Distributed Process Management

Chapter 14

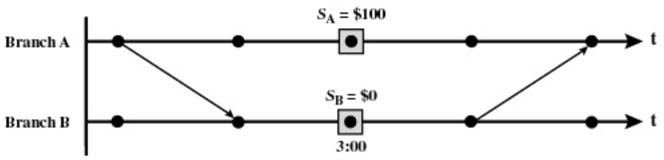
Distributed Global States

- Operating system cannot know the current state of all process in the distributed system
- A process can only know the current state of all processes on the local system
- Remote processes only know state information that is received by messages
    - These messages represent the state in the past

Example

- Bank account is distributed over two branches
- The total amount in the account is the sum at each branch
- At 3 PM the account balance is determined
- Messages are sent to request the information

Example



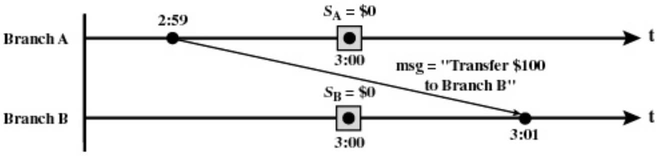
(a) Total = $100

Figure 14.3 Example of Determining Global States

Example

- If at the time of balance determination, the balance from branch A is in transit to branch B
- The result is a false reading

Example



(b) Total = $0

Figure 14.3 Example of Determining Global States

Example

- All messages in transit must be examined at time of observation
- Total consists of balance at both branches and amount in message

---

Example

- If clocks at the two branches are not perfectly synchronized
- Transfer amount at 3:01 from branch A
- Amount arrives at branch B at 2:59
- At 3:00 the amount is counted twice

Example

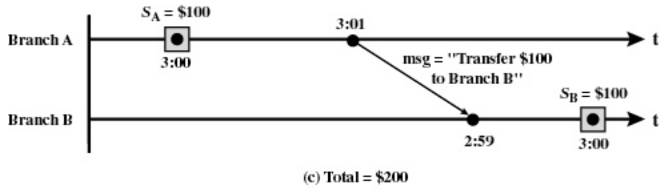


Figure 14.3 Example of Determining Global States

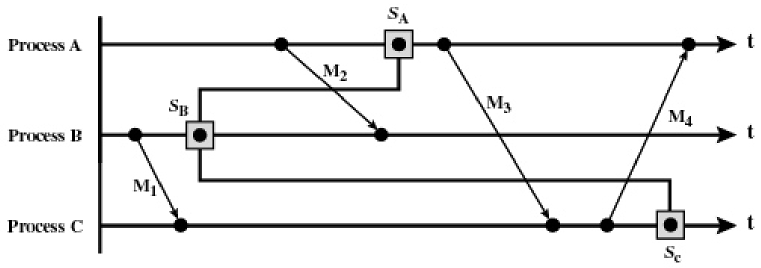
Some Terms

- Channel
  - Exists between two processes if they exchange messages
- State
  - Sequence of messages that have been sent and received along channels incident with the process

Some Terms

- Snapshot
  - Records the state of a process
- Global state
  - The combined state of all processes
- Distributed Snapshot
  - A collection of snapshots, one for each process

