Chapter 6

Deadlock

(Part 1)
Resources and Deadlocks

Processes need access to resources in order to make progress.

Examples of Resources:

- Kernel Data Structures
  (ProcessControlBlocks, Threads, OpenFile…)
- Locks/semaphores to protect critical sections
- Memory (page frames, buffers, etc.)
- Files
- I/O Devices
  (printers, ports, tape drives, speaker, etc.)
Resources and Deadlocks

Scenario:

Process P1...
is holding resource A, and
is requesting resource B

Process P2...
is holding resource B, and
is requesting resource A

Both are blocked and remain so …

This is \textit{deadlock}
Resource Usage Model

Sequence of events required to use a resource:
  request the resource (e.g., acquire a mutex lock)
  use the resource
  release the resource (e.g., release a mutex lock)

Must wait if request is denied
  block
  busy wait
  fail with error code
Preemptable vs Nonpreemptable Resources

**Preemptable resources**
Can be taken away from a process with no ill effects

**Nonpreemptable resources**
Once given to the process, can’t be taken back
Will cause the process to fail if taken away

“Deadlocks occur when processes are granted exclusive access to non-preemptable resources and wait when the resource is not available.”
Definition of Deadlock

“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:

The release of a currently held resource

All processes in the set are waiting

... for a resource request to be granted.

None of the processes can proceed

... so no process can release the resources it holds.
Starvation vs. Deadlock

*Starvation* and *Deadlock* are two different things!

**Deadlock:**
- No work is being accomplished for the processes that are deadlocked, because processes are waiting for each other. Once present, will not go away!

**Starvation:**
- Work (progress) is occurring. However, a particular set of processes may not be getting any work done because they cannot obtain the resources they need.
- May only last a short time; may go away.

Both are probabilistic events & may occur only rarely.
Deadlock Conditions

A deadlock situation can occur *if and only if* the following conditions hold simultaneously...

**Mutual Exclusion Condition**
A resource can be assigned to only one process at a time

**Hold And Wait Condition**
Processes can get more than one resource

**No Preemption Condition**

**Circular Wait Condition**
A cyclic chain of two or more processes (must be waiting for resource from next one in chain)
Resource acquisition scenarios

**Thread A:**

acquire (resource_1)
use resource_1
release (resource_1)

**Example:**

```go
var r1_mutex: Mutex
...

r1_mutex.Lock()
Use resource_1
r1_mutex.Unlock()
```
Resource acquisition scenarios

**Thread A:**

- acquire (resource_1)
- use resource_1
- release (resource_1)

**Another Example:**

```plaintext
var r1_sem: Semaphore
r1_sem.Up()
...
r1_sem.Down()
Use resource_1
r1_sem.Up()
```
Resource acquisition scenarios

**Thread A:**
- acquire (resource_1)
- use resource_1
- release (resource_1)

**Thread B:**
- acquire (resource_2)
- use resource_2
- release (resource_2)
Resource acquisition scenarios

**Thread A:**
- acquire (resource_1)
- use resource_1
- release (resource_1)

**Thread B:**
- acquire (resource_2)
- use resource_2
- release (resource_2)

*No deadlock can occur here!*
## Resource Acquisition Scenarios: 2 Resources

<table>
<thead>
<tr>
<th>Thread A:</th>
<th>Thread B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire (resource_1)</td>
<td>acquire (resource_1)</td>
</tr>
<tr>
<td>acquire (resource_2)</td>
<td>acquire (resource_2)</td>
</tr>
<tr>
<td>use resources 1 &amp; 2</td>
<td>use resources 1 &amp; 2</td>
</tr>
<tr>
<td>release (resource_2)</td>
<td>release (resource_2)</td>
</tr>
<tr>
<td>release (resource_1)</td>
<td>release (resource_1)</td>
</tr>
</tbody>
</table>
Resource Acquisition Scenarios: 2 Resources

