**Using and Understanding the Real-Time Cyclictest Benchmark – Frank Rowand (Sony)**

Posted on [October 25, 2013](https://mindlinux.wordpress.com/2013/10/25/using-and-understanding-the-real-time-cyclictest-benchmark-frank-rowand-sony/) by [Arnout Vandecappelle](https://mindlinux.wordpress.com/author/arnoutvandecappelle/) | [Leave a comment](https://mindlinux.wordpress.com/2013/10/25/using-and-understanding-the-real-time-cyclictest-benchmark-frank-rowand-sony/#respond)

Cyclictest is the one benchmark that RT people jump to when evaluating the RT performance of a system. It’s a relatively simple test, but drawing conclusions from its results can be challenging.

Cyclictest doesn’t measure the entire event-to-response time, it measures the time from event to the start of real work (i.e. that which is influenced by the OS). The latency can come from: interrupts being disabled when the event occurs; other interrupts arriving; nested interrupt handling; wake-up of the cyclictest executable; wait until preemption is re-enabled or higher priority task yields or same-priority task is pre-empted; scheduler overhead; processor sleep states, process migration, cache misses in the OS; lock priority inheritance etc.; and there may be interactions between these. cyclictest gives you a single number for the total of all of this.

What cyclictest does is basically clock\_nanosleep followed by clock\_gettime. In the ideal case, the requested time should be identical to the measured time; the difference is the latency. The whole of cyclictest is about 3000 lines of code, but the core is basically this. Cyclictest starts a number of threads. Since there is a varying setup time for each thread, they will not run in lockstep, so the wakeups from the sleep will not interfere with each other – and we want to measure exactly that. Therefore, there’s a patch that adds an option to make sure all threads start at the same time.

Cyclictest runs through a number of sleeps. The latency is the maximum of all measured latencies. It may miss tlatency sources that occur at a different time than the regular wakeups of cyclictest. It also measures only the timer IRQ latency; if you want to know about an event coming from another IRQ source, the code path is different: IRQ may have a different priority, and it runs different code (typically a driver works mostly outside of interrupt context while a timer doesn’t).

Another problem is that there is some stuff in Linux that has global effect: module loading and unloading and CPU hotplug (may) cause stop\_machine, which also affects CPUs that you have allocated fully to RT work.

To have better measurements, run the real workload together with cyclictest, and run cyclictest at a higher priority. The latency cyclictest sees in that situation is a good indication of the application’s latency. You can also vary cyclictest’s priority to see how the lower-priority threads of your application will behave.

When interpreting the results, it’s important to look at the histogram and not just average and maximum. Also run it for a *long* time, a million loops is really not a lot, a 100M loops is good. Also you need to really tune the cyclictest options to match the environment of what you want to measure. There are a lot of options: the behaviour of each thread, output, debug options (e.g. capture ftrace when a maximum is reached). Some options have side effects, e.g. real-time output influences the measurements. Option parsing is buggy so take care.

So if you get a result, is it good or bad? On the rt.wiki.kernel.org there is a list of results for various processors and boards, so you can check if you’re in the same league as you would expect – also use the mailing list because the table is a bit out of date. OSADL.org tests a wide variety of systems (OSes and boards).

Printing each latency is useful to determine where the latency comes from, e.g. a periodic issue will become visible. For instance, if you forgot to turn off throttling that will be visible.

You can also use cyclictest to evaluate of the RT application has met its deadlines: run a cyclictest at a priority above your RT application, and another one below the RT application. The difference between the two latency is approximately the total latency of the RT application. Note that it is not accurate for a single sample because the two cyclictests aren’t synchronized.

There is also a graphical tool that allows you to view a live graph.