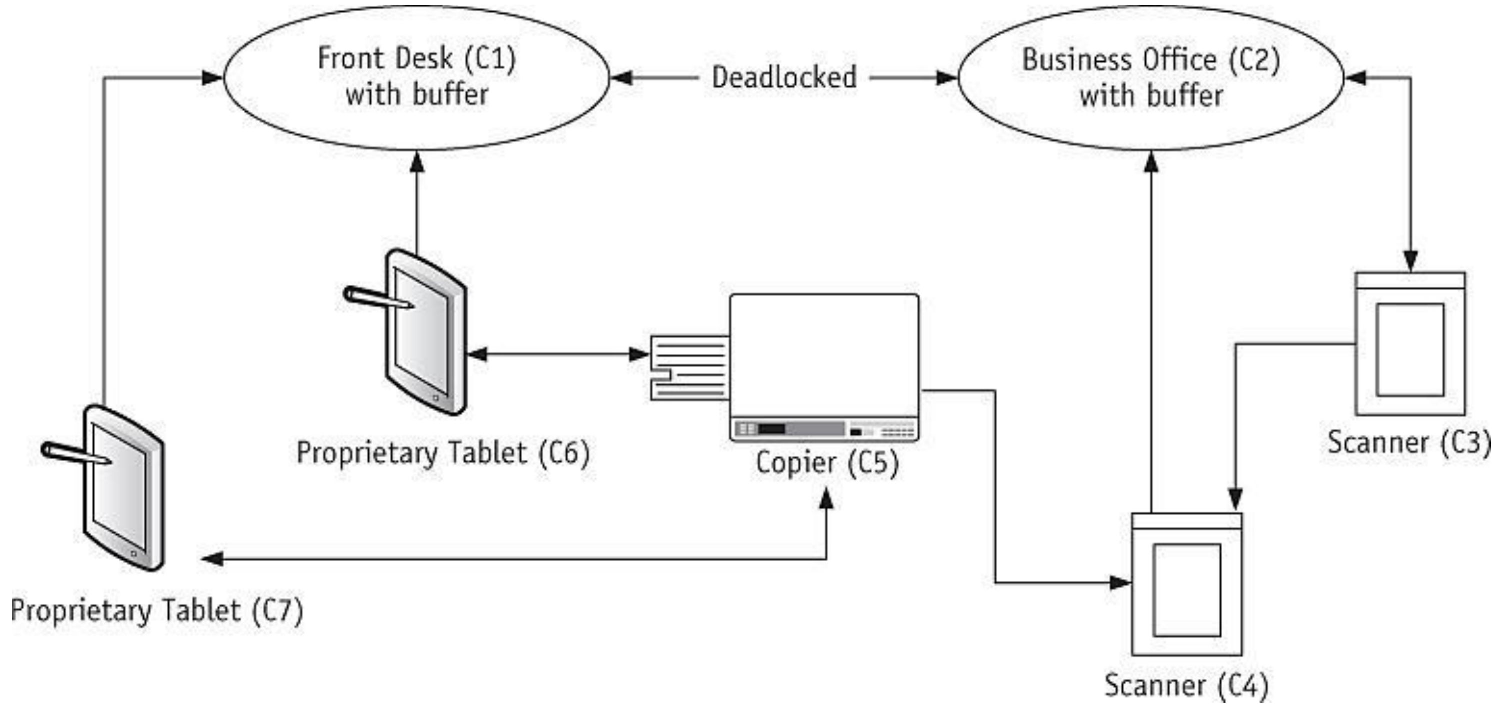
Case 5: Deadlocks in Spooling (cont'd.)

- 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

- If a network have no network protocols controlling network message flow, deadlock can occure
- Example (Figure 5.5)
  - Seven computing devices on network
    - Each on different nodes
  - Direction of arrows
    - Indicates communication flow
  - Deadlock sequence
    - All available buffer space fills

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

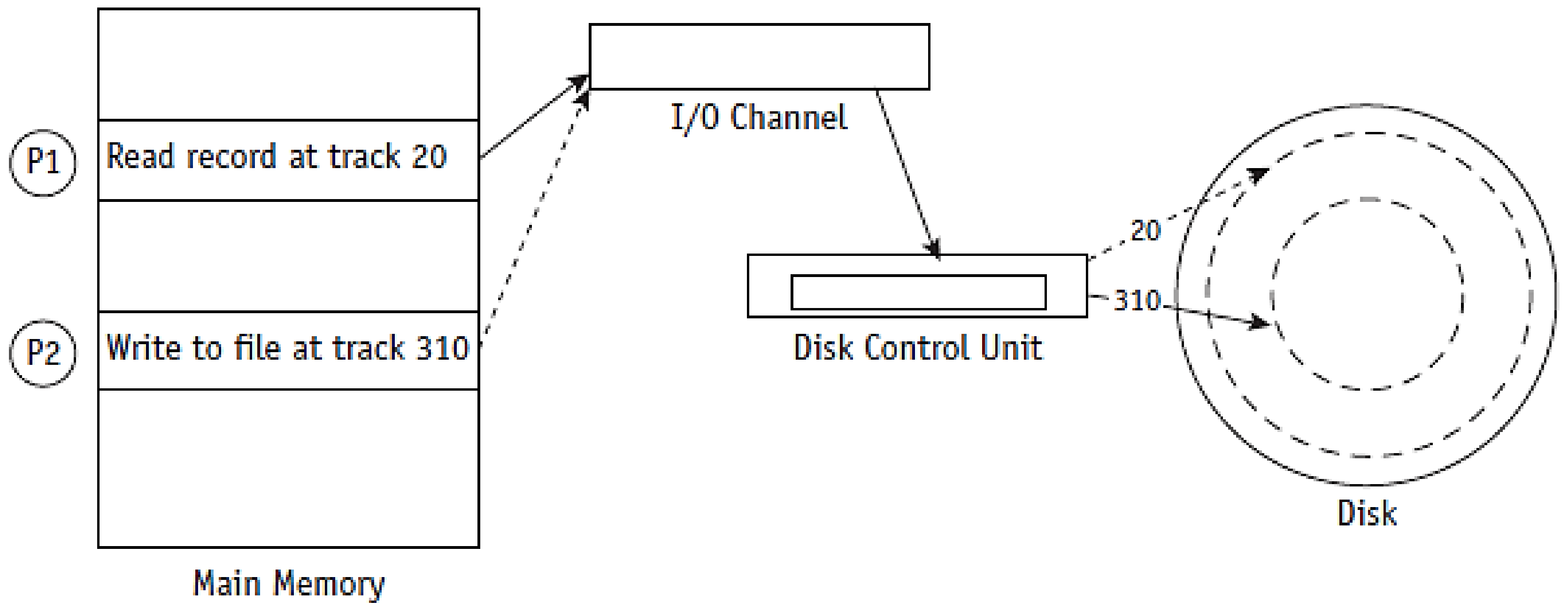


(figure 5.5)
Case 6, deadlocked network flow. Notice that only two nodes, C1 and C2, have buffers. Each line represents a communication path. The arrows indicate the direction of data flow.
© Cengage Learning 2014

Case 7: Deadlocks in Disk Sharing

- Competing processes send conflicting commands
  - Scenario: disk access resulting in livelock
- Example (Figure 5.6)
  - Two processes
  - Each process waiting for I/O request
    - One at Track 20 and one at Track 310
  - Deadlock sequence
    - Arm moves back and forth between Tracks 20 and 310 attempting to fulfill the two competing commands
    - Busy Waiting – different from natural waiting
    - Neither I/O request is satisfied

