Verification of Real Time Systems - CS5270
4th lecture

Hugh Anderson

National University of Singapore
School of Computing

February, 2007
A warning...

http://fly.to/gripen
Outline

1. Administration
   - Assignment 1
   - The road map...

2. Resource access
   - Priority Inversion
   - Priority Inheritance Protocol
   - Priority Ceiling Protocol
Assignment number 1: Correction

• Hand in Feb 15 - during lecture
Where we are so far

Three topics so far

- **RT systems**
  - Motivation, modelling vs synthesis, hard vs soft, RT architectures

- **The real-time computing environment**
  - Temporal accuracy, clocks
  - TTP? time triggered protocols

- **Scheduling**
  - Preemption, feasibility, schedulability
Formal basis for Uppaal:

- **Complete the scheduling part..**
  - PIP and PCP (today).

- **Formal basis for Uppaal**
  - Detailed study of a basis for efficient real-time analysis/model checking
  - Transition systems,
  - Automata,
  - Model checking
  - Timed transition systems,
  - Zones/regions (efficient timed systems)

This will all take some time... Perhaps 4/5 weeks
The immediate road map

After completing scheduling... four topics:

- **State transition systems**
  - some definitions
  - parallel composition

- **Timed transition systems**
  - formal definition
  - parallel composition
  - Reduction of a TTS (which has possibly infinite states and actions) to a finite TS by quotienting? (takes time)

- **Efficiency in TTS**
  - Regions
  - zones

- **Automata and safety properties**
The long distance road map

The local road map, and...

- **Verification of temporal properties**
  - LTL and CTL temporal/modal logic
  - The verification setting

- **CTL model checking**
  - Definition of CTL
  - Kripke structures
  - Definition of the modelling relation
  - Model checking algorithm for CTL

- **TCTL model checking**
  - Definition of TCTL
  - Model checking for TCTL
Mars pathfinder mission in 1997

Ran into serious problems:

The spacecraft began experiencing total system resets with loss of data each time ... due to priority inversion...
Priority inversion scenario

Three prioritized tasks

Higher priority task $\tau_1$ blocked by the much lower priority task that is holding a shared resource. The lower priority task $\tau_3$ has acquired this resource and then been preempted by the medium priority task $\tau_2$. In summary, $\tau_2$ is blocking $\tau_1$. 
On the pathfinder, the resource was a mutual exclusion semaphore

**How can we avoid priority inversion?**

- We could **disallow preemption** during the execution of a critical section, **but ...** this works only if critical sections are short, and might unnecessarily block higher priority processes that do not even use any shared resources.

- Or **use resource access protocols** such as
  - the priority **inheritance** protocol (PIP), or
  - the priority **ceiling** protocol (PCP).
Tasks have nominal and active priorities...

- A **nominal priority** is assigned by the scheduling algorithm (RMS or EDF or some other algorithm).
- The **active priority** is assigned by the protocol dynamically, specifically to avoid priority inversion.
- In **PIP**, we use scheduling based on **active priorities**:
  - When $\tau_i$ blocks higher-priority tasks, then its active priority is set to the **highest** of the priorities of the tasks it blocks.
  - When a task is blocked on a semaphore, it **transmits** its active priority to the job that holds the semaphore.
Two viewpoints...

- When $\tau_i$ blocks higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks. This prevents medium priority tasks from preempting $\tau_i$ and prolonging the blocking duration of the higher priority tasks.

- A task inherits the highest priority of the jobs blocked by it.

- When a task exits a critical section, it unlocks the semaphore; the job with the highest priority that is blocked on the semaphore, if any, is awakened.
At $t_1$, J2 does a `wait(b)`. Lock succeeds.

At $t_2$, J1 is scheduled and begins, as $P(J1) > P(J2)$.

At $t_3$, J1 does a `wait(a)`. Lock succeeds.

At $t_4$, J1 does a `wait(b)`, and from PIP, transmits its priority to J2. New priority is $P(J1)$. 
Avoiding priority inversion example...

\[\text{\(\tau_1\)}\]

\[\text{\(\tau_2\)}\]

\[\text{\(\tau_3\)}\]

1. \(\tau_1\) wants resource, blocks finally runs
2. \(\tau_2\) is blocking \(\tau_1\)
   - \(\tau_2\) has resource
   - \(\tau_2\) frees resource

3. \(\tau_2\) cannot run

Hugh Anderson
Verification of Real Time Systems - CS5270 4th lecture
Properties of PIP

Good and bad news

- The **good news** is that if there are a set of distinct semaphores that can block a task \( \tau \) then \( \tau \) can be blocked for **at most** the duration of at most one critical section, one for each of the semaphores. It can never be as long as the WCET of a lower priority task.

- The **bad news**:  
  - *chained blocking*, where a task \( \tau \) is blocked on critical sections held by lower priority jobs, and  
  - *deadlock*. 

