Resource Sharing in Real-time Microkernels
– Training: real-time, embedded and concurrent programming –

Martijn van den Heuvel - Reinder J. Bril - Johan J. Lukkien

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

19th July 2010
Outline

1. Introduction
2. Task Synchronization
3. Stack Resource Policy in microC/OS-II
4. Conclusions
An embedded system typically has:
- a simple microcontroller (e.g. TI MSP-430x1611);
- very limited resources, especially RAM ($\sim 10$ kB);
- peripherals to be controlled.
Introduction to Microkernels

An embedded system typically has:
- a simple microcontroller (e.g. TI MSP-430x1611);
- very limited resources, especially RAM (∼10 kB);
- peripherals to be controlled.

A RTOS provides:
- abstractions (e.g. tasks and synchronization primitives);
- interaction between hardware and application;
- simplified debugging;
- I/O libraries (support for platform hardware).
Introduction to Microkernels

An embedded system typically has:
- a simple microcontroller (e.g. TI MSP-430x1611);
- very limited resources, especially RAM (∼ 10 kB);
- peripherals to be controlled.

A RTOS provides:
- abstractions (e.g. tasks and synchronization primitives);
- interaction between hardware and application;
- simplified debugging;
- I/O libraries (support for platform hardware).

Microkernel objective

Provide a reduced, configurable system that fits on a typical system-on-chip.
microC/OS-II Basics

microC/OS-II is
- a commercial RTOS;
- targeted at embedded systems;
- open source;

It provides
- a portable and configurable kernel;
- a fixed-priority, preemptive task scheduler;
- basic synchronization services.
The scheduler selects the **highest-priority, ready task.**
At each time instant a task must have a unique priority.
1 Introduction

2 Task Synchronization

3 Stack Resource Policy in microC/OS-II

4 Conclusions
MicroC/OS-II: Basic Synchronization Services

- Disable and enable interrupts
  - share variables or data structures with an ISR.

Software events:
- Mailboxes: communicate messages between tasks; FIFO or Priority-based.
- Flags: synchronize the occurrence of multiple events.
- (Counting) semaphores: e.g., signaling of tasks.
- Mutexes: resolve priority inversion.

Blocking for an event means keeping track of waiting tasks!
Disable and enable interrupts
- share variables or data structures with an ISR.

**Software events:**

- **Mailboxes**
  - communicate messages between tasks;
  - FIFO or Priority-based.

- **Flags**
  - synchronize the occurrence of multiple events.

- **(Counting) semaphores**
  - e.g. signaling of tasks

- **Mutexes**
  - resolve priority inversion
Disable and enable interrupts
  share variables or data structures with an ISR.

Software events:
  Mailboxes
    communicate messages between tasks;
    FIFO or Priority-based.
  Flags
    synchronize the occurrence of multiple events.
  (Counting) semaphores
    e.g. signaling of tasks
  Mutexes
    resolve priority inversion

Blocking for an event means keeping track of waiting tasks!
The easiest and fastest way to gain exclusive access to a shared resource
The easiest and fastest way to gain exclusive access to a shared resource

Be careful, because:

- disabling interrupts for too long affects the response to interrupts (i.e. interrupt latency);
- a non-preemptive system becomes easily not schedulable.
The easiest and fastest way to gain exclusive access to a shared resource

Be careful, because:
- disabling interrupts for too long affects the response to interrupts (i.e. interrupt latency);
- a non-preemptive system becomes easily not schedulable.

However,
- it is the only way that a task can share variables or data structures with an ISR;
The easiest and fastest way to gain exclusive access to a shared resource

Be careful, because:
- disabling interrupts for too long affects the response to interrupts (i.e. interrupt latency);
- a non-preemptive system becomes easily not schedulable.

However,
- it is the only way that a task can share variables or data structures with an ISR;

Keep interrupts disabled for as little time as possible!
The price of increased preemptiveness:

What really happened on Mars Rover Pathfinder?
Mutual Exclusion: Priority Inheritance Protocol

A solution:

[Diagram showing the Priority Inheritance Protocol with tasks and priorities over time, illustrating how the protocol resolves resource issues without causing deadlocks or chained-blocking.]
Mutual Exclusion: Priority Inheritance Protocol

A solution:

Multi-resource issues:
- deadlocks;
- chained-blocking.
Priority calling is similar to *priority inheritance protocol* (PIP):

**Priority Inheritance Rule:**

when a higher-priority task blocks on a resource, the lower-priority task holding the resource inherits the higher priority;
Priority calling is similar to *priority inheritance protocol* (PIP):

**Priority Inheritance Rule:**
when a higher-priority task blocks on a resource,  
the lower-priority task holding the resource inherits the higher priority;

<table>
<thead>
<tr>
<th>Priority Calling</th>
<th>Priority Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique priority for each resource</td>
<td>run-time priority adaption</td>
</tr>
<tr>
<td>inherits a predefined priority</td>
<td>transitive priority inheritance</td>
</tr>
<tr>
<td>non-transparent</td>
<td>transparent</td>
</tr>
</tbody>
</table>
Priority calling is similar to *priority inheritance protocol* (PIP):

**Priority Inheritance Rule:**
when a higher-priority task blocks on a resource, the lower-priority task holding the resource inherits the higher priority;

<table>
<thead>
<tr>
<th>Priority Calling</th>
<th>Priority Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique priority for each resource</td>
<td>run-time priority adaption</td>
</tr>
<tr>
<td>inherits a predefined priority</td>
<td>transitive priority inheritance</td>
</tr>
<tr>
<td>non-transparent</td>
<td>transparent</td>
</tr>
</tbody>
</table>