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



(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.
© Cengage Learning 2014

Necessary Conditions for Deadlock or Livelock

- Four conditions required for a locked system
  1. Mutual exclusion: allowing only one process access to dedicated resource
  2. Resource holding: not releasing the resource; waiting for other job to retreat
  3. No preemption: lack of temporary reallocation of resources
  4. Circular wait: each process waiting for another to voluntarily release so at least one can continue
- All conditions required for deadlock

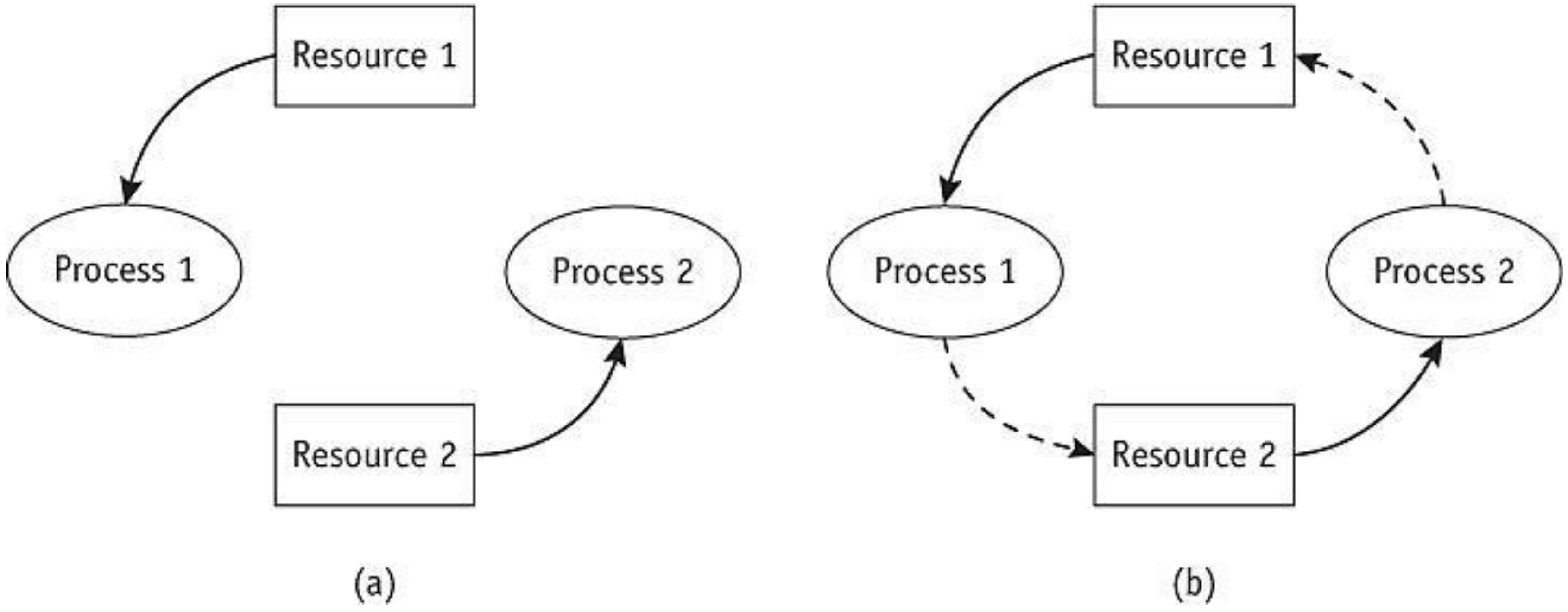
Necessary Conditions for Deadlock or Livelock (cont'd.)

- Resolving deadlock: remove one of the conditions
- All four conditions prevented simultaneously ► deadlock prevented
  - Difficult to implement

Modeling Deadlocks

- Directed graphs: Richard Holt (1972)
  - Circles represent processes
  - Squares represent resources
  - Solid line with arrow from resource to process
    - Process holding resource
  - Dashed line with arrow from a process to resource
    - Process waiting for resource
  - Arrow direction indicates flow
  - Cycle in graph
    - Deadlock involving processes and resources

Modeling Deadlocks (cont'd.)



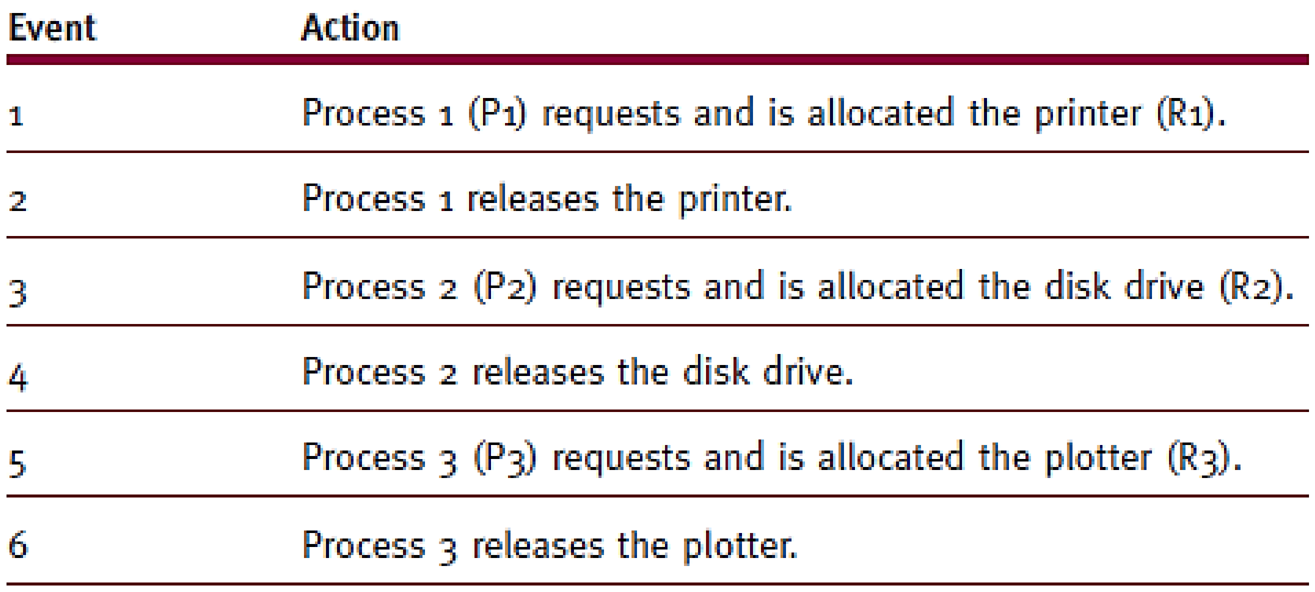
(figure 5.7)
In (a), Resource 1 is being held by Process 1 and Resource 2 is held by Process 2 in a system that is not deadlocked. In (b), Process 1 requests Resource 2 but doesn't release Resource 1, and Process 2 does the same—creating a deadlock. (If one process released its resource, the deadlock would be resolved.)
© Cengage Learning 2014

Modeling Deadlocks (cont'd.)

- 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.)

