Chapter 3

Deadlocks

3.1. Resource
3.2. Introduction to deadlocks
3.3. The ostrich algorithm
3.4. Deadlock detection and recovery
3.5. Deadlock avoidance
3.6. Deadlock prevention
3.7. Other issues

Resources

- Examples of computer resources
  - printers
  - tape drives
  - tables
- Processes need access to resources in reasonable order
- Suppose a process holds resource A and requests resource B
  - at same time another process holds B and requests A
  - both are blocked and remain so
Resources (1)

• Deadlocks occur when …
  – processes are granted exclusive access to devices
  – we refer to these devices generally as resources
• Preemptable resources
  – can be taken away from a process with no ill effects
• Nonpreemptable resources
  – will cause the process to fail if taken away
• alternatively: reusable and consumable resources

Resources (2)

• Sequence of events required to use a resource
  1. request the resource
  2. use the resource
  3. release the resource

• Must wait if request is denied
  – requesting process may be blocked
  – may fail with error code
Introduction to Deadlocks

• Formal definition:
  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

• Usually the event is release of a currently held resource

• None of the processes can …
  – run
  – release resources
  – be awakened

Four Conditions for Deadlock

1. Mutual exclusion condition
   - each resource assigned to 1 process or is available

2. Hold and wait condition
   - process holding resources can request additional

3. No preemption condition
   - previously granted resources cannot forcibly taken away

4. Circular wait condition
   - must be a circular chain of 2 or more processes
   - each is waiting for resource held by next member of the chain
Dealing with Deadlocks

• Ignore the problem - optimistic
  – unfair to ostriches
• Detect and Recover
  – detect there is a deadlock, take action
• Dynamically Avoid
  – allocate resources mindful of deadlock
• Prevent
  – preclude one of the four conditions

Deadlock Modeling

• Modeled with directed graphs

(a) resource R assigned to process A
(b) process B is requesting/waiting for resource S
(c) process C and D are in deadlock over resources T and U
How deadlock occurs

Deadlock Modeling (2)

Deadlock Modeling (3)

How deadlock can be avoided
The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- UNIX and Windows take this approach
- It is a trade off between
  - convenience
  - correctness

Detection with One Resource of Each Type (1)

- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock
Detection with One Resource of Each Type (2)

Data structures needed by deadlock detection algorithm

Detection with One Resource of Each Type (3)

An example for the deadlock detection algorithm
Recovery from Deadlock (1)

- **Recovery through preemption**
  - take a resource from some other process
  - depends on nature of the resource
- **Recovery through rollback**
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked

Recovery from Deadlock (2)

- **Recovery through killing processes**
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning
    - e.g., compilation
    - think idempotence
Deadlock Avoidance
Resource Trajectories

Two process resource trajectories

Safe and Unsafe States (1)

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>Free: Complexity 3</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>Free: Complexity 1</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>Free: Complexity 5</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>Free: Complexity 0</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>Free: Complexity 7</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td></td>
</tr>
</tbody>
</table>

Demonstration that the state in (a) is safe
Safe and Unsafe States (2)

Demonstration that the state in b is not safe

The Banker's Algorithm for a Single Resource

- Three resource allocation states
  - safe
  - safe
  - unsafe
Banker's Algorithm for Multiple Resources

Example of banker's algorithm with multiple resources

Deadlock Prevention
Attacking the Mutual Exclusion Condition

- Some devices (such as printer) can be spooled
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
  - avoid assigning resource when not absolutely necessary
  - as few processes as possible actually claim the resource
Attacking the Hold and Wait Condition

- Require processes to request resources before starting
  - a process never has to wait for what it needs

- Problems
  - may not know required resources at start of run
  - also ties up resources other processes could be using

- Variation:
  - process must give up all resources
  - then request all immediately needed

Attacking the No Preemption Condition

- This is not a viable option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer
  - !!!?
Attacking the Circular Wait Condition (1)

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

(a)

(b)  

• Normally ordered resources
• A resource graph

Summary of approaches to deadlock prevention

<table>
<thead>
<tr>
<th>Condition</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual exclusion</td>
<td>Spool everything</td>
</tr>
<tr>
<td>Hold and wait</td>
<td>Request all resources initially</td>
</tr>
<tr>
<td>No preemption</td>
<td>Take resources away</td>
</tr>
<tr>
<td>Circular wait</td>
<td>Order resources numerically</td>
</tr>
</tbody>
</table>

154 155
Other Issues
Two-Phase Locking

• Phase One
  – process tries to lock all records it needs, one at a time
  – if needed record found locked, start over
  – (no real work done in phase one)
• If phase one succeeds, it starts second phase,
  – performing updates
  – releasing locks
• Note similarity to requesting all resources at once
• Algorithm works where programmer can arrange
  – program can be stopped, restarted

Nonresource Deadlocks

• Possible for two processes to deadlock
  – each is waiting for the other to do some task
• Can happen with semaphores
  – each process required to do a down() on two semaphores (mutex and another)
  – if done in wrong order, deadlock results
Starvation

• Algorithm to allocate a resource
  – may be to give to shortest job first

• Works great for multiple short jobs in a system

• May cause long job to be postponed indefinitely
  – even though not blocked

• Solution:
  – First-come, first-serve policy