Sloth: Let the Hardware Do the Work!*

Wanja Hofer, Daniel Lohmann, Fabian Scheler, Wolfgang Schröder-Preikschat
Friedrich–Alexander University Erlangen–Nuremberg
{wanja,lohmann,scheler,wosch}@cs.fau.de

* This work was partly supported by the German Research Council (DFG) under grants no. SCHR 603/4 and SCHR 603/7-1. Wanja Hofer was supported by the German Academic Exchange Service (DAAD) under grant no. D/09/40595.

1. MOTIVATION

Traditional priority-driven real-time kernels distinguish between tasks, which are scheduled and dispatched by a software scheduler, and (possibly different kinds of) interrupt service routines (ISRs), which are scheduled and dispatched by the hardware platform. In our Sloth[1] system, we get rid of the software scheduler and let the hardware do the scheduling and dispatching work for tasks, too. We do this by designing and implementing every task as an interrupt handler on the hardware platform.

2. PLATFORM-AWARE DESIGN

Our Sloth kernel targets embedded OS kernels of the OSEK class, which is an OS standard whose implementations are widely deployed in the automotive industry. The main idea behind Sloth is to make selected use of platform properties in the design without being fully platform-dependent. The figure shows an overview of our Sloth design and how common OS abstractions are mapped to it. For scheduling, each task is assigned an IRQ source—a platform-specific IRQ abstraction—during the system configuration.

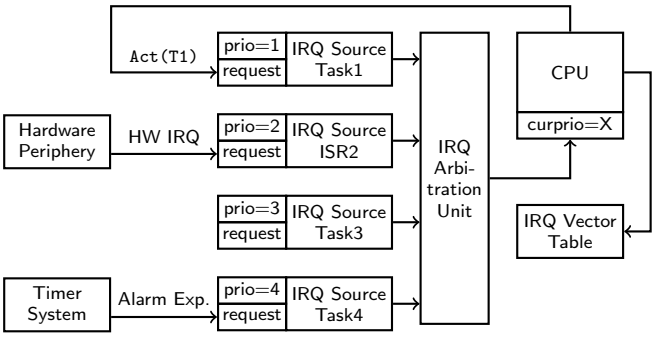
Synchronous task activations (e.g., Task 1) performed by tasks or ISRs are mapped to an interrupt request to the corresponding IRQ source. On many platforms, this corresponds to a single memory-mapped register write instruction. If the newly activated task has a higher priority than the currently running one, the hardware will immediately service the request; otherwise, it will delay it until the CPU priority has dropped.

Accessing shared resources with a stack-based priority ceiling protocol to avoid priority inversion is performed by raising and lowering the CPU priority accordingly. By raising the priority to the maximum of all tasks that can potentially lock the mutex, dispatching of those tasks is delayed until after the mutex is unlocked (and the priority is lowered).

Alarm-activated tasks (e.g., Task 4) are assigned a timer-connected IRQ source during the configuration. This way, when a timer expires, the corresponding task is automatically scheduled by the hardware and dispatched as soon as the CPU priority is low enough.

"Real" ISRs (e.g., ISR 2)—that is, those that are asynchronously triggered by hardware periphery—are transparently integrated into the system's priority space, scheduled and delayed like regular tasks.

[1]The name honors both the lazy animal breed and the deadly sin.



3. PRELIMINARY RESULTS

In order to evaluate its properties, we have implemented Sloth on the Infineon-TriCore platform (commodity hardware used in the automotive industry). Our new design has several promising implications for the implemented Sloth system:

- The Sloth implementation is very concise (< 200 SLOC), being a very easy subject to verification.
- Sloth's memory footprint is about 700 bytes small, depending on and scaling with the number of OS abstractions used.
- Sloth's execution times in system-service microbenchmarks are between 5% and 60% than those of a commercial competitive, but software-based OSEK implementation. Sloth's particular strengths are situations where rescheduling and dispatching are necessary.
- Since Sloth has a unified control-flow design, the system synchronization is very easy and uniform (altering the CPU priority); furthermore, it is not susceptible to the real-time problem of rate-monotonic priority inversion (high-priority tasks being interrupted by low-priority ISRs).

4. FUTURE WORK

Our current design does not support blocking tasks with state (stack) of their own. Nevertheless, Sloth is suitable for the implementation of a broad class of real-time systems, including RMA and DMA systems. Still, we want to investigate how blocking functionality fits into our design and how it should be implemented. Furthermore, we want to implement Sloth on other platforms like the ARM Cortex-M3 and Intel x86 with the APIC to further evaluate its applicablity.