Protocol-Transparent Resource Sharing in Hierarchically Scheduled Real-Time Systems
– Emerging Technologies and Factory Automation 2010 –

M.M.H.P. van den Heuvel - R.J. Bril - J.J. Lukkien

System Architecture and Networking (SAN)
Department of Mathematics and Computer Science
Eindhoven University of Technology
The Netherlands

13 September 2010
Outline

1. Introduction to Hierarchical Scheduling
2. RTOS and Experimental Setup
3. Inter-subsystem resource sharing
4. Conclusions
Hierarchical Scheduling: a solution for

- isolation: subsystems shall not interfere;
- development and analysis versus integration.
Hierarchical Scheduling: a solution for

- isolation: subsystems shall not *interfere*;
- development and analysis versus integration.

- subsystem: *server, set of tasks* and *local (task) scheduler*
- server: a budget allocated each *period*
Hierarchical Scheduling: a solution for

- isolation: subsystems shall not *interfere*;
- development and analysis versus integration.

- subsystem: *server*, *set of tasks* and *local (task) scheduler*
- server: a budget allocated each *period*

Tasks, located in arbitrary subsystems, may share logical resources
MicroC/OS-II Basics

MicroC/OS-II is
- a commercial RTOS
- targeted at embedded systems, e.g. in CE, military, aerospace and medical applications
- open source
- available at http://micrium.com/

It provides
- a portable and configurable kernel
- a fixed-priority, preemptive task scheduler
- basic services (mailboxes, mutexes and counting semaphores)
MicroC/OS-II Basics

MicroC/OS-II is

- a commercial RTOS
- targeted at embedded systems, e.g. in CE, military, aerospace and medical applications
- open source
- available at http://micrium.com/

It provides

- a portable and configurable kernel
- a fixed-priority, preemptive task scheduler
- basic services (mailboxes, mutexes and counting semaphores)

We created proprietary extensions for hierarchical scheduling of tasks.
Visualization of scheduling behavior:


MicroC/OS-II port to OpenRISC platform

OpenRISC: Architectural Simulator
http://opencores.org/openrisc, or1ksim
Global resource sharing problem

*Budget depletion* during a critical section can lead to excessive blocking times:

- SRP locally
- SRP globally
Global resource sharing problem

Three SRP-based solutions:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget
  1. **with payback**: the consumed overrun budget is subtracted from the next budget provisioning;
  2. **no payback**: no penalty for overrun consumption.
Global resource sharing problem

Three SRP-based solutions:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget
  1. **with payback**: the consumed overrun budget is subtracted from the next budget provisioning;
  2. **no payback**: no penalty for overrun consumption.

- **SIRAP and BROE**: Prevent budget depletion during resource access; i.e. before granting access, first check the remaining budget.
Global resource sharing problem

Three SRP-based solutions:

- **HSRP:** React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget
  1. **with payback:** the consumed overrun budget is subtracted from the next budget provisioning;
  2. **no payback:** no penalty for overrun consumption.

- **SIRAP and BROE:** Prevent budget depletion during resource access; i.e. before granting access, first check the remaining budget.

- **BROE . . .**
  - applies to EDF-scheduled systems
  - extends the constant-bandwidth server (CBS) with synchronization.
HSRP provides overrun budget (optionally a payback mechanism):

Legend:
- active
- holding mutex
SIRAP uses a skipping mechanism:

Legend:
- active
- holding mutex
### Implementation overheads and issues

<table>
<thead>
<tr>
<th>Event</th>
<th>HSRP</th>
<th>SIRAP</th>
<th>BROE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock resource</td>
<td>-</td>
<td>spinlock</td>
<td>postpone deadline replenish budget</td>
</tr>
<tr>
<td>Unlock resource</td>
<td>overrun completion</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Budget depletion</td>
<td>overrun</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Budget replenishment</td>
<td>overrun completion, payback (optionally)</td>
<td>spinlock-completion</td>
<td>-</td>
</tr>
</tbody>
</table>
Implementation overheads and issues (continued)

- **HSRP:**
  - close to default SRP;
  - expensive queue manipulations to track overrun budget;
  - complex implementation due to explicit event handling.

- **SIRAP:**
  - spinlocking is executed within a task's context, but wastes budget;
  - alternatively: suspend (i.e. block) and resume a task, but this is not SRP-compliant!

- **BROE:**
  - a task may block on its budget by postponing the deadline;
  - this happens before entering the critical section;

- **Protocol transparency:**
  - Save the maximum critical-section length of each subsystem.
Implementation overheads and issues (continued)

- **HSRP:**
  - close to default SRP;
  - expensive queue manipulations to track overrun budget;
  - complex implementation due to explicit event handling.

- **SIRAP:**
  - spinlocking is executed within a task’s context, but wastes budget;
  - alternatively: suspend (i.e. block) and resume a task, but this is not SRP-compliant!
Implementation overheads and issues (continued)

- **HSRP:**
  - close to default SRP;
  - expensive queue manipulations to track overrun budget;
  - complex implementation due to explicit event handling.

- **SIRAP:**
  - spinlocking is executed within a task’s context, but wastes budget;
  - alternatively: suspend (i.e. block) and resume a task, but this is not SRP-compliant!

- **BROE:**
  - a task may block on its budget by postponing the deadline;
  - this happens *before* entering the critical section;
HSRP:
- close to default SRP;
- expensive queue manipulations to track overrun budget;
- complex implementation due to explicit event handling.

SIRAP:
- spinlocking is executed within a task’s context, but wastes budget;
- alternatively: suspend (i.e. block) and resume a task, but this is not SRP-compliant!

BROE:
- a task may block on its budget by postponing the deadline;
- this happens before entering the critical section;

Protocol transparency:
- Save the maximum critical-section length of each subsystem.
Transparent interfaces for HSRP and SIRAP

Task1
Task2
Server-1-Idle
Task3
Task4
Server-2-Idle
OS-ServerIdle

Legend:
- active
- holding mutex

Martijn van den Heuvel (TU/e, SAN)
- Servers are sorted by monotonically increasing relative deadlines;
- The relative deadline determines the server’s *preemption level*. 
Outline

1. Introduction to Hierarchical Scheduling
2. RTOS and Experimental Setup
3. Inter-subsystem resource sharing
4. Conclusions
Conclusions

We presented:

- an efficient task-level SRP implementation;

- three alternative hierarchical SRP-implementations, i.e. SIRAP, HSRP and BROE;

- a side-by-side integration of these protocols in a single HSF;

- global protocol transparency.

Future work:

- a resource-efficient protocol-selection criterium;
- reduce the cost of transparent interfaces;
- wider range of budgeting/server models.
Conclusions

We presented:

- an efficient task-level SRP implementation;

- three alternative hierarchical SRP-implementations, i.e. SIRAP, HSRP and BROE;

- a side-by-side integration of these protocols in a single HSF;

- global protocol transparency.

Future work:

- a resource-efficient protocol-selection criterium;
- reduce the cost of transparent interfaces;
- wider range of budgeting/server models.