Hugh Anderson  
Verification of Real Time Systems - CS5270 4th lecture
Chained blocking using PIP

In the worst case...

\[ \tau_1 \]

\[ \tau_2 \]

\[ \tau_3 \]

wants resource, blocks

has resource
Deadlock using PIP

Deadlock example: J1 is ABBA, J2 is BAAB

- At $t_1$, J2 does a `wait(b)`. Lock **succeeds**.
- At $t_2$, J1 is scheduled and begins, as $P(J1) > P(J2)$.
- At $t_3$, J1 does a `wait(a)`. Lock **succeeds**.
- At $t_4$, J1 does a `wait(b)`.
- At $t_5$, J2 does a `wait(a)`. **Deadlock**
Longest blocking time

Example task set

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

- Consider three periodic tasks \(\tau_1\), \(\tau_2\) and \(\tau_3\) (having decreasing priority) which share four resources \(A\), \(B\), \(C\) and \(D\) accessed using the priority inheritance protocol.
- The longest duration \(D_{iR}\) for the task \(\tau_i\) on resource \(R\) is given by the table. (\(D_{iR} = 0\) means \(\tau_i\) does not use at all the resource \(D\)).
Longest blocking time

Compute a (conservative) maximum blocking time for tasks:

\[ B_i = \min(B_i^\ell, B_i^s) \]

\[ B_i^\ell = \sum_{j=i+1}^{n} \max_k [D_{j,k} : C(S_k) \geq P_i] \]

\[ B_i^s = \sum_{k=1}^{m} j > i \max [D_{j,k} : C(S_k) \geq P_i] \]

where \( B_i^s \) is the sum of the durations of the longest critical sections guarded by semaphore \( S_k \) that can block \( \tau_i \), and \( B_i^\ell \) is the sum of the durations of the longest critical sections in tasks with lower priority than \( \tau_i \), guarded by semaphore \( S_k \), and that can block \( \tau_i \).
Longest blocking time

Computing the longest blocking time of example

- Use equation to derive the following \textit{blocking times}

\[
\begin{align*}
B_{1}^{l} &= 8 + 5 = 13 \\
B_{2}^{l} &= 5 \\
B_{3}^{l} &= 0 \\
B_{1}^{s} &= 4 + 1 + 6 + 8 = 19 \\
B_{2}^{s} &= 2 + 1 + 5 = 8 \\
B_{3}^{s} &= 0 \\
\end{align*}
\]

and so \( B_{1} = 13, \ B_{2} = 5 \) and \( B_{3} = 0 \).

This calculation is \underline{reasonably efficient}, but if you try to find a tighter bound, then the \underline{complexity} of the algorithm would be much higher (it is \underline{exponential}).
Priority ceiling protocol

An extension of the priority inheritance protocol...

- **Avoids chained blocking** and **deadlocks**.
- The **underlying idea** is that a task is not allowed to enter a critical section if there are already locked semaphores which could block it eventually (due to a sub-critical section nested within the entering critical section).
- Hence, once a task enters a critical section, it can not be blocked by lower priority tasks till its completion.
The protocol:

- Each semaphore $S$ is assigned a priority ceiling $C(S)$, the priority of the highest priority task that can lock $S$.
- $\tau$ is allowed to lock $S$ only if the priority of $\tau$ is strictly higher than the priority ceiling $C(S')$ of the semaphore $S'$ where $S'$ is the semaphore with the highest priority ceiling among all the semaphores which are currently locked by jobs other than $\tau$.
- When $\tau$ gets blocked by $S'$ then the priority of $\tau$ is transmitted to the job that currently holds $S'$. 
Priority ceiling protocol

The protocol:

- When $\tau'$ leaves a critical section guarded by $S'$ then it unlocks $S'$ and the highest priority job, if any, which is blocked by $S'$ is awakened.

- The priority of $\tau'$ is set to the highest priority of the job that is blocked by some semaphore that $\tau'$ is still holding.

- If none, the priority of $\tau'$ is set to be its nominal one.
ABBA and BAAB example OK with PCP:

- At $t_1$, J2 does a `wait(b)`. Lock **succeeds**.
- At $t_2$, J1 is scheduled and begins, as $P(J1) > P(J2)$.
- At $t_3$, J1 does a `wait(a)`. Due to PCP, J1 cannot lock a.
- At $t_6$, J2 releases $b$ and we are back to normal running...
PCP blocking time

A computation using the previous task set

- The maximum blocking time $B_i$ for each task if the resources are accessed using the Priority Ceiling Protocol is

$$B_i = \max_{j,k} \{ D_{j,k} \mid P_j < P_i, C(S_k) \geq P_i \}$$

and we have that

$$B_1 = \max(4, 2, 1, 6, 8, 5) = 8$$
$$B_2 = \max(2, 1, 5) = 5$$
$$B_3 = 0$$