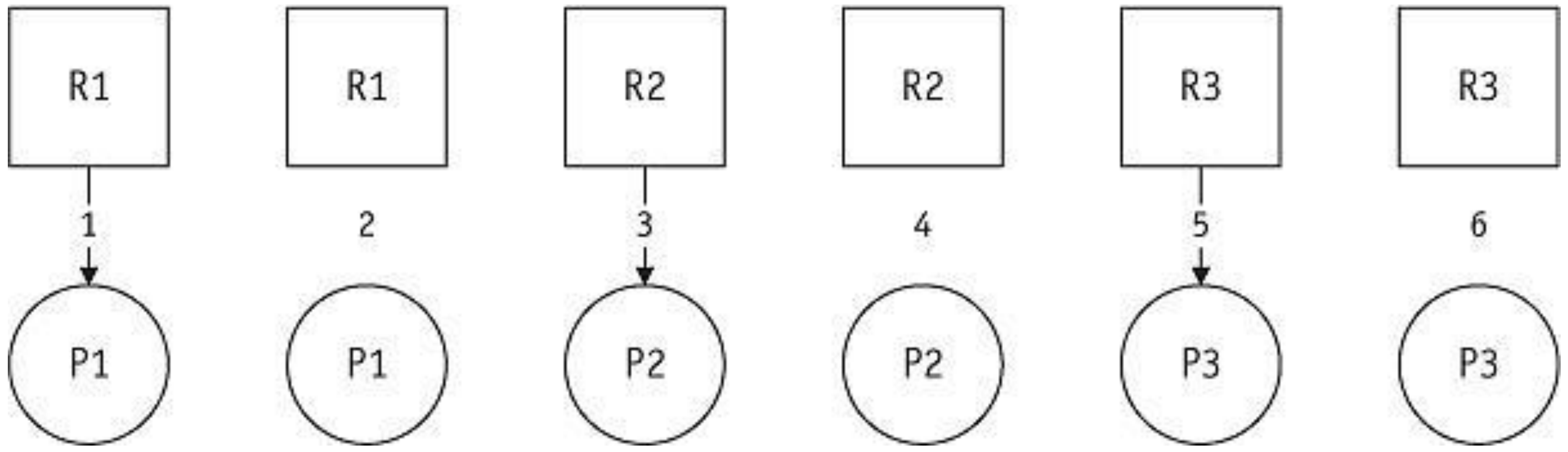
- No deadlock
  - Resources released before next process request



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

(table 5.1)
Scenario 1. These events are shown in the directed graph in Figure 5.8.
© Cengage Learning 2014

Modeling Deadlocks (cont'd.)

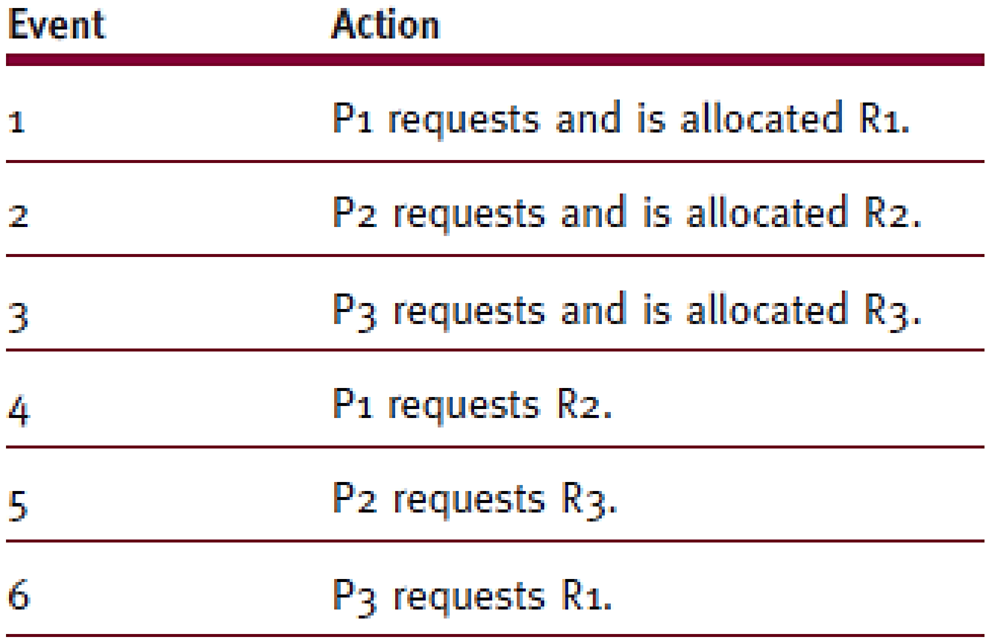


(figure 5.8)
First scenario. The system will stay free of deadlocks if each resource is released before it is requested by the next process.
© Cengage Learning 2014

Modeling Deadlocks (cont'd.)

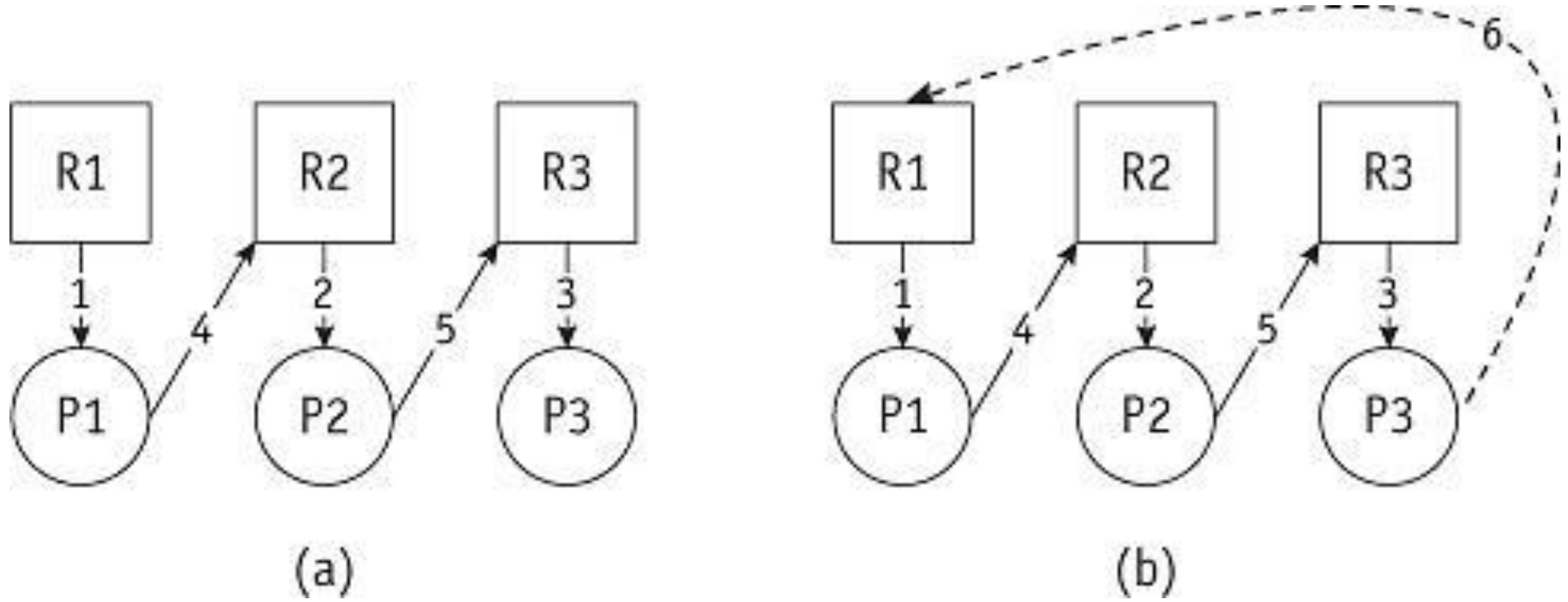
- Deadlock
  - Processes waiting for resource held by another

(table 5.2)
Scenario 2. This sequence of events is shown in the two directed graphs shown in Figure 5.9.
© Cengage Learning 2014



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.