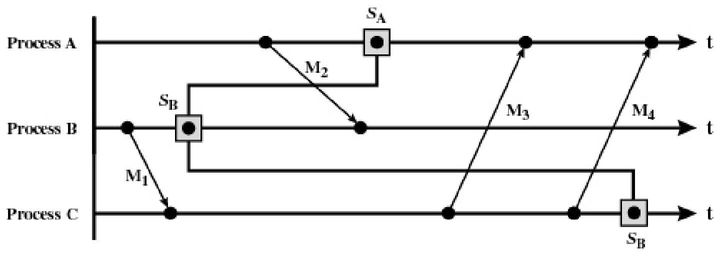
Global State



(a) Inconsistent Global State

Figure 14.4 Inconsistent and Consistent Global States

Global State



(b) Consistent Global State

Figure 14.4 Inconsistent and Consistent Global States

Distributed Snapshot Algorithm

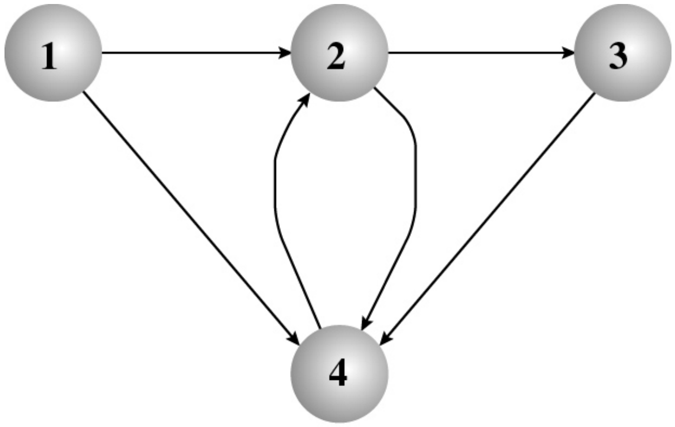
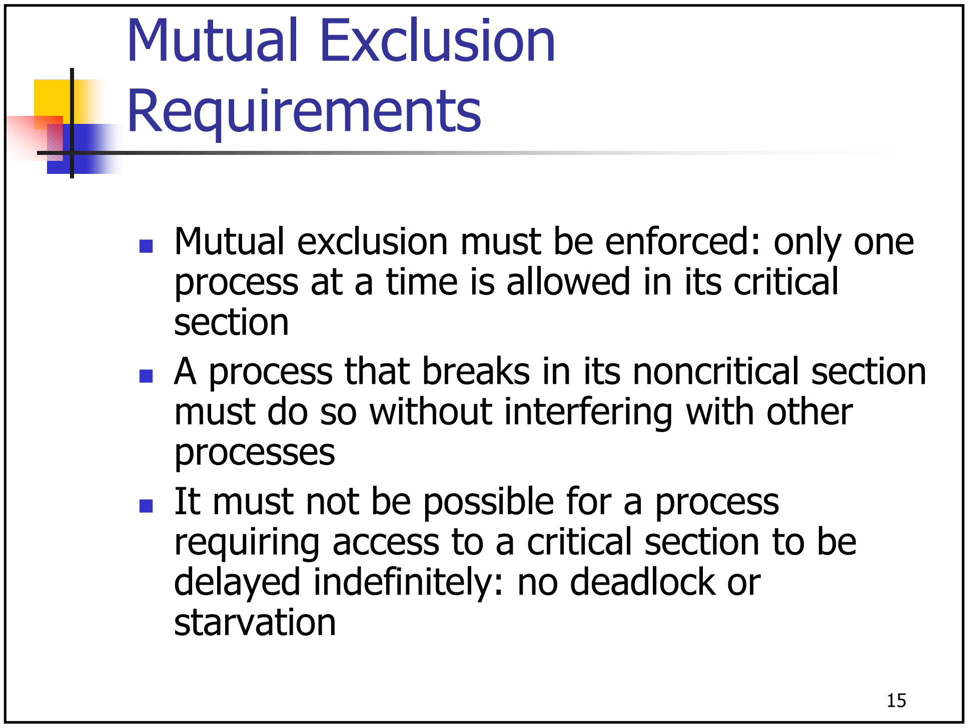
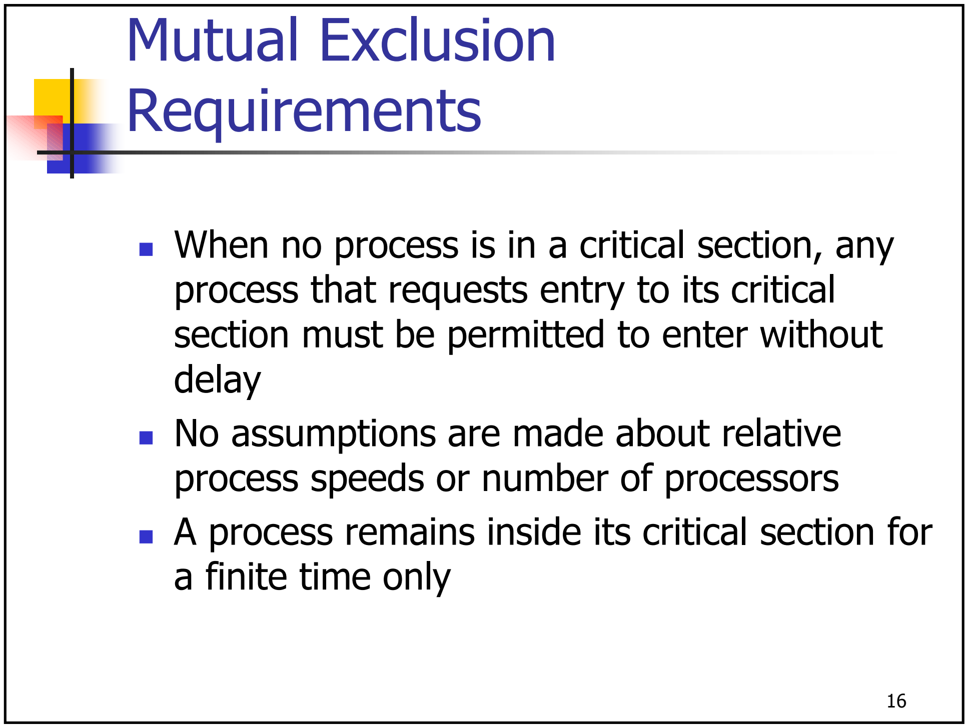


