Deadlock

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Review — Deadlock in mutual exclusion algorithms

- Each process is “waiting” for the other process to do something constructive so ever done — both processes are forever stuck

```
process P1
begin
loop
inCS[1] := TRUE
while inCS[2] do
NOOP
end while
<critical section>
inCS[1] := FALSE
end loop
end

process P2
begin
loop
while inCS[1] do
NOOP
end while
<critical section>
end loop
end
```

global var inCS : array[1..2] of boolean := (FALSE,FALSE)
Deadlock
Definition

A set of processes are deadlocked when every process in the set is waiting for an event that can only be generated by some process in the set.

Example: a computer system with two processes

❖ a process that is printing a large PostScript job is waiting for more memory
❖ a visualization process with lots of memory is waiting to use the printer

A Formal Model of Deadlock
Definitions

◆ Basic components of any resource allocation problem
    » Processes
    » Resources

◆ Paradigm of resource usage

process P
begin
  request(resource-type, amount)
  use the resource
  release(resource-type, amount)
end P
Resources

Two classes of resources

◆ Serially reusable (SR) resources
  » a constant number of units
  » boolean state (allocated or unallocated)
  » no sharing
  » units of the resource created by the system (not by processes using the resource)
◆ Consumable resources (CR)
  » the number of available units of the resource varies over time
  » a producer process may release units of the resource it did not acquire (i.e., a process may create units of the resource)
  » in general, acquired resources are not returned, they are consumed

A Graph Theoretic Model of Deadlock

The resource allocation graph (RAG)

◆ Model the state of a computer system as a directed graph $G = (V, E)$
  » $V$ the set of vertices = $\{P_1, ..., P_n\} \cup \{R_1, ..., R_n\}$
  » $E$ the set of edges =
    - (edges from a resource to a process) $\cup$
    - (edges from a process to a resource)

$P_i \rightarrow R_j \rightarrow P_k$

request edge allocation edge
Resource Allocation Graphs

Examples

- A printing process that is waiting for more memory & a large memory process that is waiting to print: $G = \{\text{print, visualization}\} \cup \{\text{memory, printer}\}$

A Graph Theoretic Model of Deadlock

Resource allocation graphs & deadlock

- **Theorem:** If a graph does not contain a cycle then no processes are deadlocked
  - A cycle in a RAG is a necessary condition for deadlock
- Is the existence of a cycle a sufficient condition?
A Graph Theoretic Model of Deadlock
Resource allocation graphs & deadlock

- **Theorem**: If there is only a single unit of all resources then a set of processes are deadlocked if the processes & resources form a cycle in the RAG

![Graph Diagram]

Using the Theory
An operational definition of deadlock

- A set of processes are deadlocked iff the following conditions hold simultaneously
  1. Mutual exclusion is required
  2. A process is in a “hold-and-wait” state
  3. Preemption is not allowed
  4. Circular waiting exists
     (A cycle exists in the RAG)
Dealing With Deadlock

Deadlock prevention & avoidance

- Adopt some resource allocation protocol that ensures deadlock can never occur
  - Deadlock prevention
  - Deadlock avoidance

- Deadlock prevention
  - Ensure that one of the four deadlock conditions never occurs
    - Mutex?
    - Hold & Wait?
    - Non-preemption?
    - Circular waiting?

Dealing With Deadlock

Deadlock avoidance

Examine each resource request and determine whether or not granting the request can lead to deadlock

- Define a set of vectors & matrices that characterize the current state of all resources & processes
  - Resource allocation state matrix
    \[ n_{ij} = \text{the number of units of resource } j \text{ held by process } i \]
  - Maximum claim matrix
    \[ n_{ij} = \text{the maximum number of units of resource } i \text{ that the process } j \text{ will ever require simultaneously} \]
  - Available vector
    \[ \langle n_1, n_2, \ldots, n_r \rangle \text{ where } n_i = \text{the number of units of resource } i \text{ that are unallocated} \]
Deadlock Avoidance

State definitions

- A resource allocation state is safe if the system can allocate resources to each process up to its maximum claim such that the system can not deadlock
  - There must be an ordering of the processes $P_1, P_2, \ldots, P_n$ such that for all processes $P_i$,
    $$\text{MAX CLAIM}_{P_i} - \text{ALLOCATION}_{P_i} \leq \text{AVAIL} + \sum_{j=1}^{i-1} \text{ALLOCATION}_{P_j}$$
    i.e., the number of resources that $P_i$ can request is less than the resources available now plus those held by lower numbered processes in the sequence
  - This ordering of processes is called a safe sequence
    - If a safe sequence exists then there exists a process ($P_i$) that can execute to completion
    - $P_i$ can execute to completion at worst after processes $P_1, P_2, \ldots, P_{i-1}$ complete

