spin_lock implementation in ARM linux

spin_lock is one of the most popular synchronization tools used inside the Linux kernel. There is a lot of information available on what is a spin_lock and when and why to use spin_lock. In this post I just want to cover what goes inside the implementation of a spin_lock with ARM processors as a reference.

spin_locks are usually implemented with some support from hardware. It is not a purely software concept in the implementation. It is usually a assembly code. ARM processor provides two special instructions which are primarily for implementation of spin_locks. These instructions are LDREX and STREX. The older versions of the ARM processor had the SWP instruction to implement a spin_lock. The x86 architecture also provides the cmpxchg instruction which is used for this.

Before understanding the implementation details, let's first understand a few basic instructions that are used in the implementation.

IMPORTANT CONCEPTS THAT ARE NEEDED TO UNDERSTAND SPIN_LOCK BETTER:

wfe: Wait For Event. This is an ARM instruction which puts the ARM core into a lower power state. The core logic is off but the wake-up mechanism and the RAM arrays are kept on. This is usally the first thing that the cpu does when it goes into idle. The wake-up from this very very fast, hence it is used in spinlocks. The wake is from a external event from another processor or interrupt. If you notice carefully we are technically not spinning during a spinlock ;-). Spinning is such a waste of power. But, we are not getting *scheduled out* as well. So, from the process point of view it is still holding the CPU.

sev: Send Event. This to send an event to another Processor in the system to wake it up from WFE state to start executing code.

Barriers: All latest processors use an Out-Of-Order execution pipeline. What this simply means is that if current instruction has completed execution, it does not mean that the previous instruction has completed execution. As, the order in which the instructions are processed depends on the several factors like execution unit availability etc.. This is done for performance gain. When implementing spin_locks() one needs to ensure this does not come in our way. We need to ensure that the previous lock holder is completely out of the critical section. So, we use these barrier instructions, which are again processor specific instructions which ensures that all the instructions before the barrier is completed.

ldrex, strex:
These instructions help in automatically updated the memory from 0 to 1. The older instructions like SWP was a single instruction with tries to change the memory in one go automatically In the newer ARM architectures this is split into two instructions and some logic can be applied in between these instructions. These instructions have been explained in detail in the ARM manuals.

Now, lets have a overview of the implementation itself.

In Linux, the ARM implementation can be seen arch/arm/include/asm/spinlock.h

Once the above concepts are understood the code below becomes very much self-explanatory. The code for implementing this keeps constantly improving, but the fundamental concept behind the implementation does not change drastically.

1: ldrex <<< load the value exclusively >>>
teq <<< test if equal to zero >>>
wfe <<< wait for event if not zero. which means someone is holding it >>>
strexeq <<< no one is holding the lock try to store 1. >>>
teq <<< check if stored correctly >>>
bne 1b <<< if not stored correctly go and and try again >>>

Posted 16th April 2013 by Narayanan Gopalakrishnan

2 View comments

Prasanna 6 May 2013 at 07:11

Brief & Nice explanation.
Request you to write blog on your understanding about UEFI. Would love to read that. It is becoming a hot topic these days.

Reply

Thinking-n-Evolving 20 April 2014 at 23:13

good summation of spinlock internals

Reply

Enter your comment...

Comment as: Aleem (Google ▼ Sign out

Publish Preview Notify me