Figure 14.5 Process and Channel Graph



Mutual Exclusion Requirements

- Mutual exclusion must be enforced: only one process at a time is allowed in its critical section
- A process that breaks in its noncritical section must do so without interfering with other processes
- It must not be possible for a process requiring access to a critical section to be delayed indefinitely: no deadlock or starvation



Mutual Exclusion Requirements

- When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay
- No assumptions are made about relative process speeds or number of processors
- A process remains inside its critical section for a finite time only

Centralized Algorithm for Mutual Exclusion

- One node is designated as the control node
- This node control access to all shared objects
- If control node fails, mutual exclusion breaks down

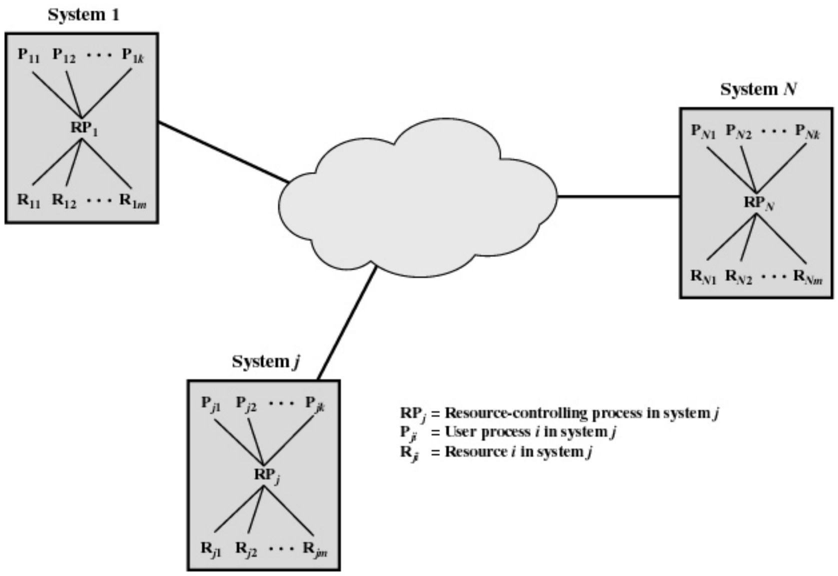


Figure 14.7 Model for Mutual Exclusion Problem in Distributed Process Management

Distributed Algorithm

- All nodes have equal amount of information, on average
- Each node has only a partial picture of the total system and must make decisions based on this information
- All nodes bear equal responsibility for the final decision