Deadlock Avoidance

Example

- A computer system with 5 processes and 4 resources

<table>
<thead>
<tr>
<th>ALLOCATION</th>
<th>MAX CLAIM</th>
<th>AVAILABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1, R_1, R_2$</td>
<td>$R_1, R_1, R_2$</td>
<td>$R_1, R_1, R_2$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>0 0 1 2</td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1 0 0 0</td>
<td>1 7 5 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1 3 5 3</td>
<td>2 3 5 6</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 6 3 2</td>
<td>0 6 5 2</td>
</tr>
<tr>
<td>$P_5$</td>
<td>0 1 4</td>
<td>0 6 5 6</td>
</tr>
</tbody>
</table>

- Is this system in a safe state?
  - Does there exist a safe sequence?
    - an ordering of the processes such that:
      $$\text{MAX CLAIM}_{P_i} - \text{ALLOCATION}_{P_i} < \text{AVAIL} + \sum_{j=1}^{i-1} \text{ALLOCATION}_{P_j}$$
Deadlock Avoidance Example
Safe sequence computation

1. Compute the largest possible resource request a process can make

<table>
<thead>
<tr>
<th>MAX_CLAIM</th>
<th>ALLOCATION</th>
<th>MAX_REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 0 0 1 2 0 0 1 2 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2 1 7 5 0 1 0 0 0 0 7 5 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3 2 3 5 6 1 3 5 3 1 0 3 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4 0 6 5 2 0 6 3 2 0 0 2 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 0 6 5 6 0 0 1 4 0 6 4 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Attempt to build a safe sequence

- Does there exist a process $P_i$ such that $\text{MAX_CLAIM}_{P_i} \leq \text{AVAILABLE}$?
  - If no, then there is no safe sequence, the state is unsafe
  - If yes, add $P_i$ to the sequence
  - Set $\text{AVAILABLE} = \text{AVAILABLE} + \text{ALLOCATION}_{P_i}$
Deadlock Avoidance Example
Safe sequence computation

<table>
<thead>
<tr>
<th>ALLOCATION</th>
<th>MAX_CLAIM</th>
<th>AVAILABLE</th>
<th>MAX_REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0 0 1 2</td>
<td>0 0 1 2</td>
<td>2 1 0 0</td>
</tr>
<tr>
<td>P₂</td>
<td>0 4 2 6</td>
<td>1 7 5 0</td>
<td>1 3 3 0</td>
</tr>
<tr>
<td>P₃</td>
<td>1 3 5 3</td>
<td>2 3 5 6</td>
<td>0 0 0 3</td>
</tr>
<tr>
<td>P₄</td>
<td>0 3 2 0</td>
<td>0 6 5 2</td>
<td>0 0 2 0</td>
</tr>
<tr>
<td>P₅</td>
<td>0 0 1 4</td>
<td>0 6 5 6</td>
<td>0 6 4 2</td>
</tr>
</tbody>
</table>

What if P₂ wants to change its allocation to <0, 4, 2, 0>?

Deadlock Avoidance Example
Banker's algorithm

Avoiding a "run on the bank"

- Simulate the effect of granting a process's resource allocation request
- Then check to see if a safe sequence exists

Issues

- Complexity?
Banker’s Algorithm
Interesting special cases

- Single instance resources
- Introduce a new edge into the RAG — a claim edge
  - Indicates that a process may request a resource in the future

- A request can be granted only if the conversion of a claim edge into an allocation edge does not create a cycle

Dealing With Deadlock
Deadlock detection & recovery

- Deadlock prevention & avoidance
  - Resource allocation protocols that prohibit deadlock

- The common approach
  - Let the system deadlock & then deal with it
    - Detect that a set of processes are deadlocked
    - Recover from the deadlock
Deadlock Detection & Recovery

Detecting deadlock

- Run Banker’s algorithm & see if a safe sequence exists
  - Replace MAX_REQUEST with simply “REQUEST”
  - If a safe sequence does not exist then the system is deadlocked

- How often should the OS check for deadlock?
  - After every resource request?
  - Only when we suspect deadlock has occurred?

Deadlock Detection & Recovery

Recovering from deadlock

- Abort all deadlocked processes & reclaim their resources

- Abort one process at a time until all cycles in the RAG are eliminated

- Where to start?
  - Low priority processes
  - Processes with the most resources
  - ...