Lecture 6

Deadlock

April 20, 2006

Prof. Joe Pasquale

Department of Computer Science and Engineering
University of California, San Diego
Before We Begin ...

Read Chapter 7 (on Deadlocks)

Programming Assignment #2
  • Due Sunday, April 23, midnight

Midterm Exam
  • May 11
  • Will cover all material up to that point
What is Deadlock?

The state of a set of permanently blocked processes

- Unblocking of one relies on progress of another
- But none can make progress!

Example

- Processes A and B
- Resources X and Y
- A holding X, waiting for Y
- B holding Y, waiting for X
- Each is waiting for the other; will wait forever
Traffic Jam as Example of Deadlock

Cars deadlocked in an intersection

Resource Allocation Graph
More Examples of Deadlock

Memory (a reusable resource)
- total memory = 200KB
- P1 requests 80KB
- P2 requests 70KB
- P1 requests 60KB (wait)
- P2 requests 80KB (wait)

Messages (a consumable resource)
- P1: receive M2 from P2
- P2: receive M1 from P1
Conditions for Deadlock

Mutual Exclusion
- Only one process may use a resource at a time

Hold-and-Wait
- Process holds resources while waiting for others

No Preemption
- Can’t take a resource away from a process

Circular Wait
- The waiting processes form a cycle
How to Attack the Deadlock Problem

Deadlock Prevention
• Make deadlock impossible by removing a condition

Deadlock Avoidance
• Avoid getting into situations that lead to deadlock

Deadlock Detection
• Don’t try to stop deadlocks
• Rather, if they happen, detect and resolve
Deadlock Prevention

Simply prevent any one of the conditions for deadlock

Mutual exclusion
  • Relax where sharing is possible

Hold-and-wait
  • Get all resources simultaneously (wait until all free)

No preemption
  • Allow resources to be taken away

Circular wait
  • Order all the resources, force ordered acquisition
How Can We Prevent a Traffic Jam?

What are the processes?
What are the resources?
How does deadlock occur?
How to prevent deadlock?
  • Add a traffic light
Which condition is being prevented?
Deadlock Avoidance

Avoid situations that lead to deadlock

• Selective prevention
• Remove condition only when deadlock a possibility

Works with incremental resource requests

• Resources are asked for in increments
• Do not grant request that can lead to a deadlock

Requires knowledge of maximum resource requirements
Banker’s Algorithm: Concepts

System has a fixed number of processes and resources
  • Each process has zero or more resources allocated

System state: either safe or unsafe
  • Depends on allocation of resources to processes

Safe state: deadlock is absolutely avoidable
  • Can avoid deadlock by certain order of execution

Unsafe state: deadlock is possible (but not certain)
  • May not be able to avoid deadlock
Safe, Unsafe, and Deadlock States

- Safe
- Unsafe
- Deadlock
Banker’s Algorithm

Given

• process/resource claim matrix
• process/resource allocation matrix
• resource availability vector

Is there a sequence of process executions such that

• a process can run to completion, return resources
• resources can then be used by another to complete
• eventually, all the processes complete?
Example of a Safe State

Current state

<table>
<thead>
<tr>
<th></th>
<th>Claim</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P_1$</td>
<td>$P_2$</td>
</tr>
<tr>
<td>$R_1$</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>$R_2$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$R_3$</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

This is a safe state

- Which process can run to completion? $P_2$
- After $P_2$ completes, its resources are returned
- Next select $P_1$, then $P_3$, then $P_4$
Example of an Unsafe State

Current state

<table>
<thead>
<tr>
<th></th>
<th>Claim</th>
<th>Allocation</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Avail</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>P_1 3</td>
<td>P_2 6</td>
<td>P_3 3</td>
<td>P_4 4</td>
<td>P_1 2</td>
<td>P_2 5</td>
<td>P_3 2</td>
<td>P_4 0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>R_2</td>
<td>P_1 2</td>
<td>P_2 1</td>
<td>P_3 1</td>
<td>P_4 2</td>
<td>P_1 0</td>
<td>P_2 1</td>
<td>P_3 1</td>
<td>P_4 0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>R_3</td>
<td>P_1 2</td>
<td>P_2 3</td>
<td>P_3 4</td>
<td>P_4 2</td>
<td>P_1 1</td>
<td>P_2 1</td>
<td>P_3 1</td>
<td>P_4 2</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

This is an unsafe state

- Can any process definitely run to completion? No!
- P_1 may block asking for R_1; same for P_2, P_3, and P_4
- Possible deadlock does not mean certain deadlock
How Do We Avoid a Traffic Jam?

What are the incremental resources?

What is a safe state?

How to avoid deadlock?
  • Allow at most 3 cars into intersection

Which condition is being prevented?
Deadlock Detection and Recovery

Don’t do anything special to prevent or avoid deadlocks
- If they happen, they happen
- Periodically, try to detect if a deadlock occurred
- Do something (or even nothing) about it

Reasoning
- Deadlocks rarely happen
- Cost of prevention or avoidance is not worth it
- Deal with them in special way (may be very costly)

Most general purpose OS’s take this approach!
Detecting Deadlocks

Construct resource allocation “wait-for” graph

- if cycle, deadlock

Requires

- identifying all resources
- tracking their use
- periodically running detection algorithm
Recovery from Deadlock

Abort all deadlocked processes
  • Will remove deadlock, but drastic and costly

Abort deadlocked processes one-at-a-time
  • Do until deadlock goes away (need to detect)
  • What order should processes be aborted?

What happens to resources in inconsistent states
  • such as files that are partially written?
  • or interrupted message (e.g., file) transfers?
Classical Synchronization Problems

The Producer/Consumer (Bounded Buffer) Problem
The Dining Philosophers Problem
The Readers/Writers Problem

Study these problems and their solutions!
The Dining Philosophers Problem

Five philosophers
  • Think, eat, think, eat, ...

To eat
  • Pick up two forks
    - one at a time
  • Eat
  • Put down forks

Mutual exclusion
  • Avoid deadlock or starvation
Implementing Dining Philosophers

Identify critical section(s)

How to achieve mutual exclusion?

How to avoid deadlock?

How to avoid starvation?

How to generalize to n philosophers?

DoPhilosopher (int i)
while (TRUE) {
    Think ();
    PickupFork (i);
    PickupFork ((i+1)%5);
    Eat ();
    PutdownFork ((i+1)%5);
    PutdownFork (i);
}
The Readers-Writers Problem

Multiple readers: processes that only read a file

Multiple writers: processes that modify a file

Rules

- Allow multiple simultaneous readers
- A writer gets exclusive access
- Avoid starvation
  - Once a writer arrives
    - wait until current readers leave
    - do not allow any new readers