Why can't you sleep while holding spinlock?

In the linux kernel, why can't you sleep while holding a spinlock?

linux kernel

edited Oct 24 '11 at 1:39
caf
122k 8 139 269

asked Jan 20 '11 at 20:10
Bandicoot
954 1 14 26

2 Oversimplified: because you locked out everybody else with the spinlock, so they can't schedule but anything to your holding thread. – user562374 Jan 20 '11 at 21:18

6 Answers

Example: your driver is executing and has just taken out a lock that controls access to its device. While the lock is held, the device issues an interrupt, which causes your interrupt handler to run. The interrupt handler, before accessing the device, must also obtain the lock. Taking out a spinlock in an interrupt handler is a legitimate thing to do; that is one of the reasons that spinlock operations do not sleep. But what happens if the interrupt routine executes in the same processor as the code that took out the lock originally? While the interrupt handler is spinning, the noninterrupt code will not be able to run to release the lock. That processor will spin forever.

Source: http://www.makelinux.net/ldd3/chp-5-sect-5.shtml

answered Jan 20 '11 at 20:13

William Tate
18k 5 47 57

1 Evidently, this situation can be avoided by disabling interrupts in critical sections that are protected using spinlocks. For example, Oberon takes this approach (Pg. 57). e-collection.library.ethz.ch/eserv/eth:26082/eth-26082-02.pdf – gjain Aug 8 '13 at 23:05

It's not that you *can't* sleep while holding a spin lock. It is a **very very bad** idea to do that. Quoting LDD:

> Therefore, the core rule that applies to spinlocks is that any code must, while holding a spinlock, be atomic. It cannot sleep; in fact, it cannot relinquish the processor for any reason except to service interrupts (and sometimes not even then).

Any deadlock like mentioned above may result in an unrecoverable state. Another thing that could happen is that the spinlock gets locked on one CPU, and then when the thread sleeps, it wakes up on the other CPU, resulting in a kernel panic.

Answering Bandicoot's comment, in a spin lock context, pre-emption is disabled *only in case of a uniprocessor pre-emptible kernel* because disabling pre-emption effectively prevents races.

> If the kernel is compiled without CONFIG_SMP, but CONFIG_PREEMPT is set, then spinlocks simply disable preemption, which is sufficient to prevent any races. For most purposes, we can think of preemption as equivalent to SMP, and not worry about it separately.

http://www.kernel.org/pub/linux/kernel/people/rusty/kernel-locking/index.html

answered May 24 '12 at 6:40

Omair
301 3 13

Apart from what willtate has mentioned, assume that a process sleeps while holding a spilock. If the new process that is scheduled tries to acquire the same spinlock, it starts spinning for the lock to be available. Since the new process keeps spinning, it is not possible to schedule the first process and thus the lock is never released making the second process to spin for ever and we have a deadlock.

edited Apr 8 '13 at 9:58

answered Feb 19 '12 at 15:59

San
31 5

I disagree with William's response (his example). He's mixing two different concepts: preemption and synchronization.

An Interrupt Context could preempt a Process Context and thus if there a RESOURCE shared by the both, we need to use

```
spin_lock_irqsave()
```

to (1) disable the IRQ (2) acquire the lock. By step 1, we could disable interrupt preemption.

I think this thread is much convincing. Sleep() means a thread/process yields the control of the CPU and CONTEXT SWITCH to another, without releasing the spinlock, that's why it's wrong.

answered Jul 1 '13 at 9:57

Qylin
169 1 10

> Agree, if I am not wrong William's issue is solved by using the appropriate spin_lock function to disable interrupt preemption. The sleep problem is different, you must just avoid sleeping in the spinlock critical section, otherwise you will get bad performance or a deadlock. – Jorge González Lorenzo Jan 7 '14 at 15:46

Another likely explanation is that, in a spinlock context pre-emption is disabled.

answered Jan 20 '11 at 22:11

TheLoneJoker
546 6 19

> Why is it disabled ? – Bandicoot Feb 6 '11 at 23:42

The key point is in Linux kernel, acquiring a spin lock will disable preemption. Thus sleeping while holding a spin lock could potentially cause deadlock.

For example, thread A acquires a spin lock. Thread A will not be preempted until it releases the lock. As long as thread A quickly does its job and releases the lock, there is no problem. But if thread A sleeps while holding the lock, thread B could be scheduled to run. And thread B could acquire the same lock as well. Thread B also disables preemption and tries to acquire the lock. And a deadlock occurs. Thread B will never get the lock since thread A holds it, and thread A will never get to run since thread B disables preemption.

And why disabling preemption in the first place? I guess it's because we don't want threads on other processors to wait too long.

answered Dec 23 '14 at 11:26

Nan Wang

1