Chapter 6

Deadlock Problems

Deadlock Situations - Examples
A Systems Model
Serially Reusable Resources
Deadlock Detection
Recovery and Prevention
Consumable Resources
Resource Graphs
Dynamic Addition and Removal

Policy versus Mechanism

- Separate what is allowed to be done with how it is done
  - a process knows which of its children threads are important and need priority
- Requests for resources
- Scheduling of requests
- Allocation of resources
- Release of resources
Deadlock – reasons, occurrences

- Processes compete for “resources”
- Sharing of resources among competitors
- Sequence of allocations can lead to blocking
- Dynamic resource sharing, parallel programming and communicating processes are critical

Deadlock – “deadly embrace”

- Resource: reusable, scarce, stable commodity (hw,sw)
  - serially reusable:
    * request
    * use
    * release
  on a serial basis
  - consumable:
    * producer
    * “message-send/receive”
    * consumer
  resource with a short lifetime
**“deadly embrace” - deadlock**

\[
\begin{align*}
p1: & \quad \text{Request}(D); \quad \text{Request}(T); \\
r1: & \quad \text{Request}(T); \quad \text{Request}(D); \\
p2: & \quad \text{Release}(D); \quad \text{Release}(T);
\end{align*}
\]

**Reusable Resources**

- Reusable resources are permanent objects
  - The number of units within a class is constant
  - Each unit is either available, or is allocated to one and only one process
  - A process must first request and acquire a resource unit before it can release that unit
Consumable Resources

- Reusable resources are produced and consumed dynamically
  * The number of units within a class varies at runtime
  * A process may increase the number of units of a resource class by releasing one or more units without acquiring them first
  * A process may decrease the number of units of a resource class by requesting and acquiring one or more units

Deadlock on File Access

```
p1: p2:
   :    :
   :    :
   a: open(f1, w);  c: open(f2, w);
b: open(f2, w);  d: open(f1, w);
   :    :
   :    :
```
“deadly embrace” - deadlock

\[
\begin{align*}
  p1: & & p2: & & p3: \\
  & : & : & : \\
  & : & : & : \\
  \text{if } (C) & \text{send}(p2,m); & \text{receive}(p3,m); & \text{receive}(p1,m); & \text{receive}(p2,m); \\
  \text{LOOP } \{} & \text{LOOP } \{} & \text{LPOOP } \{} \\
  \text{receive}(p3,m); & \text{receive}(p1,m); & \text{receive}(p2,m); & \text{send}(p2,m); & \text{send}(p3,m); & \text{send}(p1,m); \\
  & : & : & : \\
  & : & : & : \\
  \end{align*}
\]

Deadlock on Memory Allocation

\[
\begin{align*}
  p1: & & p2: \\
  & : & : \\
  & : & : \\
  a: & \text{Get Mem}(2); & c: & \text{Get Mem}(1); \\
  b: & \text{Get Mem}(2); & d: & \text{Get Mem}(2); \\
  & : & : & : \\
  & : & : & :
\end{align*}
\]
Strategies - Approaches

• Ignore

• Handle
  – Detect & recover
  – Avoid
  – Prevent

Conditions for Deadlocks

1. Mutual exclusion condition: each resource is either currently assigned to exactly one process or it is available.

2. Hold and Wait condition: processes currently holding resources allocated earlier can request new resources.

3. No preemption condition: resources previously acquired cannot be forcibly taken away from processes; they must be explicitly released by the process holding them.

4. Circular wait condition: there must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.
Resources

- Two basic categories
  - “reusable” resources – e.g. peripherals, printer
  - “consumable” resources – e.g. messages

- Two basic operations
  - request/allocate
  - release

Resource Graphs

- Elements and Relationships
  - Resources
  - Processes
  - Resource Requests
  - Resource Allocations

- Notational Elements
  - Vertices
  - Edges
Resource Graphs

(a) Process holding a resource

(b) Process requesting a resource

(c) Deadlocked processes

example of a resource graph

deadlock with SR resources
Deadlock Occurrence

$P_1$: requests $R_1$
$P_1$: requests $R_2$
$P_1$: releases $R_1$
$P_1$: releases $R_2$

$P_2$: requests $R_2$
$P_2$: requests $R_3$
$P_2$: releases $R_2$
$P_2$: releases $R_3$

$P_3$: requests $R_3$
$P_3$: requests $R_1$
$P_3$: releases $R_3$
$P_3$: releases $R_1$

3 concurrent processes accessing common resources

Deadlock Occurrence - sequence 1

$P_1$: requests $R_1$
$P_2$: requests $R_2$
$P_3$: requests $R_3$
**Deadlock Occurrence - sequence 1**

- **p₁** requests **R₂**
- **p₂** requests **R₃**
- **p₃** requests **R₁**

**Deadlock Occurrence - sequence 2**

- **p₁** requests **R₁**
- **p₂** requests **R₃**
- **p₃** requests **R₃**
Deadlock Occurrence – sequence 2

Deadlock Strategies

1. Just ignore the problem – hopefully it ignores us too.

2. Detection and recovery: let deadlocks occur, then detect them and take action.

3. Dynamic avoidance by careful resource allocation.

4. Prevention by structurally negating one of the four conditions necessary for a deadlock.
(a) simple example of a deadlock situation

(b) single resource sharing (n=3, m=2)

deadlock with SR resources

States and Transitions

- Requests
- Acquisitions
- Releases
- Deadlock states and safe states
process operations

State transitions caused by process p₁
Research on Deadlocks

Quotes from Andrew Tanenbaum:
"If there ever was a subject that was investigated mercilessly during the early days of computing, it was deadlocks. ...
Deadlock detection is a nice little graph theory problem that one mathematically inclined graduate student can get his jaws around and chew on for 3 or 4 years. All kinds of algorithms were devised, each one more exotic and less practical than the previous one. ...
When an operating system wants to do deadlock detection or prevention, which few of them do, they use one of the methods discussed"
Research on Deadlocks

Final quote from Andrew Tanenbaum:

“There is still a little research on distributed deadlock detection. ... None of it is even remotely practical in real systems. Its main function seems to be keeping otherwise unemployed graph theorists off the streets.”

Deadlock states and safe states

- Blocked processes
- Deadlocked processes
- Deadlock states
- Safe states
**Compute and IO Process**

```
::
p_c: Request(M, all); p_{io}: Request(IO_request);
Release(IO_request);
r_c: Request(IO_completion); r_{io}: Request(D);
::
::
::
Release(M, all); Perform_IO;
::
Release(IO_completion);
```
System Model

- System $\langle \sigma, \pi \rangle$ of states $\sigma \{S, T, U, V, \ldots \}$ and a set $\pi$ of processes $\{p_1, p_2, \ldots \}$
- A process $p_i : \sigma \rightarrow \{ \sigma \}$
  - $p_i : S \rightarrow^i W$; $S \rightarrow^* W$;
  - $p_i$ is blocked $\Rightarrow \not\exists S \rightarrow^i T$
- A process $p_i$ is deadlocked,
  $\forall T : S \rightarrow^i T p_i$ is blocked.

$S$ is a deadlock state if a process $p_i$ is deadlocked in $S$. 

"granularity"

(a) SR an CR resource sharing

(b) single resource sharing

deadlock with CR resources
examples of systems

(a) system with deadlock states:

p1 is blocked here

p2 is deadlocked in U and in V since it is blocked in either one;

(b) system with deadlock and safe states

deadlock state

not safe

safe states

system state changes in file sharing example
Serially Reusable Resources

- A SR resource is a finite set of identical units:
  - The number of units is constant
  - Each unit is either available or allocated to one an only one process – no sharing is allowed.
  - A process may release a unit only if it has previously acquired that unit.

- Representation by a process-resource graph (reusable resource graph)
  - A set of two mutually exclusive subsets of process and resource nodes [containing one or more units]
  - The graph is bipartite
  - The edges are directed [allocated/requested] and limited

Deadlock Detection

- Process progression - investigate the most favorable state changes
- Is there a sequence of operations such that the system will NOT deadlock
- Simulate the most favorable execution of each unblocked process in a sequential mode:
  - An unblocked process acquires all resources it needs, then releases all of its resources and remains dormant thereafter
  - The released resources might wake up some previously blocked processes
  - Continue in this way until there are no more unblocked processes
  - If at termination there remain some blocked processes the original state is a deadlock state;
example of a resource graph reduction

Graph reduction - 1

example of a resource graph reduction

Graph reduction - 2
example of a resource graph reduction

Graph reduction - 3

Another sequence of reductions

Reduced by $p_1$

Reduced by $p_2$

unblocked

unblocked
Deadlock Theorem

- Necessary and sufficient condition:
  - A state $S$ is a deadlock state IFF the reusable resource graph is NOT completely reducible;
  - Based on the observation (Lemma 1) that all reduction sequences lead to the same irreducible graph.
  - If $\text{Reach}(n)$ is the set of nodes reachable from node $n$ then the graph contains a cycle if $n \ni \text{Reach}(n)$;
  - A cycle in a graph is a necessary condition for deadlock.
  - A knot is a subset of nodes which are reachable from every node inside but no outside nodes are reachable.
  - Reducible resource graphs can be represented in matrix or list form (compression techniques to save storage).
Deadlock Detection

- Reduction of resource graphs

- Special cases
  - Testing for a specific process
  - Continuous deadlock detection
  - Immediate allocation (knots)
  - Single unit resources

- Deadlock detection in distributed systems
  - Central coordinator
  - Distributed coordination
Recovery from Deadlock

- Recovery through preemption
- Recovery through rollback (checkpoints)
- Recovery through process termination

2 general approaches
- Process termination
- Resource preemption

Termination costs:
- Process priority
- Restarting and re-running the process
- “external” costs

Termination schemes:
- Least expensive process
- Minimum cost process
- Apply resource preemption
Dynamic Deadlock Avoidance

- Claim graphs
- Banker's Algorithm