CPU Scheduler implementation hints for architecture specific code

Nick Piggin, 2005

Context switch

1. Runqueue locking

By default, the switch_to arch function is called with the runqueue locked. This is usually not a problem unless switch_to may need to take the runqueue lock. This is usually due to a wake up operation in the context switch. See arch/ia64/include/asm/system.h for an example.

To request the scheduler call switch_to with the runqueue unlocked, you must `#define __ARCH_WANT_UNLOCKED_CTXSW` in a header file (typically the one where switch_to is defined).

Unlocked context switches introduce only a very minor performance penalty to the core scheduler implementation in the CONFIG_SMP case.

2. Interrupt status

By default, the switch_to arch function is called with interrupts disabled. Interrupts may be enabled over the call if it is likely to introduce a significant interrupt latency by adding the line `#define __ARCH_WANT_INTERRUPTS_ON_CTXSW` in the same place as for unlocked context switches. This define also implies `__ARCH_WANT_UNLOCKED_CTXSW`. See arch/arm/include/asm/system.h for an example.

CPU idle

Your cpu_idle routines need to obey the following rules:

1. Preempt should now disabled over idle routines. Should only be enabled to call schedule() then disabled again.

2. need_resched/TIF_NEED_RESCHED is only ever set, and will never be cleared until the running task has called schedule(). Idle threads need only ever query need_resched, and may never set or clear it.

3. When cpu_idle finds (need_resched() == 'true'), it should call schedule(). It should not call schedule() otherwise.

4. The only time interrupts need to be disabled when checking need_resched is if we are about to sleep the processor until the next interrupt (this doesn't provide any protection of need_resched, it prevents losing an interrupt).

   4a. Common problem with this type of sleep appears to be:

```
local_irq_disable();
if (!need_resched()) {
        local_irq_enable();
        *** resched interrupt arrives here ***
        __asm__("sleep until next interrupt");
}
```

5. TIF_POLLING_NRFLAG can be set by idle routines that do not
   need an interrupt to wake them up when need_resched goes high.
   In other words, they must be periodically polling need_resched,
   although it may be reasonable to do some background work or enter
   a low CPU priority.

        5a. If TIF_POLLING_NRFLAG is set, and we do decide to enter
            an interrupt sleep, it needs to be cleared then a memory
            barrier issued (followed by a test of need_resched with
            interrupts disabled, as explained in 3).

arch/i386/kernel/process.c has examples of both polling and
sleeping idle functions.

Possible arch/ problems
=======================

Possible arch problems I found (and either tried to fix or didn't):

h8300 - Is such sleeping racy vs interrupts? (See #4a).
        The H8/300 manual I found indicates yes, however disabling IRQs
        over the sleep mean only NMIs can wake it up, so can't fix easily
        without doing spin waiting.

ia64 - is safe_halt call racy vs interrupts? (does it sleep?) (See #4a)

sh64 - Is sleeping racy vs interrupts? (See #4a)

sparc - IRQs on at this point(?), change local_irq_save to _disable.
      - TODO: needs secondary CPUs to disable preempt (See #1)