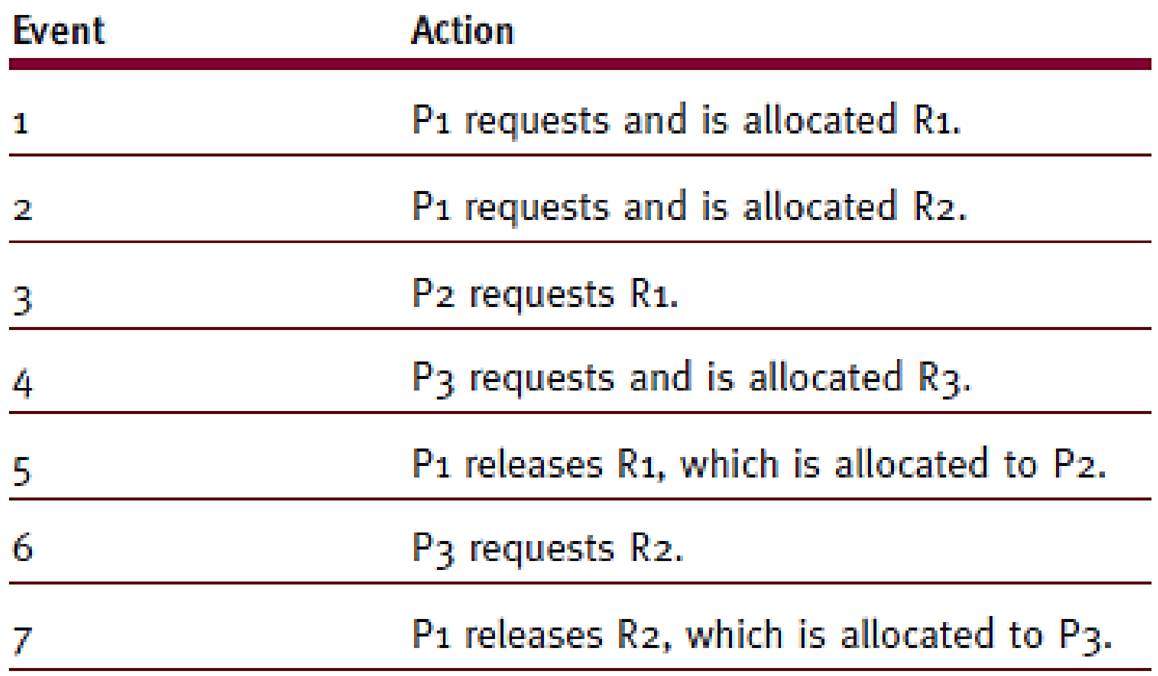
Modeling Deadlocks (cont'd.)



(figure 5.9)
Second scenario. The system (a) becomes deadlocked (b) when P3 requests R1. Notice the circular wait.
© Cengage Learning 2014

Modeling Deadlocks (cont'd.)

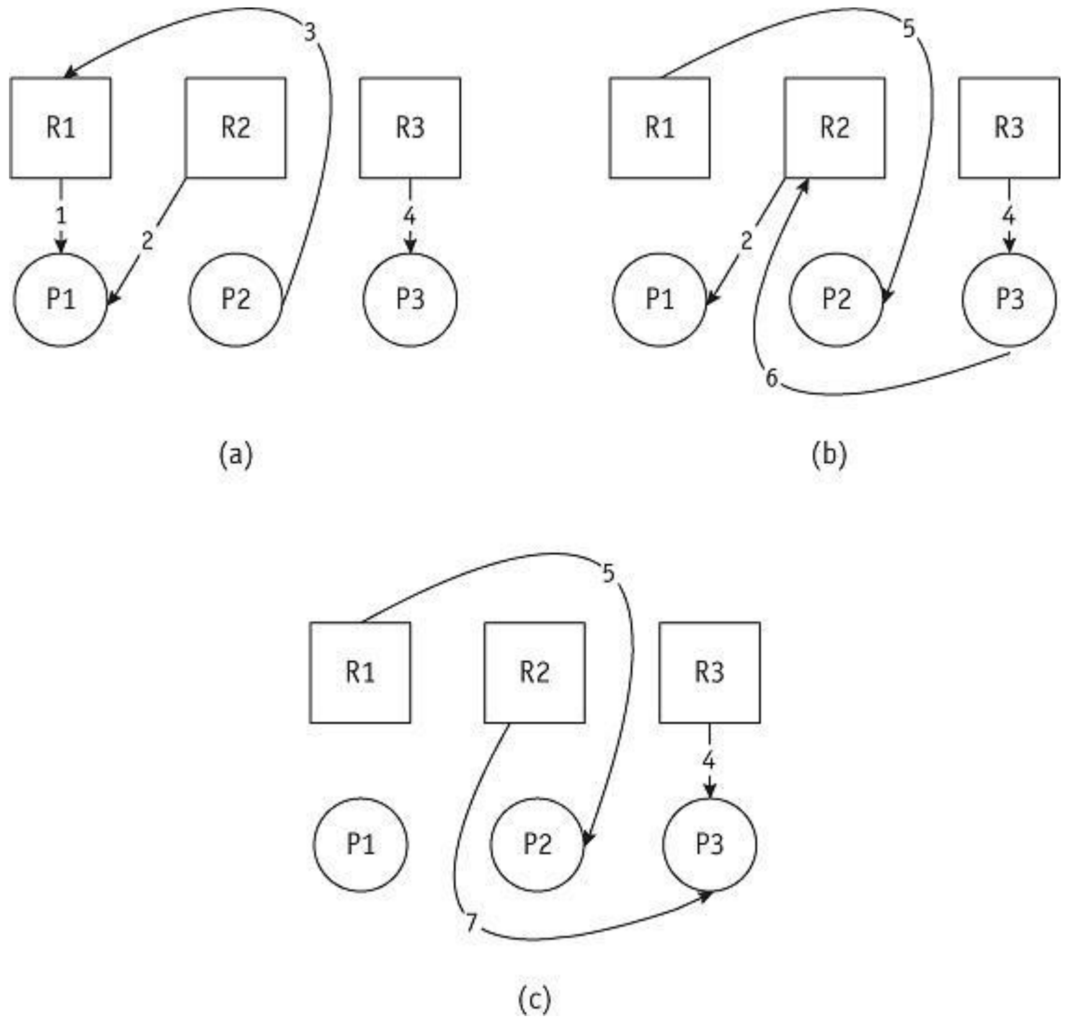
- No deadlock
  - Resources released before deadlock



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)
Scenario 3. This sequence of events is shown in the directed graph in Figure 5.10.
© Cengage Learning 2014

Modeling Deadlocks (cont'd.)