Distributed Algorithm

- All nodes expend equal effort, on average, in effecting a final decision
- Failure of a node, in general, does not result in a total system collapse
- There exists no system-wide common clock with which to regulate the time of events

Ordering of Events

- Events must be ordered to ensure mutual exclusion and avoid deadlock
- Clocks are not synchronized
- Communication delays
- State information for a process is not up to date

Ordering of Events

- Need to consistently say that one event occurs before another event
- Messages are sent when wanting to enter critical section and when leaving critical section
- Time-stamping
  - Orders events on a distributed system
  - System clock is not used

Time-Stamping

- Each system on the network maintains a counter which functions as a clock
- Each site has a numerical identifier
- When a message is received, the receiving system sets is counter to one more than the maximum of its current value and the incoming time-stamp (counter)

Time-Stamping

- If two messages have the same time-stamp, they are ordered by the number of their sites
- For this method to work, each message is sent from one process to all other processes
  - Ensures all sites have same ordering of messages
  - For mutual exclusion and deadlock all processes must be aware of the situation

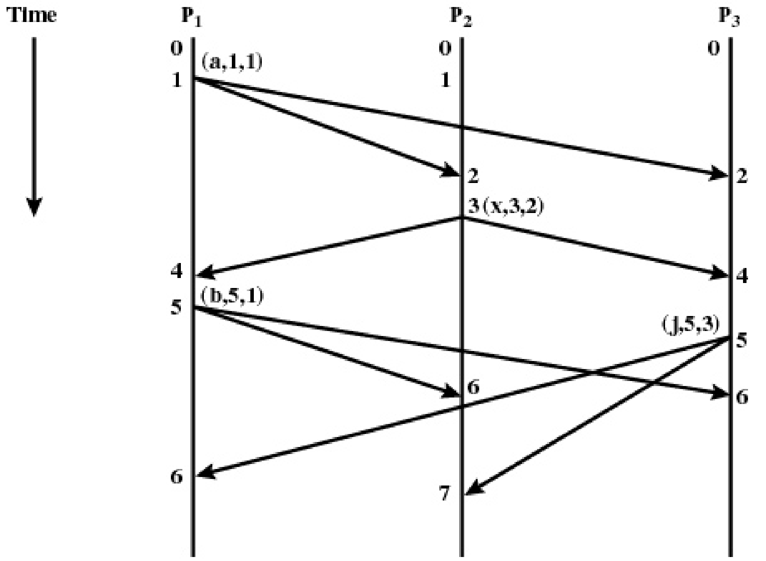


Figure 14.8 Example of Operation of Timestamping Algorithm

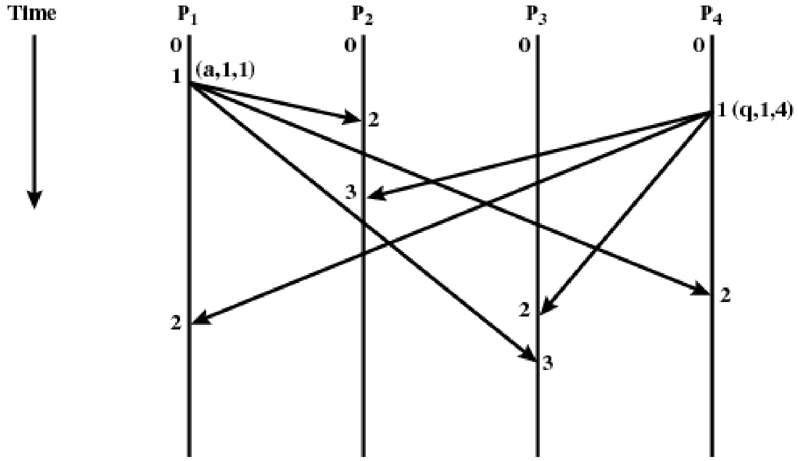


Figure 14.9 Another Example of Operation of Timestamping Algorithm

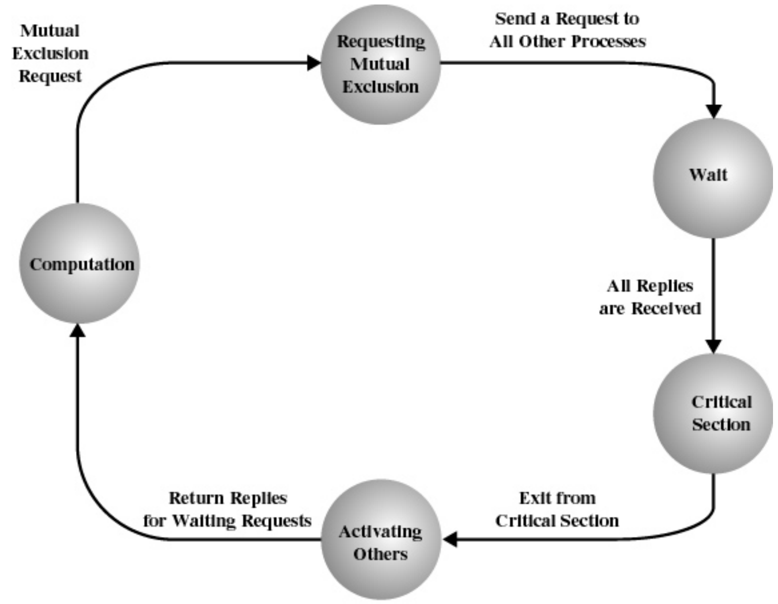


Figure 14.10 State Diagram for Algorithm in [RICA81]

Token-Passing Approach

- Pass a token among the participating processes
- The token is an entity that at any time is held by one process
- The process holding the token may enter its critical section without asking permission