**It is not:** Highest Locker Protocol (HLP) or Stack Resource Policy (SRP).

- microC/OS-II: a task inherits a higher priority *only* when a higher priority task is blocked;
- in HLP/SRP a task immediately inherits a priority *when it locks a resource.*
microC/OS-II: protocol classification

microC/OS-II’s synchronization protocol suffers from deadlock:

Legend:
- active
- holding
- mutex

Conclusion:
microC/OS-II implements a non-transparent, priority-inheritance protocol.
**microC/OS-II: protocol classification**

microC/OS-II’s synchronization protocol suffers from deadlock:

![Graph showing task scheduling and mutex usage](image)

Legend:
- active
- holding
- mutex

**Conclusion:**

microC/OS-II implements a *non-transparent, priority-inheritance protocol*
Outline

1. Introduction
2. Task Synchronization
3. Stack Resource Policy in microC/OS-II
4. Conclusions
Each resource has a statically determined *resource ceiling*:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.
Intermezzo: Stack Resource Policy (SRP)

- Each resource has a statically determined resource ceiling:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.

- A dynamically updated system ceiling is maintained:

**Definition of the system ceiling**
the maximum resource ceiling of any resource currently being locked in the system.
Intermezzo: Stack Resource Policy (SRP)

- Each resource has a statically determined *resource ceiling*:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.

- A dynamically updated *system ceiling* is maintained:

**Definition of the system ceiling**
the maximum resource ceiling of any resource currently being locked in the system.

- A task can only be selected for execution if
  1. it has the highest priority among all ready tasks;
  2. its priority is higher than the current system ceiling.
SRP provides **non-blocking** primitives:
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource (**no wait queues**).
SRP provides non-blocking primitives:
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource (no wait queues).

It therefore allows tasks to share their execution stack;

Sharing of the execution stack is possible, if two tasks execute non-interleaved.
SRP provides **non-blocking** primitives:
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource (no wait queues).
- it therefore allows tasks to share their execution stack;

Sharing of the execution stack is possible, if two tasks execute non-interleaved.

Why does stack-sharing not work in microC/OS-II?
SRP is non-transparent:

- similar to microC/OS-II’s PIP-like implementation.
- each resource has a statically computed ceiling
  (represents the highest priority task that uses the resources).
SRP is non-transparent:
- similar to microC/OS-II’s PIP-like implementation.
- each resource has a statically computed ceiling
  (represents the highest priority task that uses the resources).

Maintaining the system ceiling can be implemented using a stack data structure:
- we stack the resource ceilings of used resources in a monotonically increasing order;
- the top of the stack represents the system ceiling.
Deadlocks are resolved when critical sections are properly nested:
Deadlocks are resolved when critical sections are properly nested:

- Easy implementations (approx. 170 lines of code);
- Extended microC/OS-II scheduler with SRP’s preemption rule.
Alternative SRP implementation

Assuming fixed-priority scheduling:

Immediate Priority Ceiling exhibits the same execution behavior as SRP

Instead of changing the scheduler:

- **Lock**: raise a task’s priority equal to the resource ceiling;
- **Unlock**: restore a task’s priority.

---

**Note 1**: Immediate Priority Ceiling == Highest Locker Protocol

**Note 2**: SRP/IPC/HLP do not need waiting lists of blocked tasks!

---

Martijn van den Heuvel (TU/e, SAN) Resource Sharing in Real-time Microkernels 19th July 2010 19 / 24
Alternative SRP implementation

Assuming fixed-priority scheduling:
Immediate Priority Ceiling exhibits the same execution behavior as SRP

Instead of changing the scheduler:
- Lock: raise a task’s priority equal to the resource ceiling;
- Unlock: restore a task’s priority.

Note 1: Immediate Priority Ceiling == Highest Locker Protocol

Note 2: SRP/IPC/HLP do not need waiting lists of blocked tasks!
Alternative SRP implementation

Assuming fixed-priority scheduling:
Immediate Priority Ceiling exhibits the same execution behavior as SRP

Instead of changing the scheduler:
- Lock: raise a task’s priority equal to the resource ceiling;
- Unlock: restore a task’s priority.

Note 1: Immediate Priority Ceiling == Highest Locker Protocol

Note 2: SRP/IPC/HLP do not need waiting lists of blocked tasks!

Advantages or disadvantages?
Outline

1. Introduction
2. Task Synchronization
3. Stack Resource Policy in microC/OS-II
4. Conclusions
Conclusions

We presented:

- an overview of microC/OS-II’s task-communication methods;

- a classification of microC/OS-II’s priority-inversion protocol;

- an efficient task-level SRP implementation;
Conclusions

We presented:

- an overview of microC/OS-II’s task-communication methods;
- a classification of microC/OS-II’s priority-inversion protocol;
- an efficient task-level SRP implementation;

Keep in mind:

- Make a minimal number of modifications to MicroC/OS-II;
- Keep durations of disabled interrupts short and predictable;
- Be aware of the consequences of blocking.
References

T. Baker.
Stack-based scheduling of realtime processes.

G. Buttazzo.
Hard real-time computing systems - predictable scheduling algorithms and applications (2nd edition).

L. Sha, R. Rajkumar, and J. Lehoczky.
Priority inheritance protocols: an approach to real-time synchronisation.
Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip.  

J. J. Labrosse.  
Microc/OS-II.  

M. M. H. P. van den Heuvel, R. J. Bril, J. J. Lukkien, and M. Behnam.  
Extending a HSF-enabled open-source real-time operating system with resource sharing.  
Questions

1. What are the disadvantages of SRP compared to PIP?

2. What are the (dis)advantages of both SRP implementation methods?

3. Implement SRP in microC/OS-II.

4. Make 3 test-cases and exchange the test-cases with your neighbor.

5. Make a test application with at least two tasks that need mutual exclusive access to the UART device, i.e. two concurrent tasks try to print a string to the serial port.