<table>
<thead>
<tr>
<th><strong>Thread A:</strong></th>
<th><strong>Thread B:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire (resource_1)</td>
<td>acquire (resource_1)</td>
</tr>
<tr>
<td>acquire (resource_2)</td>
<td>acquire (resource_2)</td>
</tr>
<tr>
<td>use resources 1 &amp; 2</td>
<td>use resources 1 &amp; 2</td>
</tr>
<tr>
<td>release (resource_2)</td>
<td>release (resource_2)</td>
</tr>
<tr>
<td>release (resource_1)</td>
<td>release (resource_1)</td>
</tr>
</tbody>
</table>

No deadlock can occur here!
Resource Acquisition Scenarios: 2 Resources

**Thread A:**
- acquire (resource_1)
- use resources 1
- release (resource_1)
- acquire (resource_2)
- use resources 2
- release (resource_2)

**Thread B:**
- acquire (resource_2)
- use resources 2
- release (resource_2)
- acquire (resource_1)
- use resources 1
- release (resource_1)
Resource Acquisition Scenarios: 2 Resources

**Thread A:**
- acquire (resource_1)
- use resources 1
- release (resource_1)
- acquire (resource_2)
- use resources 2
- release (resource_2)

**Thread B:**
- acquire (resource_2)
- use resources 2
- release (resource_2)
- acquire (resource_1)
- use resources 1
- release (resource_1)

*No deadlock can occur here!*
Resource Acquisition Scenarios: 2 Resources

**Thread A:**
- acquire (resource_1)
- acquire (resource_2)
- use resources 1 & 2
- release (resource_2)
- release (resource_1)

**Thread B:**
- acquire (resource_2)
- acquire (resource_1)
- use resources 1 & 2
- release (resource_1)
- release (resource_2)
Resource Acquisition Scenarios: 2 Resources

<table>
<thead>
<tr>
<th>Thread A:</th>
<th>Thread B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire (resource_1)</td>
<td>acquire (resource_2)</td>
</tr>
<tr>
<td>acquire (resource_2)</td>
<td>acquire (resource_1)</td>
</tr>
<tr>
<td>use resources 1 &amp; 2</td>
<td>use resources 1 &amp; 2</td>
</tr>
<tr>
<td>release (resource_2)</td>
<td>release (resource_1)</td>
</tr>
<tr>
<td>release (resource_1)</td>
<td>release (resource_2)</td>
</tr>
</tbody>
</table>

Deadlock is possible!
Examples of Deadlock

• Deadlock occurs within a single application
  ...Not so bad
  Programmer created a situation that deadlocks
  Kill the program and move on

• Deadlock occurs within the OS
  ...More of a problem
  System crashes, or some threads become frozen
  Must restart system (i.e., kill every thread)
Other examples of deadlock
Resource Allocation Graphs

Process/Thread

Resource

A

R
Resource Allocation Graphs
Resource Allocation Graphs

Process/Thread → A → S → Resource

Resource

“is requesting”
Resource Allocation Graphs

A -> S
R <-> B
Resource Allocation Graphs

Deadlock
Resource Allocation Graphs

Deadlock = a cycle in the graph
Multiple Units of a Resource

Some resources have only one “unit”. Only one thread at a time may hold the resource.
- Printer
- Lock on ProcessTable

Some resources have several units. All units are considered equal; any one will do.
- Page Frames
- Dice

A thread requests “k” units of the resource. Several requests may be satisfied simultaneously.
Dealing with deadlock

General strategies

Ignore the Problem
  Hmm… advantages, disadvantages?

Detection and Recovery

Avoidance
  through careful resource allocation

Prevention
  by structurally negating one of the four conditions
Deadlock detection (1 unit of each)

Let the problem happen, then recover

How do you know it happened?

Do a depth-first-search on the resource allocation graph
Deadlock detection (1 unit of each)

Let the problem happen, then recover

How do you know it happened?