- When a process leaves its critical section, it passes the token to another process

Deadlock in Resource Allocation

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

Deadlock Prevention

- Circular-wait condition can be prevented by defining a linear ordering of resource types
- Hold-and-wait condition can be prevented by requiring that a process request all of its required resource at one time, and blocking the process until all requests can be granted simultaneously

Deadlock Avoidance

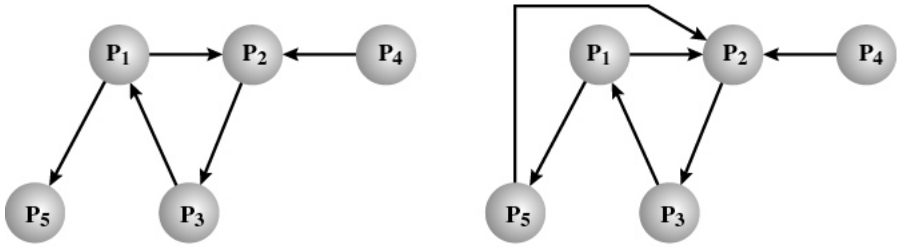
- Distributed deadlock avoidance is impractical
  - Every node must keep track of the global state of the system
  - The process of checking for a safe global state must be mutually exclusive
  - Checking for safe states involves considerable processing overhead for a distributed system with a large number of processes and resources

Distributed Deadlock Detection

- Each site only knows about its own resources
  - Deadlock may involve distributed resources
- Centralized control – one site is responsible for deadlock detection
- Hierarchical control – lowest node above the nodes involved in deadlock
- Distributed control – all processes cooperate in the deadlock detection function

Deadlock in Message Communication

- Mutual Waiting
  - Deadlock occurs in message communication when each of a group of processes is waiting for a message from another member of the group and there are no messages in transit



(a) No deadlock

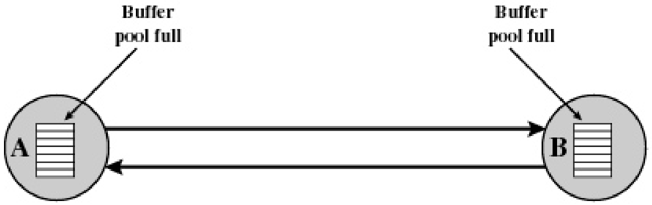
(b) Deadlock

Figure 14.16 Deadlock in Message Communication

Deadlock in Message Communication

- Unavailability of Message Buffers
  - Well known in packet-switching data networks
  - Example: buffer space for A is filled with packets destined for B. The reverse is true at B.

