Lecture - XII

Deadlocks & Main Memory Management

Tevfik Koşar

University at Buffalo
October 11th, 2011

Roadmap

- Deadlocks
  - Resource Allocation Graphs
  - Deadlock Detection
  - Deadlock Prevention
  - Deadlock Avoidance
  - Deadlock Recovery
- Main Memory Management

Deadlock Avoidance

Deadlock Prevention: prevent deadlocks by restraining resources and making sure one of 4 necessary conditions for a deadlock does not hold. (system design)

--> possible side effect: low device utilization and reduced system throughput

Deadlock Avoidance: Requires that the system has some additional a priori information available. (dynamic request check)
i.e. request disk and then printer..
or request at most n resources

--> allows more concurrency

- Similar to the difference between a traffic light and a police officer directing the traffic!

Example

P1:
Request Disk
Request Printer
....
Release Printer
Release Disk

P2:
Request Printer
Request Disk
....
Release Disk
Release Printer

Safe State

- A state is safe if the system can allocate resources to each process (upto its maximum) in some order and can still avoid a deadlock.
- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a safe sequence of all processes.
**Safe State**

- Sequence \(<P_1, P_2, ..., P_n>\) is safe if for each \(P_i\), the resources that \(P_i\) can still request can be satisfied by currently available resources + resources held by all the \(P_j\), with \(j<i\).
  - If \(P_i\) resource needs are not immediately available, then \(P_i\) can wait until all \(P_j\) have finished.
  - When \(P_i\) is finished, \(P_i\) can obtain needed resources, execute, return allocated resources, and terminate.
  - When \(P_i\) terminates, \(P_{i+1}\) can obtain its needed resources, and so on.
- If no such sequence exists, the state is **unsafe!**

**Basic Facts**

- If a system is in safe state \(\Rightarrow\) no deadlocks.
- If a system is in unsafe state \(\Rightarrow\) possibility of deadlock.
- Avoidance \(\Rightarrow\) ensure that a system will never enter an unsafe state.

**Safe, Unsafe, Deadlock State**

```
  deadlock
    
      unsafe

    safe
```

**Example**

Consider a system with 3 processes and 12 disks.

\[ \text{At } t = t_0; \]

<table>
<thead>
<tr>
<th>Maximum Needs</th>
<th>Current Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>9</td>
</tr>
</tbody>
</table>

**Example (cont.)**

Consider a system with 3 processes and 12 disks.

\[ \text{At } t = t_1; \]

<table>
<thead>
<tr>
<th>Maximum Needs</th>
<th>Current Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>9</td>
</tr>
</tbody>
</table>

**Resource-Allocation Graph Algorithm**

- **Claim edge** \(P_i \rightarrow R_j\) indicated that process \(P_i\) may request resource \(R_j\); represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed **a priori** in the system.
Banker's Algorithm

- Works for multiple resource instances.
- Each process declares maximum # of resources it may need.
- When a process requests a resource, it may have to wait if this leads to an unsafe state.
- When a process gets all its resources it must return them in a finite amount of time.

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   \[
   \text{Work} = \text{Available} \\
   \text{Finish} = \text{false} \text{ for } i = 1, 2, \ldots, n.
   \]
2. Find an i such that both:
   (a) Finish[i] = false
   (b) Need ≥ Work
   If no such i exists, go to step 4.
3. Work = Work + Allocation[i]
   Finish[i] = true
   go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

Resource-Request Algorithm for Process \(P_i\)

Let \(Request\) be the request vector for process \(P_i\).
If \(Request[j] = k\) then process \(P_i\) wants \(k\) instances of resource type \(R_j\).
1. If \(Request_j ≥ Need\), go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If \(Request_j ≤ Available\), go to step 3. Otherwise \(P_i\) must wait, since resources are not available.
3. Pretend to allocate requested resources to \(P_i\) by modifying the state as follows:
   \[
   \text{Available} = \text{Available} - \text{Request}; \\
   \text{Allocation} = \text{Allocation} + \text{Request}; \\
   \text{Need} = \text{Need} - \text{Request}.
   \]
   ● If safe ⇒ the resources are allocated to \(P_i\).
   ● If unsafe ⇒ \(P_i\) must wait, and the old resource-allocation state is restored.
Example of Banker’s Algorithm

- 5 processes $P_0$ through $P_4$; 3 resource types: $A$ (10 instances), $B$ (5 instances), and $C$ (7 instances).

- Snapshot at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- The content of the matrix. Need is defined to be $\text{Max} - \text{Allocation}$.

<table>
<thead>
<tr>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>$B$</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>$C$</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>$P_0$</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>$P_1$</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>$P_3$</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

- The system is in a safe state since the sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies safety criteria.

Example of Banker’s Algorithm

- Snapshot at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
<td>$A$</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Example: $P_1$ Requests $(1,0,2)$

- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true.

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Need</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Executing safety algorithm shows that sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies safety requirement.

- Can request for $(3,3,0)$ by $P_4$ be granted?
- Can request for $(0,2,0)$ by $P_0$ be granted?

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes. \(\rightarrow\) expensive
- Abort one process at a time until the deadlock cycle is eliminated. \(\rightarrow\) overhead of deadlock detection alg.

- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?
Recovery from Deadlock: Resource Preemption

• Selecting a victim - minimize cost.
• Rollback - return to some safe state, restart process for that state.
• Starvation - same process may always be picked as victim, include number of rollback in cost factor.

Main Memory Management

Memory Management Requirements

- The O/S must fit multiple processes in memory
  - memory needs to be subdivided to accommodate multiple processes
  - memory needs to be allocated to ensure a reasonable supply of ready processes so that the CPU is never idle
  - memory management is an optimization task under constraints

Fitting processes into memory is like fitting boxes into a fixed amount of space

Memory Allocation

- Fixed-partition allocation
  - Divide memory into fixed-size partitions
  - Each partition contains exactly one process
  - The degree of multi programming is bound by the number of partitions
  - When a process terminates, the partition becomes available for other processes
    ➔ no longer in use

Memory Allocation (Cont.)

- Variable-partition Scheme (Dynamic)
  - When a process arrives, search for a hole large enough for this process
  - Hole - block of available memory; holes of various size are scattered throughout memory
  - Allocate only as much memory as needed
  - Operating system maintains information about:
    a) allocated partitions
    b) free partitions (hole)

Dynamic Storage-Allocation Problem

How to satisfy a request of size $n$ from a list of free holes

- First-fit: Allocate the first hole that is big enough
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- Worst-fit: Allocate the largest hole; must also search entire list. Produces the largest leftover hole.

First-fit is faster.
Best-fit is better in terms of storage utilization.
Worst-fit may lead less fragmentation.
Example

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Summary

- Deadlocks
  - Resource Allocation Graphs
  - Deadlock Detection
  - Deadlock Prevention
  - Deadlock Avoidance
  - Deadlock Recovery
- Main Memory Management

Acknowledgements

- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR