Deadlock and Livelock

Ingo Sander
ingo@imit.kth.se

Deadlock

- Deadlock can occur in an interconnection network, when a group of agents (usually packets) cannot make progress, because they are waiting on each other to release resource (buffers, channels)
- If a sequence of waiting agents form a cycle the network is deadlocked

Deadlock Example

- Connection A holds channels $u$ and $v$ and wants to acquire channel $w$
- Connection B holds channels $w$ and $x$ and wants to acquire channel $u$
- Since neither connection A nor connection B will release their channels there is a deadlock in the network

Deadlock

- Deadlock paralyzes the network, which can have catastrophic consequences
- Two possible solutions
  - Avoid deadlocks
  - Recover from deadlocks
- Almost all networks today use deadlock avoidance
**Livelock**

- In livelock packets continue to move through the network, but do not make progress to their destination.
- Lifelock has to be considered, if packets are allowed to take non-minimal routes through a network.
- It can be avoided by limiting the number of times a packet can be misrouted.

**Agents and Resources**

- Depending on the type of connections different agents and resources are involved.

<table>
<thead>
<tr>
<th>Flow Control</th>
<th>Agent</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Switching</td>
<td>Connection</td>
<td>Physical Channel</td>
</tr>
<tr>
<td>Packet-Buffer</td>
<td>Packet</td>
<td>Packet Buffer</td>
</tr>
<tr>
<td>Flit-Buffer</td>
<td>Packet</td>
<td>Virtual Channel</td>
</tr>
</tbody>
</table>

**Wait-For and Hold-Relations**

- Agents and resources are related by *wait-for* and *hold* relations.
- Agent A
  - Holds resources $u$, $v$
  - Waits for resource $w$
- Agent B
  - Holds resources $w$, $x$
  - Waits for resource $w$

- If an agent holds a resource than the resource can be viewed as “waiting” for the agent to release it.
- Thus each hold relation induces a wait-for relation in the opposite direction.
**Wait-For and Hold-Relations**

- Replacing the holds with wait for in the opposite direction, the lower figure is generated.
- The arrows in the figure reveal a cycle that shows that the configuration is deadlocked.

**Cycles mean Deadlock**

- Deadlock occurs, if
  1. Agents hold and do not release a resource, while waiting for another.
  2. A cycle exists between waiting agents, such that there exists a set of agents $A_0, A_1, \ldots, A_{n-1}$, where the agent $A_i$ holds resource $R_i$ while waiting for resource $R_{(i+1 \mod n)}$ for $i = 0, 1, \ldots, n-1$.

**Resource Dependences**

- In case two resources $R_i$ and $R_{i+1}$ are two edges apart in the wait-for graph, it is possible for an agent $A_i$ holding resource $R_i$ to wait indefinitely on resource $R_{i+1}$, if $R_{i+1}$ is never released.
- Whenever it is possible for an agent holding $R_i$ to wait on $R_{i+1}$, we say that a resource dependence exists from $R_i$ to $R_{i+1}$, and denote it as $R_i \succ R_{i+1}$.

**Resource Dependences**

- A cycle in the resource dependence graph indicates that it is possible for a deadlock to occur.
- A cycle in this graph is a necessary, but not sufficient condition for deadlock.
Another example

- Again the four node network is taken as example, but this time packet buffer flow with a single packet buffer per node is used
- Agents are packets
- Shared resources are now buffers

Resource Dependence Graph

Another example

- Again the four node network is taken as example, but this time packet buffer flow control with a single packet buffer per node is used
- Agents are packets
- Shared resources are now buffers
- The resource dependence graph says that deadlock might occur

Resource Dependence Graph

Another example

- Once again, the resource dependence graph only shows the possibility for deadlock
- The upper configuration is deadlocked
- The lower one is not! Packet 3 can acquire buffer 0

Wait-for Graph

Another example

- Again the four node network is taken as example, but this time flit buffer flow control with two virtual channels per physical channel is used
- Agents are packets
- Shared resources are virtual channels
- The resource dependence graph says that deadlock might occur

Resource Dependence Graph
... and another one

The example shows a deadlocked configuration:
- Packet $P_0$ holds virtual channel $u_0$ and $v_0$ and tries to acquire $w_0$
- Packet $P_1$ holds virtual channel $w_0$ and $x_0$ and tries to acquire $u_0$
- Though there are free virtual channel resources there is a cycle in the network.

