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
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
Deadlock Modeling (2)

- Modeled with directed graphs

- resource R assigned to process A
- process B is requesting/waiting for resource S
- process C and D are in deadlock over resources T and U
Deadlock Modeling (3)

Strategies for dealing with Deadlocks

1. just ignore the problem altogether
2. detection and recovery
3. dynamic avoidance
   • careful resource allocation
4. prevention
   • negating one of the four necessary conditions
Deadlock Modeling (4)

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 (5)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock

(k)

How deadlock can be avoided

(o)  (p)  (q)
The Ostrich Algorithm

• Pretend there is no problem
• Reasonable if
  – deadlocks occur very rarely
  – cost of prevention is high
• UNIX and Windows takes 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

Resources in existence
\((E_1, E_2, E_3, \ldots, E_m)\)

Current allocation matrix

\[
\begin{bmatrix}
C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\
C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm}
\end{bmatrix}
\]

Row \(n\) is current allocation to process \(n\)

Resources available
\((A_1, A_2, A_3, \ldots, A_m)\)

Request matrix

\[
\begin{bmatrix}
R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\
R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm}
\end{bmatrix}
\]

Row 2 is what process 2 needs
Detection with One Resource of Each Type (3)

An example for the deadlock detection algorithm

\[
E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}
\]

Current allocation matrix

\[
C = \begin{bmatrix}
0 & 0 & 1 & 0 \\
2 & 0 & 0 & 1 \\
0 & 1 & 2 & 0
\end{bmatrix}
\]

Request matrix

\[
R = \begin{bmatrix}
2 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0
\end{bmatrix}
\]
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
Deadlock Avoidance

Resource Trajectories

Two process resource trajectories
Safe and Unsafe States (1)

 Demonstration that the state in (a) is safe

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 3

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 1

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 5

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 0

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>

Free: 7

[Source: CuuDuongThanCong.com](https://fb.com/tailieudientucntt)
## Safe and Unsafe States (2)

### Demonstration that the state in b is not safe

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 3

(a)

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 2

(b)

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 0

(c)

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Free: 4

(d)
The Banker's Algorithm for a Single Resource

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

Free: 10

(a)

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
</tbody>
</table>

Free: 2

(b)

<table>
<thead>
<tr>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
</tbody>
</table>

Free: 1

(c)

- 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

- Normally ordered resources
- A resource graph
### Attacking the Circular Wait Condition (1)

**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>
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