(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).
© Cengage Learning 2014

Strategies for Handling Deadlocks

- 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
- Recovery
  - Resume system normalcy quickly and gracefully

Strategies for Handling Deadlocks (cont'd.)

- Prevention eliminates one of four conditions
  - Complication: same condition cannot be eliminated from every resource
  - 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

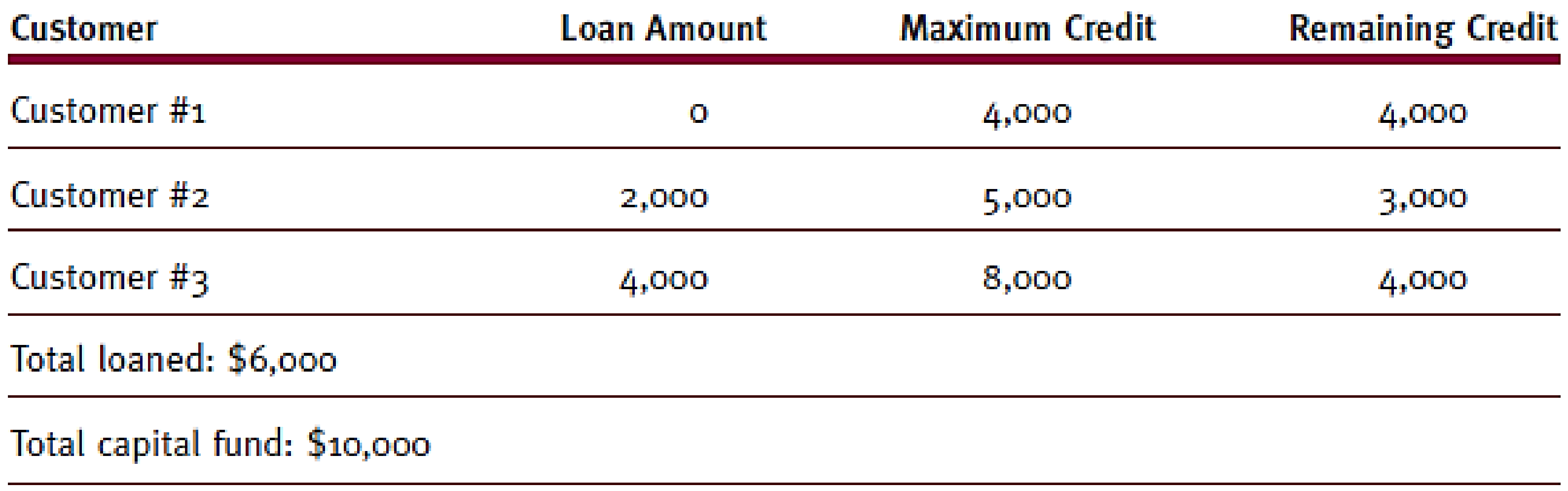
Strategies for Handling Deadlocks (cont'd.)

- Prevention (cont'd.)
  - No preemption
    - Bypassed if operating system allowed to deallocate resources from jobs
    - Okay if job state easily saved and restored
    - Preempting dedicated I/O device or files during modification: extremely unpleasant recovery tasks
  - Circular wait
    - Bypassed if operating system prevents circle formation
    - Uses hierarchical ordering scheme
    - Requires jobs to anticipate resource request order
    - Difficult to satisfy all users

Strategies for Handling Deadlocks (cont'd.)

- Avoidance: use if non of the conditions can be removed
  - System should know ahead of time
    - Sequence of requests associated with each active process
- Dijkstra’s Bankers Algorithm (Dijkstra, 1965)
  - Regulates resource allocation to avoid deadlocks
    - 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

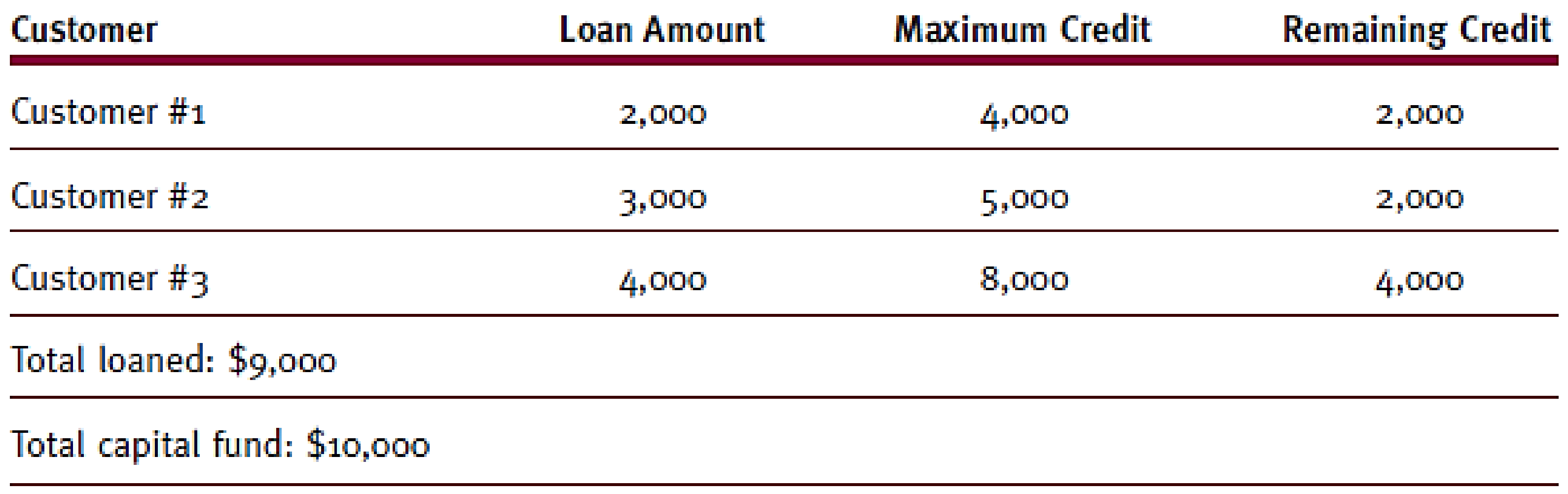
Strategies for Handling Deadlocks (cont'd.)



Customer	Loan Amount	Maximum Credit	Remaining Credit
Customer #1	0	4,000	4,000
Customer #2	2,000	5,000	3,000
Customer #3	4,000	8,000	4,000
Total loaned: $6,000			
Total capital fund: $10,000			

