Softlockup detector and hardlockup detector (aka nmi_watchdog)

The Linux kernel can act as a watchdog to detect both soft and hard lockups.

A 'softlockup' is defined as a bug that causes the kernel to loop in kernel mode for more than 20 seconds (see "Implementation" below for details), without giving other tasks a chance to run. The current stack trace is displayed upon detection and, by default, the system will stay locked up. Alternatively, the kernel can be configured to panic; a sysctl, "kernel.softlockup_panic", a kernel parameter, "softlockup_panic" (see "Documentation/admin-guide/kernel-parameters.rst" for details), and a compile option, "BOOTPARAM_SOFTLOCKUP_PANIC", are provided for this.

A 'hardlockup' is defined as a bug that causes the CPU to loop in kernel mode for more than 10 seconds (see "Implementation" below for details), without letting other interrupts have a chance to run. Similarly to the softlockup case, the current stack trace is displayed upon detection and the system will stay locked up unless the default behavior is changed, which can be done through a sysctl, 'hardlockup_panic', a compile time knob, "BOOTPARAM_HARDLOCKUP_PANIC", and a kernel parameter, "nmi_watchdog" (see "Documentation/admin-guide/kernel-parameters.rst" for details).

The panic option can be used in combination with panic_timeout (this timeout is set through the confusingly named "kernel.panic" sysctl), to cause the system to reboot automatically after a specified amount of time.

Implementation

The soft and hard lockup detectors are built on top of the hrtimer and perf subsystems, respectively. A direct consequence of this is that, in principle, they should work in any architecture where these subsystems are present.

A periodic hrtimer runs to generate interrupts and kick the watchdog job. An NMI perf event is generated every "watchdog_thresh" (compile-time initialized to 10 and configurable through sysctl of the same name) seconds to check for hardlockups. If any CPU in the system does not receive any hrtimer interrupt during that time the 'hardlockup detector' (the handler for the NMI perf event) will generate a kernel warning or call panic, depending on the configuration.

The watchdog job runs in a stop scheduling thread that updates a timestamp every time it is scheduled. If that timestamp is not updated for 2*watchdog_thresh seconds (the softlockup threshold) the 'softlockup detector' (coded inside the hrtimer callback function) will dump useful debug information to the system log, after which it will call panic if it was instructed to do so or resume execution of other kernel code.

The period of the hrtimer is 2*watchdog_thresh/5, which means it has two or three chances to generate an interrupt before the hardlockup detector kicks in.

As explained above, a kernel knob is provided that allows administrators to configure the period of the hrtimer and the perf event. The right value for a particular environment is a trade-off between fast response to lockups and detection overhead.

By default, the watchdog runs on all online cores. However, on a kernel configured with NO_HZ_FULL, by default the watchdog runs only on the housekeeping cores, not the cores specified in the "nohz_full" boot argument. If we allowed the watchdog to run by default on the "nohz_full" cores, we would have to run timer ticks to activate the scheduler, which would prevent the "nohz_full" functionality from protecting the user code on those cores from the kernel. Of course, disabling it by default on the nohz_full cores means that when those cores do enter the kernel, by default we will not be able to detect if they lock up. However, allowing the watchdog to continue to run on the housekeeping (non-tickless) cores means that we will continue to detect lockups properly on those cores.

In either case, the set of cores excluded from running the watchdog may be adjusted via the kernel.watchdog_cpumask sysctl. For nohz_full cores, this may be useful for debugging a case where the kernel seems to be hanging on the nohz_full cores.