Do a depth-first-search on the resource allocation graph
Deadlock detection (1 unit of each)

Let the problem happen, then recover

How do you know it happened?

Do a depth-first-search on the resource allocation graph
Deadlock detection (1 unit of each)

Let the problem happen, then recover

How do you know it happened?

Do a depth-first-search on the resource allocation graph
Deadlock detection (1 unit of each)

Let the problem happen, then recover

How do you know it happened?

Do a depth-first-search on the resource allocation graph
Deadlock modeling with multiple resources

**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?
Deadlock modeling with multiple resources

**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?
**Deadlock Detection (multiple resources)**

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
### Example

The current allocation matrix and the request matrix are given as follows:

**Current allocation matrix**

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

**Request matrix**

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

Is there a sequence of running the processes such that all the resources will be returned?

1. Look for an unmarked process \( P_i \), for which the \( i \)th row of \( R \) is less than or equal to \( A \)

2. If such a process is found, add the \( i \)-th row of \( C \) to \( A \), mark the process and go back to step 1

3. If no such process exists the algorithm terminates

If all marked, no deadlock!
Deadlock Detection Algorithm - Example

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

\[ A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix} \]

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

Request matrix
\[ R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \]
Deadlock Detection Algorithm - Example

<table>
<thead>
<tr>
<th></th>
<th>Tape drives</th>
<th>Plotters</th>
<th>Scanners</th>
<th>CD Roms</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Tape drives</th>
<th>Plotters</th>
<th>Scanners</th>
<th>CD Roms</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Current allocation matrix

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

Request matrix

\[
R = \begin{bmatrix}
2 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0
\end{bmatrix}
\]
Deadlock Detection Algorithm - Example

E = ( 4  2  3  1 )
A = ( 2  1  0  0 )

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}
Deadlock Detection Algorithm - Example

E = (4 2 3 1)  
A = (2 1 0 0)

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

Request matrix
R = \[
\begin{bmatrix}
2 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0 \\
\end{bmatrix}
\]
### Deadlock Detection Algorithm - Example

#### Current allocation matrix

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

#### Request matrix

$$
R = \begin{bmatrix}
2 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0 \\
\end{bmatrix}
$$

#### Vectors

$$
E = (4, 2, 3, 1) \\
A = (2, 1, 0, 0) \\
$$

#### Result

$$
2 \quad 2 \quad 2 \quad 0
$$
Deadlock Detection Algorithm - Example

E = (4 2 3 1)

A = (2 1 0 0)

2 2 2 0

4 2 2 1

Current allocation matrix

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

Request matrix

\[
R = \begin{bmatrix}
2 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0
\end{bmatrix}
\]
# Deadlock Detection Algorithm - Example

<table>
<thead>
<tr>
<th>Tape drives</th>
<th>Plotters</th>
<th>Scanners</th>
<th>CD Roms</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ E = (4 \ 2 \ 3 \ 1) \]

\[ A = (2 \ 1 \ 0 \ 0) \]

\[
\begin{array}{cccc}
2 & 2 & 2 & 0 \\
4 & 2 & 2 & 1 \\
\end{array}
\]

**Current allocation matrix**

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

**Request matrix**

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

No deadlock!
Deadlock Detection Issues

How often should the algorithm run?

• After every resource request?

• Periodically?

• When CPU utilization is low?

• When we suspect deadlock?

• When some process/thread has been asleep for a long time?
Recovery from Deadlock

What should be done to recover?
- Abort deadlocked processes and reclaim resources
- Temporarily reclaim resource, if possible
- Abort one process at a time until deadlock cycle is eliminated

Where to start?
Low priority processes
How long process has been executing
How many resources a process holds
Batch or interactive
Number of processes that must be terminated
Other Deadlock Recovery Techniques

Recovery through rollback

• Save state periodically
  (Take a “checkpoint”)

• Need to kill a process?
  Start computation again from checkpoint

• Done for large computation tasks