(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.”
© Cengage Learning 2014

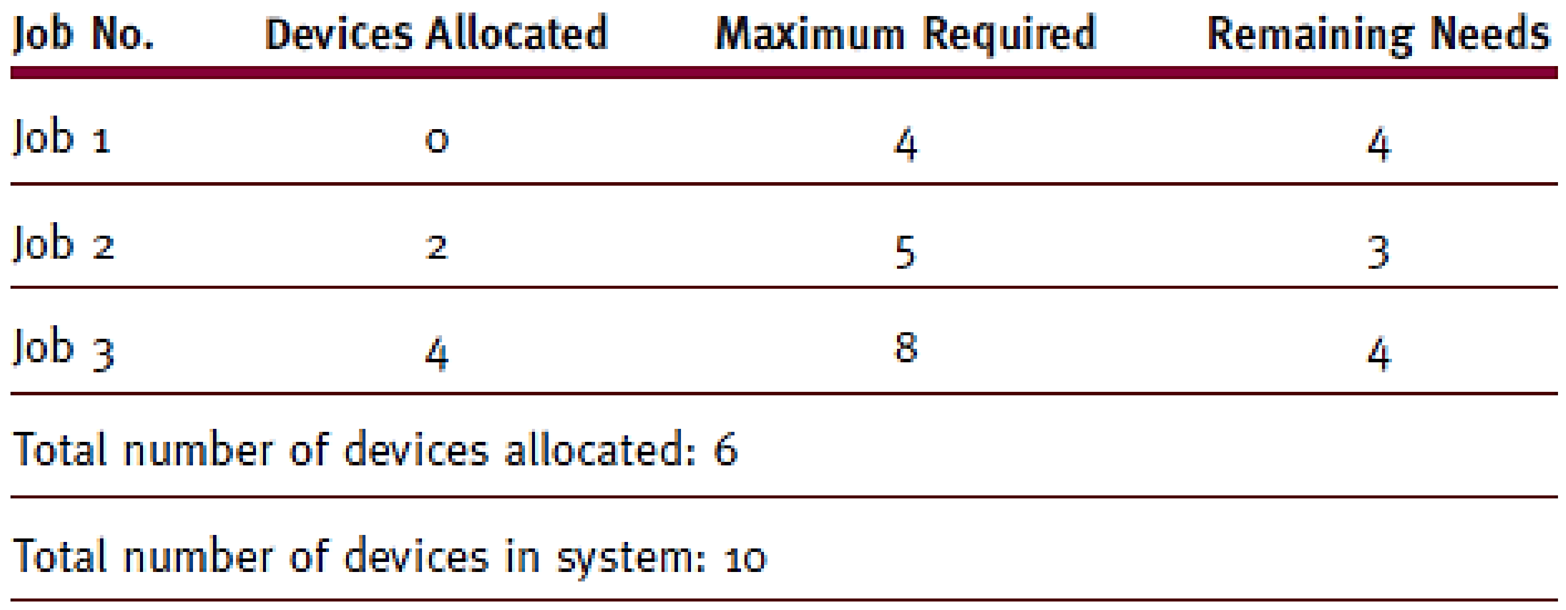
Strategies for Handling Deadlocks (cont'd.)



Customer	Loan Amount	Maximum Credit	Remaining Credit
Customer #1	2,000	4,000	2,000
Customer #2	3,000	5,000	2,000
Customer #3	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."
© Cengage Learning 2014

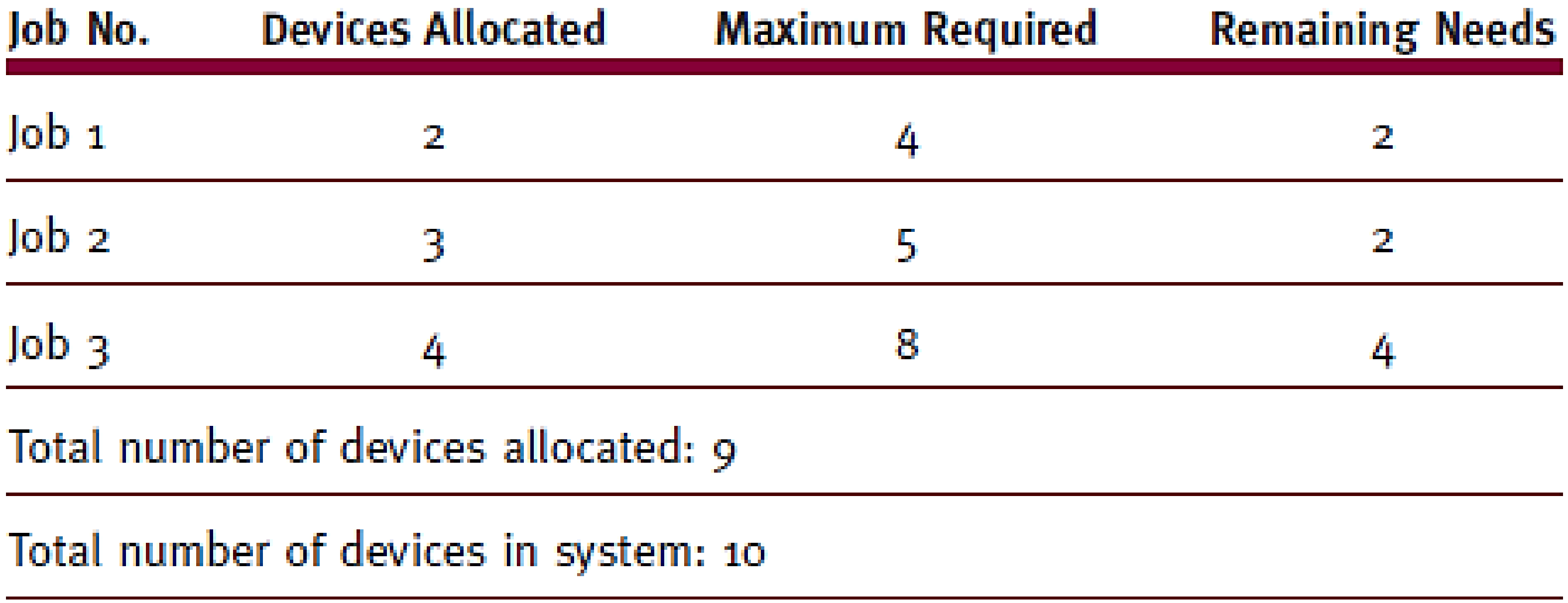
Strategies for Handling Deadlocks (cont'd.)



Job No.	Devices Allocated	Maximum Required	Remaining Needs
Job 1	0	4	4
Job 2	2	5	3
Job 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. This is a safe state: Six devices are allocated and four units are still available.

Strategies for Handling Deadlocks (cont'd.)



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

(table 5.7)
Resource assignments after later allocations. This is an unsafe state: Only one unit is available but every job requires at least two to complete its execution.
© Cengage Learning 2014

Strategies for Handling Deadlocks (cont'd.)

- Operating systems deadlock avoidance assurances
  - Never satisfy request if its moves OS 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

