Disabling interrupts for synchornization in kernel code

Why do you need to disable interrupts when synchornizing code in kernel?

For example, in the following code taken from the linux schedule() function:

```
need_resched:
prev = current;
rq = this_rq();

**spin_lock_irq(&rq->lock);** //disables interrupts

switch (prev->state) {
case TASK_INTERRUPTIBLE:
    if (unlikely(signal_pending(prev))) {
        prev->state = TASK_RUNNING;
        break;
    }
default:
    deactivate_task(prev, rq);
case TASK_RUNNING:
    ;
}
```

The spin_lock_irq() function disables interrupts, but why is it needed? Suppose I don't disable interrupts, and an interrupt comes, so I'll just handle it and come back to the scheduler and resume what I was doing.

linux operating-system synchronization interrupt

asked Jun 28 '14 at 11:33

Dan Dv
93 1 7

1 Err - what if the interrupt requests a scheduler run on exit? – Martin James Jun 28 '14 at 12:03

@Martin What kind of interrupt calls the scheduler? – Dan Dv Jun 28 '14 at 12:27

Any interrupt that needs to make a thread ready immediately upon exit, eg. a NIC interrupt that reads data into a buffer that a thread is waiting on with a read() call - the thread must be made ready/running immediately the interrupt exits so that it can run on and process the data. – Martin James Jun 28 '14 at 12:36

1 Answer

It is necessary because the scheduler can be entered via both software and hardware interrupts.

Software interrupts, eg. sleep() calls and inter-thread comms calls, (eg semaphore, condvar or event signal), may change the set of running threads and so will request a scheduler run. These calls have thread/process context and happen whenever they call into the kernel.

Hardware interrupts, eg. KB, mouse, disk, NIC cause drivers to run and the driver may well wish to change the set of running threads by running the scheduler, eg. to make a thread ready that was blocked waiting for a disk read. Hardware interrupts have no thread/process context and can happen at any time while interrupts are enabled.

There are sections of scheduler code/data that are not reentrant. If interrupts are not briefly disabled for those sections, chaos will ensue when the scheduler is interrupted by hardware and reentered from a driver.

answered Jun 30 '14 at 9:02

Martin James
16.2k 3 14 33