Is spinlock required for every interrupt handler?

In Chapter 5 of ULK the author states as follows:

> "...each interrupt handler is serialized with respect to itself-that is, it cannot execute more than one concurrently. Thus, accessing the data struct does not require synchronization primitives"

I don't quite understand why interrupt handlers is "serialized" on modern CPUs with multiple cores. I'm thinking it could be possible that a same ISR can be run on different cores simultaneously, right? If that's the case, if you don't use spinlock to protect your data it can come to a race condition.

So my question is, on a modern system with multi-cpus, for every interrupt handler you are going to write that will read & write some data, is spinlock always needed?

c linux linux-kernel

asked Aug 24 '13 at 17:55

Jun
151 9

3 Answers

While executing interrupt handlers, the kernel explicitly disables that particular interrupt line at the interrupt controller, so one interrupt handler cannot be executed more than once concurrently. (The handlers of *other* interrupts can run concurrently, though.)

edited Aug 26 '13 at 7:41

answered Aug 24 '13 at 22:16

CL.
58.2k 5 25 54

I thought, it disables the interrupt only for the CPU that executes that handler but not for other CPUs. Is that right? – Eugene Aug 26 '13 at 6:51

1 It disables the interrupt line itself, at the interrupt controller. Whether *other* interrupts on the same CPU are disabled or not is architecture specific. – CL. Aug 26 '13 at 7:34

Clarification: as per CL. remark below - the kernel makes sure not to fire the interrupt handler for the *same interrupt* but if you have multiple registrations of the same interrupt handler for *multiple interrupts* than the below answer is, I believe, correct.

You are right that the same interrupt handler can run concurrently on multiple cores and that shared data needs to be protected. However, a spinlock is not the only and certainly not always the recommended way to achieve this.

A multitude of other synchronization methods, from per-CPU data, accessing shared data only using atomic operations and even Read-Copy-Update variants may be used to protect the shared data.

edited Aug 26 '13 at 15:49

answered Aug 25 '13 at 14:55


gby
8,494 12 35

2 This is wrong; the kernel explicitly prevents the same interrupt handler from running concurrently on multiple cores. When data is accessed *only* from interrupt handlers, locking is required only if different handlers share the same data. ULK is correct; and LDD is correct ("you will never see two processors handling the same IRQ at the same time.", chapter 10). – CL. Aug 26 '13 at 7:45

@CL. I probably parsed the question wrong, but I thought he was asking about the same interrupt *handler* on different cores, not for the same interrupt. I understand the kernel blocks the interrupt line, but AFAIK understand there is nothing preventing the same interrupt handler being registered multiple times, if it is registered to handle a different interrupt. – gby Aug 26 '13 at 15:47

@gby got your point. Thanks! – Jun Aug 26 '13 at 18:30

If the critical data is shared b/w the interrupt handler and your process (may be a kernel thread) then you need to protect your data and hence spinlock is required.A common Kernel api for spinlock is : spin_lock(). There are also variants of these api e.g. spin_lock_irqsave() which can help avoiding the deadlock problems which one can face while acquiring/holding the spin locks.Please go through the below link to find details of the subject:
http://www.linuxjournal.com/article/5833

answered Aug 25 '13 at 9:39


a.saurabh
190 1 7

-1: Either explain why `spin_lock_irqsave` is needed here, or drop it. It does not answer the question anyway. – Jan Hudec Aug 26 '13 at 7:57