

Re: after ids

Source: <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2008-07/msg00244.html>

- *From:* Kevin Kenny <kennykb@xxxxxxx>
 - *Date:* Sat, 05 Jul 2008 12:56:23 -0400
-

Donal K. Fellows wrote:

OK, how do you work out how long until the next event fires without an absolute time source? For added bonus points, do so with arbitrary real-time delays between processing points and without extra threads.

Donal,

Many people here are unquestionably arguing from ignorance, but I do suspect after some conversations with Alexandre Ferrieux that they are onto something with wanting a distinction between absolute and relative times.

Absolute time, as far as Tcl is concerned, is UTC with smoothing (<http://www.cl.cam.ac.uk/~mgk25/uts.txt>). It advances monotonically and (more or less) continuously, ticking at an uneven rate to accommodate TAI-UTC differences and NTP corrections. It **can** be nonmonotonic or discontinuous in unusual circumstances, owing to an act of God (a network outage causing too large an NTP jump to recover by advancing or retarding clock frequency), an act of the operator (adjusting system time by the wristwatch-and-eyeball method), or an Act of Congress (Daylight Saving Time, on operating systems that use civil time as the reference).

Its reference on Unix-like systems is `gettimeofday()`; on Windows systems its ultimate reference is the system clock returned by `GetSystemTimeAsFileTime`, with additional precision provided by interpolation using the performance counter.

Relative time is not something that Tcl currently recognises, but perhaps should. It is strictly monotonic and uniform; it can be thought of as the independent variable of Newton's Laws of Motion. It is not – indeed, cannot be – tied to any concept of absolute time outside the computer, since all the "acts" above can cause dislocations. Nevertheless, it appears to be what people want in scheduling short-period tasks. Moreover, the anomalies in time handling on Windows (where near the Spring and Autumn transitions of Daylight Saving Time,

Re: after ids

the "UTC" returned by the system can be an hour off!) appear to be with us to stay. I've seen bugs in that area in every Windows release from 3.1 to Vista.

We have historically (and I am an offender in that respect) offered the answer, "make your NTP infrastructure work!" That's certainly part of the answer, and necessary for other reasons (such as making sure that the system clock is reasonably synchronised with the clock of remotely-mounted file systems). It does not address the fact that there are timing windows at the Daylight Saving Time transitions during which the Windows API returns times that are simply incorrect. It also fails to address the concerns of those who deal with systems that have only intermittent network connectivity or lack it altogether. (Tcl gets ported to some very strange places, indeed.)

Given these considerations, I think we can all agree that, all else being equal, we might gain something by distinguishing the relative timing function (today's [after] used for short-period tasks with loose accuracy being tolerable) and the absolute timing function (delivery of an event at a given time in the future, accounting for any known dislocations of the clock). The latter might be implemented by a command analogous to 'after' that accepts seconds (milliseconds, microseconds, choose a convenient unit) of UTS time from the epoch instead of milliseconds from the current time.

The chief argument against that approach, we have stated in the past, is infeasibility. Unquestionably, we have system calls that delay a certain length of time irrespective of changes to the civil clock – indeed, sleep() and select() on Unix; and Sleep and MsgWaitForMultipleObjectsEx on Windows work that way. What we have lacked, or so we have claimed, is a reliable way to handle multiple relative timers – once the first timer has rung, how do we wind the second timer with the now-shorter interval that it needs? What is needed is a reference clock that advances with the properties that we require (monotonicity, uniformity, but not accuracy nor synchrony with the outside world).

I – and apparently you, Donal – had long believed that no portable way existed to get such a clock, but I am coming to suspect that the world is catching up with our needs. On Windows, the information is available with the GetTickCount function, which has existed since the beginning (Windows attempts to calibrate its rate to NTP corrections, but guarantees its monotonicity except for the 49.7-day rollover, which we could deal

Re: after ids

Re: after ids

with easily). Unix-like systems are a tougher nut to crack, but with the Posix standards being implemented more and more widely, I suspect that the times() function is a reasonable reference on Unix-like systems. While its primary purpose is to retrieve CPU time for a process, it has the side effect of returning clock ticks since an arbitrary point in the past, advancing in monotonic fashion. (This reference is needed when successive calls to times() are used to compute percent-CPU-usage.)

It remains to be seen whether the combination of times() and GetTickCount() will work on all the platforms that we support, but it's also something that we could exploit anyway, by having the configurator check for the routines and fall back upon today's usage of absolute time if they are not available.

I don't personally have the time at the moment to tackle such a project, but if someone else wants to draft a TIP and attempt a reference implementation, I'd be willing to shepherd them through the process.

—

73 de ke9tv/2, Kevin

.