# Tick-tock - Understanding the Neutrino microkernel's concept of time

**Authored by:** **Brian Stecher**
**Updated by:** **Thomas Fletcher**

With a few hundred thousand new users and developers, we're quite certain that you're all going to have programming questions and problems. To help you on your journey towards realtime development, let's take a quick look at Neutrino's concept of time.

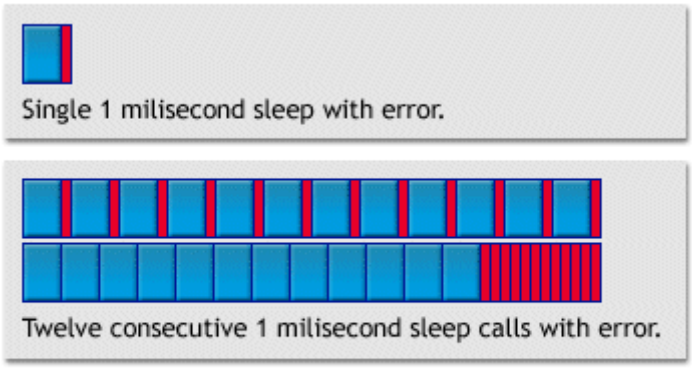
When you're dealing with timing, every moment within Neutrino microkernel is referred to as a tick. A tick is measured in milliseconds; its initial length is determined by the clock rate of your processor. If your CPU is 40 MHz or better, a tick is 1 ms. For slower processors a tick represents 10 ms. Programmatically you can change the clock period via the **ClockPeriod**() function.

This becomes important just about every time you ask the kernel to do something relating to pausing or delaying your process. These include the functions: **select**(), **alarm**(), **nanosleep**(), **nanospin**(), **sigaction**(), **delay**, as well the whole family of **timer_***() functions. Normally, we use these function assuming they'll do exactly what we say ... "Sleep for 8 seconds!", "Sleep for 1 minute!" and so on. Unfortunately, we get into problems when you ask, "Sleep for 1 millisecond, ten thousand times!"

***Does this code work assuming a 1 ms tick?***

```
void OneSecondPause()
{
    for ( i=0; i<1000; i++ ) delay(1); // Wait 1000 milliseconds
}
```

Unfortunately, no, this won't return after one second on IBM PC hardware. It'll likely wait for three seconds. In fact, when you call any function based on the nanosleep or select functions, with an argument of n milliseconds, it actually takes anywhere from n to infinity milliseconds. But more than likely, this example will take three seconds.




*Timer Quantization: Error Accumulation*

***So why, exactly does this function take three seconds?***

What you're seeing is called timer quantization error. One aspect of this error is actually something that's so well understood and accepted that it's even documented in a standard - the POSIX Realtime Extension (1003.1b-1993/1003.1i-1995). Within this document, it says that it's OK to delay too much, but it's not OK to delay too little. I'm sure we all know that the premature firing of a timer is undesirable...

Since the calling of delay() is asynchronous with the running of the clock interrupt, that means that we have to add one clock tick to a relative delay to ensure the correct amount of time (consider what would happen if we didn't and a one tick delay was requested just before the clock interrupt went off). That normally adds half a millisecond each time, but in the example given we end up synchronized with the clock interrupt, so the full millisecond gets tacked on each time.

***OK, that should make the loop last 2 seconds, where's the extra second coming from?***

The problem is that when you request a 1 ms tick rate, we may not be able to actually give it too you because of the frequency of the input clock to the timer hardware. In those cases, we choose the closest number that's faster than what you requested. In terms of IBM PC hardware, requesting a 1 ms tick rate actually gets you 999,847 nanoseconds between each tick. With the requested delay, that gives us the following:

- 1,000,000 ns + 999,847 ns = 1,999,847 ns of actual delay.
- 1,999,847 ns / 999,847 ns = 2.000153 ticks before the timer expires

Since we only expire timers at a clock interrupt, ceil(2.000153) gives us that each delay(1) call actually waits:

999,847 ns * 3 = 2,999,541 ns

Multiply that by a 1000 for the loop count and you get a total loop time of 2.999541 seconds.

***So this code should work?***

```
void OneSecondPause()
{
    for ( i=0; i<100; i++ ) delay(10); // Wait 1000 milliseconds
}
```

It will certainly get you closer to the time you expect, with an accumulated error of only 1/10 of a second.

***In Conclusion...***

Certainly, this is a simple error, but you'll probably encounter the smallest errors first! Don't let these things slow you down. When you run into something that you're sure should work, and the documentation just isn't helping, please post in the forums and ask for help. Your development will go a lot smoother and we'll know when and where to improve our documentation, or we can write an article about your problems.