Stop Polling! The Case Against OS Ticks

Dan Tsafrir* Yoav Etsion Dror G. Feitelson
The Hebrew University of Jerusalem, Israel

1 The Problem

All general-purpose operating systems (GPOSs) use periodic clock interrupts called "ticks" to regain control and measure the passage of time. On each tick the kernel performs administrative tasks like accounting the CPU time used by the current process, designating it for preemption if needed, waking processes with pending signals, etc. This mechanism has been in use since the late 1960s. However, due to the rapidly growing applicability of GPOSs (ranging from as little as mobile phones and PDAs to as large as supercomputers), drawbacks of periodic timing accumulate into a critical mass, suggesting it's time for a change. Indeed, we have identified quite a few mainstream system domains that inherently conflict with the polling nature of ticks:

Mobile and embedded devices waste power on unnecessary ticks that happen even if the machine is otherwise idle; power is also wasted even if the machine is busy, as tasks run longer than necessary due to the indirect overhead of ticks (see below). We show an idle "crippled" laptop (disconnected from its screen and hard disk) consumes 4W due to ticks, and more, for increased tick rates. This is the result of ticks continuously preventing the processor from maintaining a power save mode [1].

Virtual machine settings suffer from excessive overhead. The base overhead of ticks is intensified when a VMM/hypervisor is positioned between the ticking OS and the hardware. Further, VM servers can be overwhelmed by ticking guest OSs. One example is an S/390 mainframe for which servicing clock interrupts of multiple idle VMs (running Linux) led to 100% utilization of the physical processor [2].

Ticks enable denial-of-service attacks. CPU consumption accounting is done in tick resolution. We show that any unprivileged "cheater" process can take advantage of this to monopolize the machine, by sleeping when ticks take place and systematically avoiding being billed. The fact cheaters appear to consume no CPU makes their priority very high and allows them to run whenever they choose while starving "honest" processes [3].

Ticks are a security breach. A side-effect of being able not to be billed is that monitoring applications like the UNIX 'top' utility report such cheater processes as consuming 0% CPU, essentially making them "invisible". We show that cheaters can get as much or as little CPU cycles as they want, without this information showing up on monitors [3]. Knowing about an attack is essential to stopping it, and so the fact offending processes do not appear on CPU monitors constitute a serious security breach.

Parallel applications suffer from "noise" (OS activity unrelated to the application), where one late process holds up hundreds to thousands of peers with which it synchronizes, leaving the entire parallel machine idle until the late process catches up. We analytically quantify the effect and empirically show ticks' noise is a major source of degraded performance in supercomputer settings [4].

Desktop applications suffer from slowdown due to indirect overheads of useless kernel-user context switching that is triggered by ticks. We show that this penalty can be as high as 8% for various commodity Pentium-IV machines [4].

Soft realtime and multimedia tasks suffer from limited clock resolution. For example, we show that a movie player displaying a clip can lose up to a third of its frames because the resolution of alarm-timers is limited by the tick rate. We also show that alarm latency can be greatly reduced when increasing the tick rate, but that this incurs severe overhead penalties [5].

Hard realtime systems experience difficulties in predicting deterministic timing behavior, as ticks may occur while tasks are running. We show the duration of periodic work is susceptible to significant variance [4], which e.g. may even be dependent on the number of processes present in the system [6].

Micro kernels complexity is increased if tick-related code is included in the kernel. Reducing the size of the kernel is essential for obtaining more dependable systems. Micro kernels' code base could be further reduced if the timing mechanisms are pushed away from the kernel, but this is not done due to overhead considerations as ticks are too frequent (e.g. Minix3's 4,000-lines kernel includes the timing subsystem). Eliminating ticks can largely solve this problem.

2 The Solution

The source of the above problems is periodic ticks that, as mentioned earlier, turn the OS into a polling-based system. The alternative is to go event-based by leveraging the fast "one-shot timers" mechanisms (timers that are set only for specific needs) made available by commodity hardware (e.g. APIC and HPET). However, simply doing this in a general-purpose setting is not an option, as it allows for any user to essentially bring the system down e.g. by generating numerous events with nanosecond differences. This can be solved by aggregating the events, e.g. by using a modified version of "firm timers" [7] that eliminates the periodic component from this mechanism. A description of our solution can be found in [1].

References

[1] D. Tsafrir, Y. Etsion, and D. G. Feitelson. General-purpose timing: The failure of periodic timers. Tech. Report 2005-6, The Hebrew University, Feb 2005.
[2] M. Schwidefsky, A. Cox, and many others. No 100 HZ timer! URL http://lkml.org/lkml/2001/4/9/79, Apr 2001. Linux Kernel Mailing List.
[3] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Is your PC secretly running nuclear simulations? Tech. Report 2006-78, The Hebrew University, Sep 2006. Submitted.
[4] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, OS clock ticks, and fine-grained parallel applications. In *ACM ICS*, Jun 2005.
[5] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Effects of clock resolution on the scheduling of interactive and soft real-time processes. In *ACM SIGMETICS*, Jun 2003.
[6] D. Tsafrir. Barrier synchronization on a loaded SMP using two-phase waiting algorithms. Master's thesis, The Hebrew University, Sep 2002.
[7] A. Goel, L. Abeni, C. Krasic, J. Snow, and J. Walpole. Supporting time-sensitive applications on a commodity OS. In *USENIX OSDI*, Dec 2002.

*Current affiliation: IBM T. J. Watson Research Center.