# Why spinlocks don't work in uniprocessor (unicore) systems?



I know that spinlocks work with spining, different kernel paths exist and Kernels are preemptive, so why spinlocks don't work in uniprocessor systems? (for example, in Linux)

linux-kernel spinlock

asked Feb 6 '12 at 20:26

user683595
**109** 1 7

2 possible duplicate of Is spin lock useful in a single processor uni core architecture? – Joe Feb 6 '12 at 20:33

It's not that spinlocks don't work, it's that they're highly wasteful of cpu cycles. – Marc B Feb 6 '12 at 20:37

## 4 Answers

If I understand your question, you're asking why spin locks are a bad idea on single core machines.

They should still *work*, but can be much more expensive than true thread-sleeping concurrency:

When you use a spinlock, you're essentially asserting that you don't think you will have to wait long. You are saying that you think it's better to maintain the processor time slice with a busy loop than the cost of sleeping your thread and context-shifting to another thread or process. If you have to wait a very short amount of time, you can sleep and be reawakened almost immediately, but the cost of going down and up is more expensive than just waiting around.

This is more likely to be OK on multi-core processors, since they have much better concurrency profiles than single core processors. On multi core processors, between loop iterations, some other thread may have taken care of your prerequisite. On single core processors, it's not possible that someone else could have helped you out - you've locked up the one and only core.

The problem here is that **if you wait or sleep on a lock, you hint to the system that you don't have everything you need yet, so it should go do some other stuff and come back to you later**. With a spin lock, you *never* tell the system this, so you lock it up waiting for something else to happen - but, meanwhile, you're **holding up the whole system**, so something else *can't* happen.

edited Aug 6 '13 at 17:14

kumar
**549** 8 35

answered Feb 6 '12 at 20:40

Matt
**7,216** 1 16 28



The nature of a spinlock is that it does not deschedule the process - instead it spins until the process acquires the lock.

On a uniprocessor, it will either immediately acquire the lock or it will spin forever - if the lock is contended, then there will never be an opportunity for the process which currently holds the resource to give it up. Spinlocks are only useful when another process can execute while one is spinning on the lock - which means multiprocessor systems.

answered Feb 10 '12 at 5:07

caf
**122k** 8 139 269

Caf, I have a bit of confusion here. << On a uniprocessor,it will either immediately acquire the lock or it will spin forever. >> You have only one processor , when that one processor tries to hold the spin lock it will immediately acquire it. As you have taken spin lock, descheduling is not possible. So NO OTHER PROCESS will be able to execute now. But, you have mentioned contention is possible? When you say contention here, you mean contention between Proces context and interrupt context? Once spin lock is acquired, NO OTHER PROCESS will be able to execute? – kumar Aug 6 '13 at 17:02

1 @kumar: That's right - if a process holding a spinlock is interrupted and the interrupt tries to take the same lock. – caf Aug 6 '13 at 22:44

Spinlocks are, by their nature, intended for use on multiprocessor systems, although a uniprocessor workstation running a preemptive kernel behaves like SMP, as far as concurrency is concerned. If a nonpreemptive uniprocessor system ever went into a spin on a lock, it would spin forever; no other thread would ever be able to obtain the CPU to release the lock. For this reason, spinlock operations on uniprocessor systems without preemption enabled are optimized to do nothing, with the exception of the ones that change the IRQ masking status. Because of preemption, even if you never expect your code to run on an SMP system, you still need to implement proper locking.

Ref:***Linux device drivers*** By Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartma

answered Feb 29 '12 at 15:23

laksbv
**87** 2

there are different versions of spinlock: spin_lock_irqsave(&xxx_lock, flags); ... critical section here .. spin_unlock_irqrestore(&xxx_lock, flags); In Uni processor spin_lock_irqsave() should be used when data needs to shared between process context and interrupt context, as in this case IRQ also gets disables. spin_lock_irqsave()work under all circumstances, but partly *because* they are safe they are also fairly slow. However, in case data needs to be protected across different CPUs then it is better to use below versions, these are cheaper ones as IRQs dont get disabled in this case: spin_lock(&lock); ... spin_unlock(&lock);

In uniprocessor systems calling spin_lock_irqsave(&xxx_lock, flags); has the same effect as disabling interrupts which will provide the needed interrupt concurrency protection without unneeded SMP protection. However, in multiprocessor systems this covers both interrupt and SMP concurrency issues.

answered Jul 11 '14 at 8:31

inderjeet
**11** 2