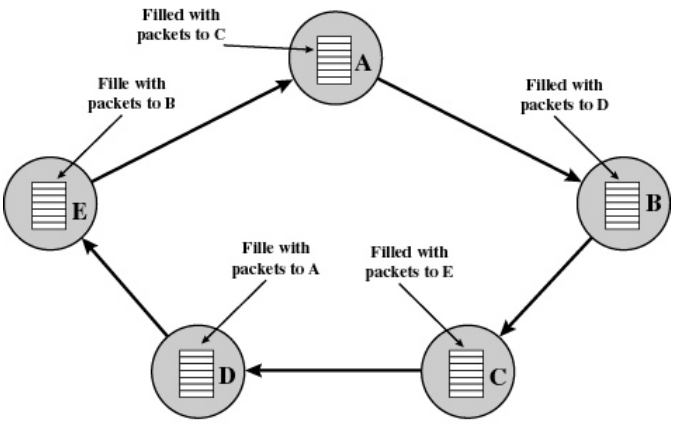
Direct Store-and-Forward Deadlock



(a) Direct store-and-forward deadlock

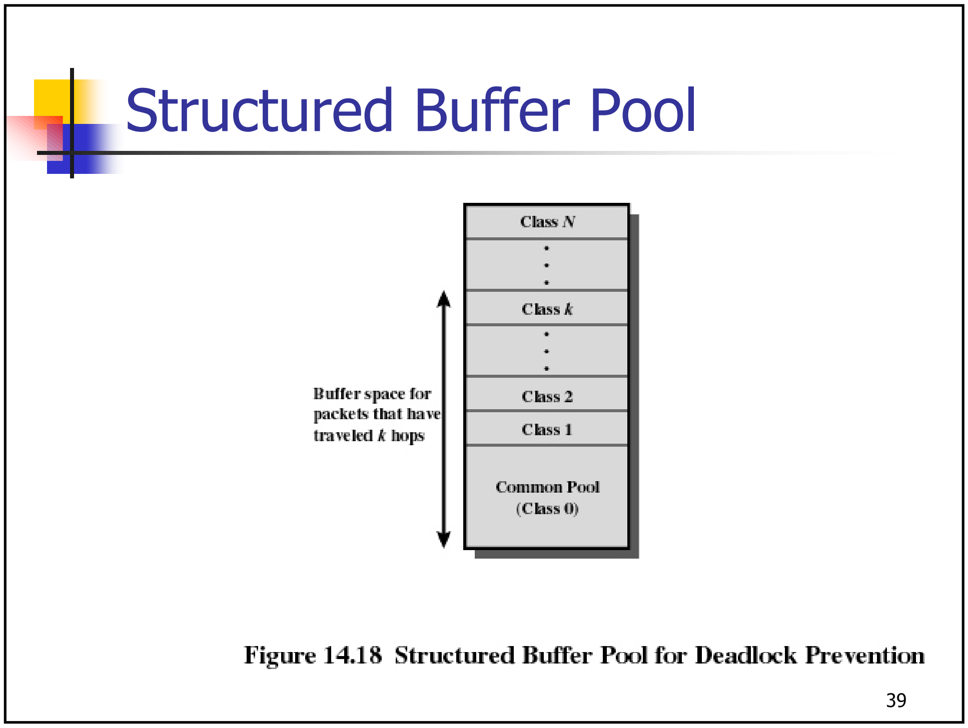
Deadlock in Message Communication

- Unavailability of Message Buffers
  - For each node, the queue to the adjacent node in one direction is full with packets destined for the next node beyond



(b) Indirect store-and-forward deadlock

Figure 14.17 Store-and-Forward Deadlock



Structured Buffer Pool

Figure 14.18 Structured Buffer Pool for Deadlock Prevention