delays - Information on the various kernel delay / sleep mechanisms

This document seeks to answer the common question: "What is the RightWay (TM) to insert a delay?"

This question is most often faced by driver writers who have to deal with hardware delays and who may not be the most intimately familiar with the inner workings of the Linux Kernel.

Inserting Delays

The first, and most important, question you need to ask is "Is my code in an atomic context?" This should be followed closely by "Does it really need to delay in atomic context?" If so...

ATOMIC CONTEXT:

You must use the **delay* family of functions. These functions use the jiffie estimation of clock speed and will busy wait for enough loop cycles to achieve the desired delay:

ndelay(unsigned long nsecs) udelay(unsigned long usecs) mdelay(unsigned long msecs)

udelay is the generally preferred API; ndelay-level precision may not actually exist on many non-PC devices.

mdelay is macro wrapper around udelay, to account for possible overflow when passing large arguments to udelay. In general, use of mdelay is discouraged and code should be refactored to allow for the use of msleep.

NON-ATOMIC CONTEXT:

You should use the **sleep[_range]* family of functions. There are a few more options here, while any of them may work correctly, using the "right" sleep function will help the scheduler, power management, and just make your driver better :)

-- Backed by busy-wait loop:

udelay(unsigned long usecs)

-- Backed by hrtimers:

usleep_range(unsigned long min, unsigned long max)

-- Backed by jiffies / legacy_timers

msleep(unsigned long msecs) msleep_interruptible(unsigned long msecs)

Unlike the **delay* family, the underlying mechanism driving each of these calls varies, thus there are quirks you should be aware of.

SLEEPING FOR "A FEW" USECS (< ~10us?):

- Use udelay
- Why not usleep?

  On slower systems, (embedded, OR perhaps a speed- stepped PC!) the overhead of setting up the hrtimers for usleep *may* not be worth it. Such an evaluation will obviously depend on your specific situation, but it is something to be aware of.

SLEEPING FOR ~USECS OR SMALL MSECS (10us - 20ms):

- Use usleep_range
- Why not msleep for (1ms - 20ms)?

  Explained originally here:
  https://lore.kernel.org/r/15327.1186166232@lwn.net

  msleep(1~20) may not do what the caller intends, and will often sleep longer (~20 ms actual sleep for any value given in the 1~20ms range). In many cases this is not the desired behavior.
- Why is there no "usleep" / What is a good range?

  Since usleep_range is built on top of hrtimers, the wakeup will be very precise (ish), thus a simple usleep function would likely introduce a large number of undesired interrupts.

  With the introduction of a range, the scheduler is free to coalesce your wakeup with any other wakeup that may have happened for other reasons, or at the worst case, fire an interrupt for your upper bound.

  The larger a range you supply, the greater a chance that you will not trigger an interrupt; this should

be balanced with what is an acceptable upper bound on delay / performance for your specific code path. Exact tolerances here are very situation specific, thus it is left to the caller to determine a reasonable range.

SLEEPING FOR LARGER MSECS (10ms+)

- Use msleep or possibly msleep_interruptible
- What's the difference?

  msleep sets the current task to TASK_UNINTERRUPTIBLE whereas msleep_interruptible sets the current task to TASK_INTERRUPTIBLE before scheduling the sleep. In short, the difference is whether the sleep can be ended early by a signal. In general, just use msleep unless you know you have a need for the interruptible variant.

FLEXIBLE SLEEPING (any delay, uninterruptible)

- Use fsleep