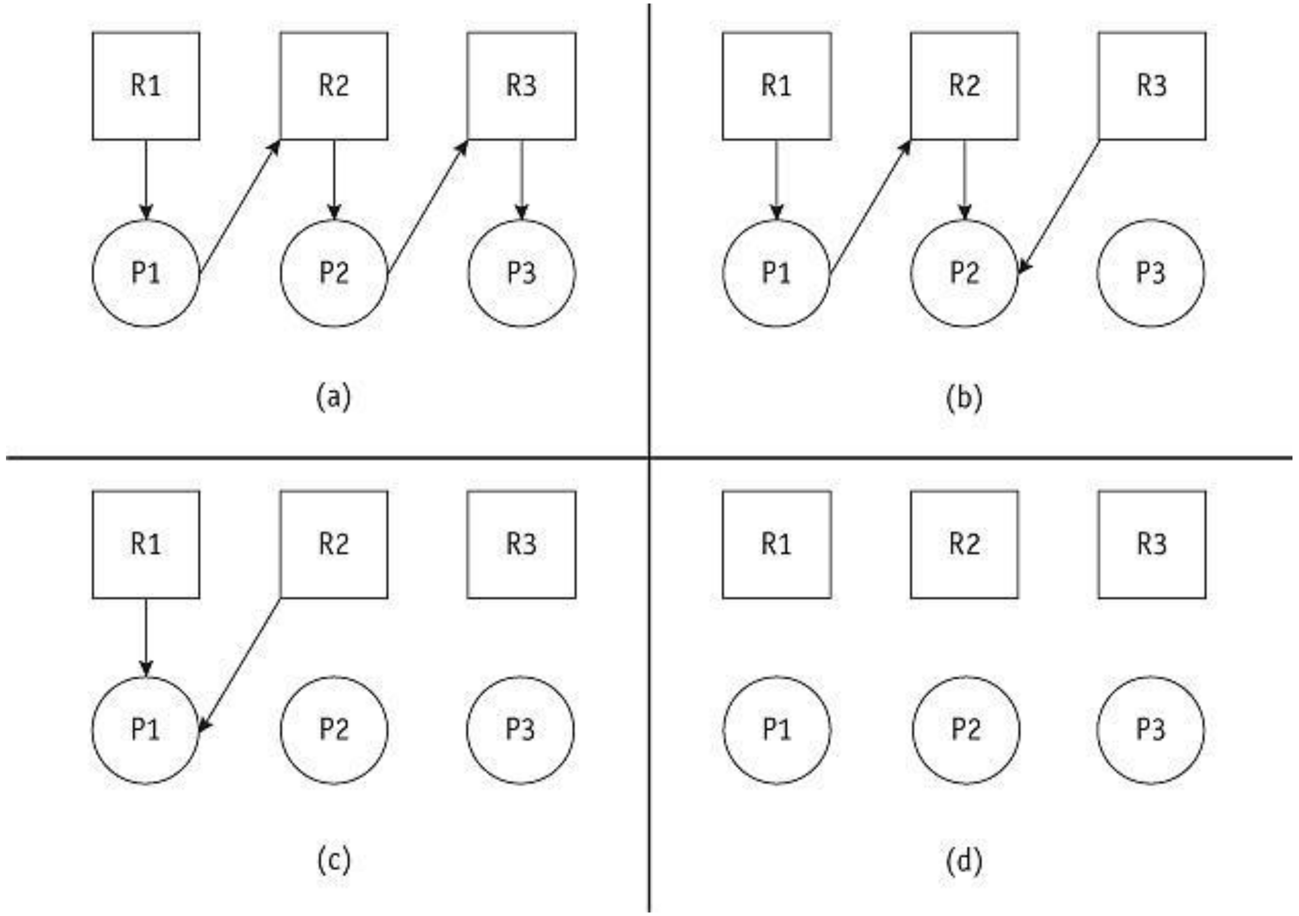
Strategies for Handling Deadlocks (cont'd.)

- 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.)

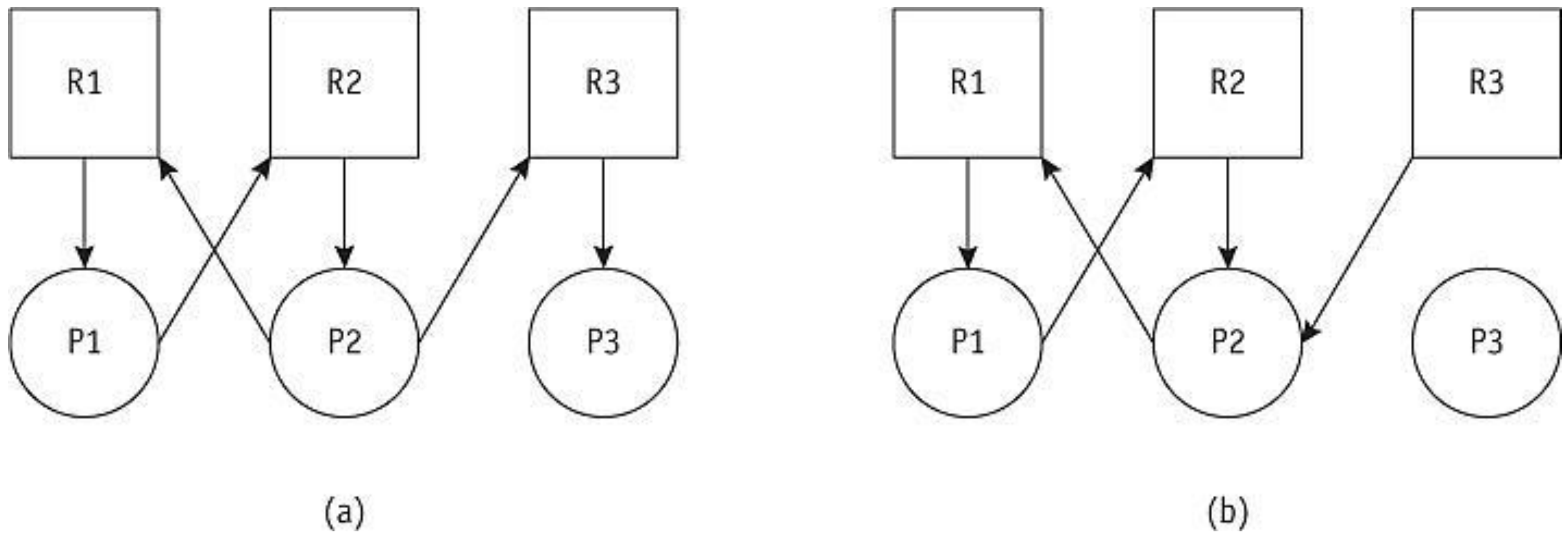
- Detection: build directed resource graphs
  - Look for cycles
- Algorithm detecting circularity
  - Executed whenever appropriate
- Detection algorithm
  1. Remove process using current resource and not waiting for one
  2. Remove process waiting for one resource class
     - Not fully allocated
 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

Strategies for Handling Deadlocks (cont'd.)



(figure 5.11)
This system is deadlock-free because the graph can be completely reduced, as shown in (d).
© Cengage Learning 2014

Strategies for Handling Deadlocks (cont'd.)



(figure 5.12)
Even after this graph (a) is reduced as much as possible (by removing the request from P3), it is still deadlocked (b).
© Cengage Learning 2014

Strategies for Handling Deadlocks (cont'd.)

- Recovery
  - Deadlock untangled once detected
  - System returns to normal quickly
- Nearly 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.)