Deadlock Avoidance

- Deadlock can be avoided by eliminating cycles in the resource dependence graph.
- This can be done by imposing a partial order on the resources and then insisting on that agents take these resource in ascending order.
- Then there is no possibility for a cycle, since in any cycle at least one agent that hold a higher-number resource must wait on a lower-numbered resource, but this is not allowed by the ordering relation.

Distance Classes

- Resources are grouped into numbered classes and restrict allocation of resources so that they only can be acquired in ascending order.

Distance Classes Example

- A packet at distance $i$ from its source node needs to allocate a resource from class $i$.
- At each hop, the packet acquires a hop from the next higher class.
Distance Classes Example

- Using distance classes the resource dependence graph can look like this.
- There are no cycles => deadlock cannot occur!

Distance Classes

- Distance classes provide a very general way to order resource in any topology.
- Distance Classes are very costly, since they require a number of buffers (or virtual channels) proportional to the diameter of the network.
- However, for some topologies the cost can be reduced significantly because of specific topology properties.

Dateline Classes

- For a ring the number of needed classes can be much reduced.
- Each node has only two buffers:
  - Class “0” buffer: $B_0$
  - Class “1” buffer: $B_1$
- Packets enter the ring in node $B_0$
- When they cross the dateline, they are placed into buffer $B_1$ until they reach their destination.
- Result is an acyclic graph => Deadlock cannot occur!

Restricted Physical Routes

- Dividing the network into different classes allows to create a deadlock free network, but can be very costly to the large number of resources needed.
- An alternative is to restrict the routing function with the objective to generate a dependence graph that is acyclic.
**Dimension Order Routing**

- Dimension Order Routing guarantees deadlock-freedom in $k$-ary $n$-meshes.
- Within the first dimension (here $x$) a packet traveling in $+x$ direction can only wait on a channel on the $+x$, $+y$, and $-y$ direction.
- In the second dimension a packet traveling on the $+y$ direction can only wait on a channel on the $+y$ direction.
- These relations can be used to number the channels, so that every packet follows increasingly numbered channels.

**The Turn Model**

- A more general model for Mesh-networks is the "Turn Model".
- The eight possible turns in a 2D-Mesh can be combined to create 2 abstract cycles.
- In order to avoid deadlock at least one turn must be removed for each cycle.

**Dimension Order Routing**

- Only the following turns are allowed (x-y routing) in dimension order routing ($x$-direction is routed first)

```
Counterclockwise
  
  Clockwise
```

**The Turn Model**

- Assume we remove the N-W turn in the counter-clockwise graph.
- In the clockwise direction either the S-W, N-E or E-S turn can be eliminated in order to yield a deadlock-free network (Turn W-N cannot be eliminated! Why?)

```
Counterclockwise

Clockwise

West-First
North-Last
Negative-Last
```
**Turn Model West First**

- In the West-First model a packet has first to make all its west turns (1)
- Packets shall be routed up (clockwise) or down (counterclockwise) before making a turn to the east (2)
- Packet shall then be routed to the east (3)
- Packets shall be routed down (clockwise) or up (counterclockwise) if needed (4)
- This scheme shall be reflected by the numbering!

**Deadlock Recovery**

- Deadlock recovery needs less resources than deadlock avoidance
- A deadlock must be first detected
  - A cycle in the “wait-for” graph indicates a deadlock
  - Detection is often done by means of timeout counters
- … and then the deadlock must be resolved
  - Either packets or connections are removed from the network
  - Packets that are deadlocked can enter an “escape buffer” that is used to resolve the deadlock

**Livelock**

- Can be caused by a non-minimal routing algorithm where a packet never reaches its destination due to misrouting
- Can also be caused by dropping flow techniques where the same packet is always dropped

**Livelock**

- There are two main techniques to avoid livelock:
  - Deterministic Avoidance
    - A state is added to a packet to ensure its progress
    - Misroute count or age of packet
    - Packet with higher age or misroute count wins arbitration
  - Probabilistic Avoidance
    - If it can be guaranteed that the probability of packet delivery approaches one for infinite time there is a guarantee to avoid livelock
    - Network can be considered livelock free, if there is a non-zero probability of a packet moving towards its destination (and the sum of these probabilities must approach one for infinite time)