Priority Inheritance on Condition Variables

Tommaso Cucinotta
Bell Laboratories, Alcatel-Lucent
Dublin, Ireland
Introduction

In computing systems, it is convenient to have tasks with different priorities

• to ensure low-latency and responsiveness
• I/O prioritized over computing
• Virtual Machines in cloud infrastructures run at different priority/urgency level
  – gold vs. bronze customers
Introduction

In computing systems, it is convenient to have **tasks with different priorities**
- to ensure low-latency and responsiveness
- I/O prioritized over computing
- Virtual Machines in cloud infrastructures run at different priority/urgency level
  - **gold** vs. **bronze** customers

**Interactions** among tasks often
- use shared data structures in memory
- **serialize access** through a **mutual exclusion semaphore** (mutex)
Introduction

In computing systems, it is convenient to have **tasks with different priorities**
- to ensure low-latency and responsiveness
- I/O prioritized over computing
- Virtual Machines in cloud infrastructures run at different priority/urgency level
  - **gold** vs. **bronze** customers

**Interactions** among tasks often
- use shared data structures in memory
- **serialize access** through a **mutual exclusion semaphore** (mutex)

Mixing these two paradigms leads to **undesirable situations**
Problem of Priority Inversion on Mutexes

Priority Inversion occurs when

- HP task A synchronizes with LP task C
- ... but a middle-priority task B defers execution of C, therefore A
Fixing Priority Inversion on Mutexes

Priority Inheritance avoids the problem
- mutex owner inherits highest priority among tasks waiting for mutex unlock (if higher than its own priority)
More Problems

More complex interactions require
• a condition variable (condvar)
  – over which tasks may suspend waiting for a condition, before the critical section

```
Task A
Lock(M);
// push in Q
Unlock(M);
Signal(CV);

Task C
Lock(M);
while (Q empty)
Wait(CV, M);
// pop from Q
Unlock(M);
```
Background Information

More **complex interactions** require
- **a condition variable** (condvar)
  - over which tasks may suspend waiting for a condition, before the critical section

Priority Inversion still occurs when
Related Work

Cornhill & Sha, '87 (International Workshop on Real-Time Ada Issues)
- Indefinite delay of HP tasks by LP tasks

Sha et al., '90
- Basic Priority Inheritance and Priority Ceiling Protocol

Later
- SRP, BWI, MBWI, FMLP, others...

Proposal: novel general solution
- for the problem of priority inversion
- in presence of arbitrary interactions among tasks
- based on mutexes and condition variables

Problem only marginally addressed in RT literature
- Limited to blocking RPC/RMI/Client-Server case (condition helper implicitly known)
Proposed Solution: PI-CV

Priority Inheritance on Condition Variable (PI-CV)
- declare which tasks may signal() on a condition variable (the helpers)
- helpers automatically inherit highest priority among wait()-ers
  - (if higher than their own priority)
- inheritance cancelled on signal()
- transitive behaviour
  - C inherits from B, which inherits from A
  - integrate with classical priority inheritance mutexes

How to realize it?
- add new syscall, e.g., to POSIX pthreads
  - pthread_cond_helpers_add(condvar, thread)
  - pthread_cond_helpers_del(condvar, thread)
- kernel modifications
  - (futex code on Linux)
Simulation

PI-CV implemented in RTSim
  • open-source simulator from SSSA
Simulated scenario
  • Client-server interactions (see table)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Client1</th>
<th>Client2</th>
<th>Client3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task period</td>
<td>676</td>
<td>683</td>
<td>687</td>
</tr>
<tr>
<td>Ovh lock/unlock</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ovh wait/signal</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Ovh push/pop</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Job comp</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Server call</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Exp duration</td>
<td></td>
<td></td>
<td>200000</td>
</tr>
</tbody>
</table>
Simulation Results

Results

- 41% reduction of WCET for HP task (as due to avoiding priority inversion)
Current Status & Future Work

Future Work

• **Real implementation** on Linux (half-way)
• **Use-case study with real application**
• **Schedulability analysis** (theoretical)

On-going collaboration with Scuola Superiore Sant’Anna and University of Trento
Thanks for your attention!

Questions?