- 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

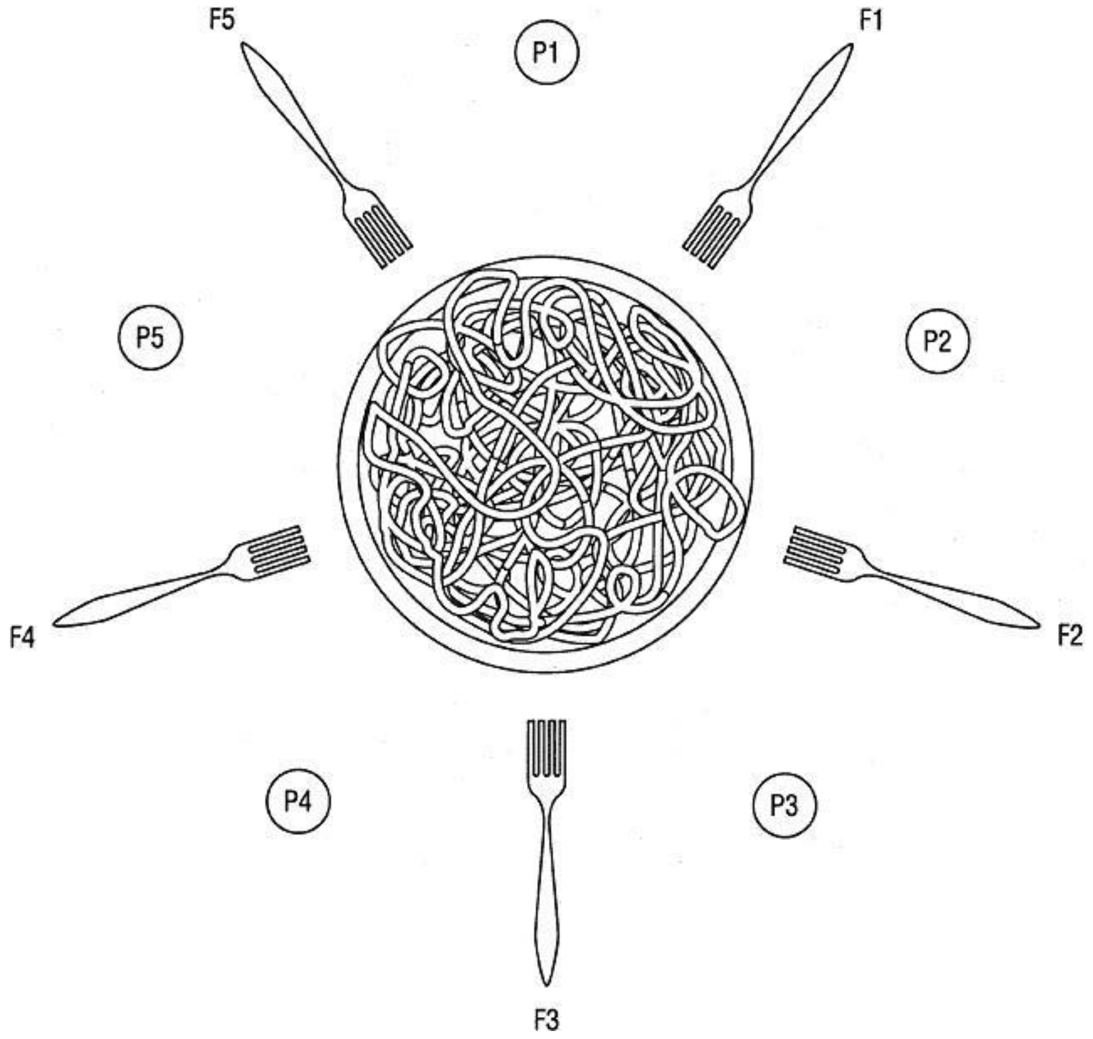
Strategies for Handling Deadlocks (cont'd.)

- Factors to consider
  - 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

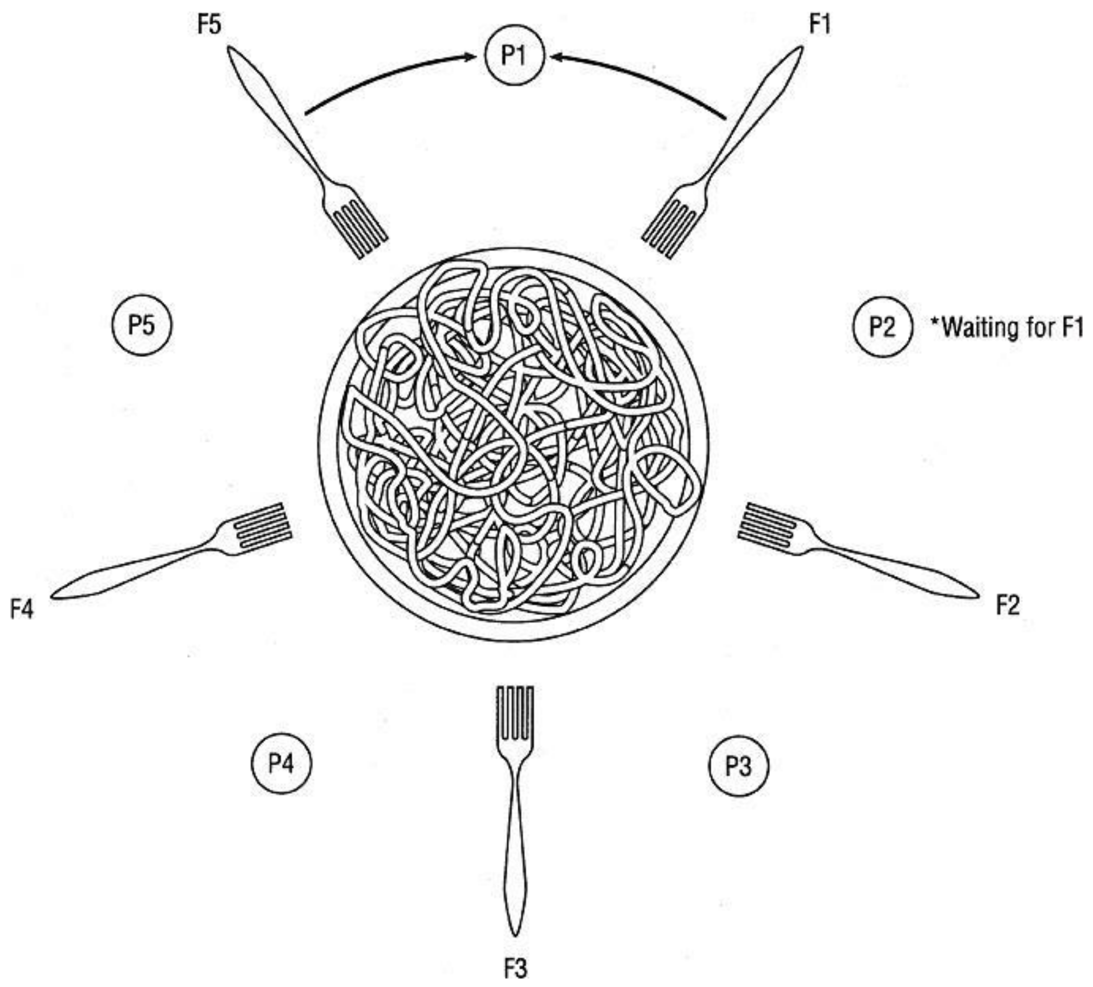
- Job execution prevented
  - Waiting for resources that never become available
  - Results from conservative resource allocation
- Example
  - Dining Philosophers Problem (Dijkstra, 1968)
- Starvation avoidance
  - Implement algorithm tracking how long each job has been waiting for resources (aging)
  - Starvation detected: block new jobs until starving jobs satisfied

Starvation (cont'd.)



(figure 5.13)
The dining philosophers' table, before the meal begins.
© Cengage Learning 2014

Starvation (cont'd.)



(figure 5.14)
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.
© Cengage Learning 2014

Conclusion

- Operating systems
  - Must dynamically allocate resources while avoiding deadlock and starvation
- Four methods for dealing with deadlocks
  - Prevention, avoidance, detection, and recovery
- Prevention
  - Remove simultaneous occurrence of one or more conditions: system will become deadlock-free
  - Prevention algorithms
    - Complex algorithms and high execution overhead

Conclusion (cont'd.)

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