Deadlocks

- What are Deadlocks?
- How do they occur?
- How do we deal with them?

What is a Deadlock?

- 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

Why have those cars been abandoned?
Resources

• Examples of computer resources
  – printers
  – tape drives
  – tables, lists…

• 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

Resources

• Deadlocks can occur when …
  – processes are granted exclusive access to resources

• Preemptable resources
  – can be taken away from a process with no ill effects
  – therefore they don’t contribute to deadlock.

• Nonpreemptable resources
  – will cause the process to fail if taken away
Deadlock Modeling

Resource Allocation Graph with a single instance of each resource.

(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

Deadlock Modeling

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
deadlock

How deadlock occurs
Deadlock Modeling

How deadlock can be avoided

Resource-Allocation Graph with multiple instances of a resource

- Process
- Resource Type with 4 instances
- $P$ requests instance of $R$
- $P$ is holding an instance of $R$
Resource Allocation Graph

Resource Allocation Graph
Coffman’s Four *Necessary* Conditions for Deadlock

1. **Mutual exclusion**
   - A resource can be assigned to at most one process at a time.

2. **Hold and wait**
   - A process holding a resource is allowed to request additional resources

3. **No preemption**
   - Previously granted resources cannot be forcibly taken away from a process.

4. **Circular wait**
   - Must be a circular chain of 2 or more processes, each waiting for a resource held by another.
Strategies for dealing with Deadlocks

1. Ostrich Algorithm
2. Prevention
   • negating one of the four necessary conditions
3. Detection and recovery
4. Dynamic avoidance
   • careful resource allocation

[Note: Remember the distinction between Prevention and Avoidance.]

Ostrich Algorithm

• One “solution” to any problem is to ignore it.
• Reasonable, if solving the problem takes more effort than its worth.
  – How often does deadlock occur?
  – How disruptive is it?
  – Is it worth addressing all forms of deadlock?
  – How expensive is the solution?
• Tannenbaum suggests that Unix and NT take this approach to deadlock…
Deadlock Prevention

• To prevent deadlock
  – Remember the four requirements for deadlock
  – Negate any one of those requirements.

We'll look at each of the requirements and how they can be negated in the next few slides.

Deadlock Prevention
The Mutual Exclusion Condition

• Some devices (such as printer) can be spooled
  – This way, only the printer daemon actually uses printer resource
  – deadlock for the printer is eliminated
• Not all resources can be spooled
• Principle:
  – avoid assigning a resource to a process when not absolutely necessary
  – Allow as few processes as possible actually claim the resource
Deadlock Prevention
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 that are immediately needed

Deadlock Prevention
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
  - !!??
Deadlock Prevention
Circular Wait Condition

- Assign an ordering to all the resources. 
  E.g:
  1. Printer
  2. Scanner
  3. Plotter
  4. CD-ROM drive
  5. Tape Drive

- Require that processes only request them in order!

Summary of approaches to Deadlock Prevention

<table>
<thead>
<tr>
<th>Condition</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hold and wait</td>
<td>Request all resources initially or give up any resources held before requesting more</td>
</tr>
<tr>
<td>Mutual Exclusion</td>
<td>Spool</td>
</tr>
<tr>
<td>No Preemption</td>
<td>Not feasible</td>
</tr>
<tr>
<td>Circular Wait</td>
<td>Assign an ordering to resources, and only allow requests in order</td>
</tr>
</tbody>
</table>
Detecting And Recovering

• How to Detect?
• How to Recover?

Detection with **Only One** Resource of Each Type

• Note the resource ownership and requests
• A cycle can be found within the graph, denoting deadlock
Detection with Multiple Resources of Each Type

Data structures needed by deadlock detection algorithm

Detection with Multiple Resources of Each Type

An example for the deadlock detection algorithm
Recovery from Deadlock

- Recovery through preemption
  - take a resource from some other process
  - But these were non-preemptible, right?
- Recovery through rollback
  - “checkpoint” all processes periodically
  - This means that we save their state on disk somewhere. That would include the state of any files they have open...
  - “rollback” one of the processes causing deadlock to a time when it didn’t hold the needed resource. All work that was done by the process after that time is lost.

Recovery from Deadlock

- 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. Kind of an extreme form of “rollback”.
Avoid Deadlocks by OS Vigilance

• When asked for a resource, the OS has to decide if it is safe to grant the request.
• OS will need to know the maximum resources every process is going to need.
  – This may seem unrealistic, but batch jobs frequently are required to state their possible requirements ahead of time.

Deadlock Avoidance
Resource Trajectories

![Diagram showing resource trajectories]
Safe, Unsafe, Deadlock State

Safe and Unsafe States

Demonstration that the state in (a) is safe
Starting with the same state (a) as before, can we grant Process A another resource?

The Banker's Algorithm for a Single Resource

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Tape drives</th>
<th>Printers</th>
<th>Scanners</th>
<th>CD ROMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Resources assigned

<table>
<thead>
<tr>
<th>Process</th>
<th>Tape drives</th>
<th>Printers</th>
<th>Scanners</th>
<th>CD ROMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Resources still needed

Example of banker's algorithm with multiple resources

E = (6342)
P = (5322)
A = (1020)