High Resolution Timing with Low Resolution Clocks

and

A Microsecond Resolution Timer for Sun Workstations

Peter B. Danzig
Stephen Melvin

Computer Science Division
University of California, Berkeley
Berkeley, California 94720
email: danzig@ucbarpa.Berkeley.EDU, melvin@ucbarpa.Berkeley.EDU

When tuning operating system and network code, profiling programs, analyzing message interarrival times, and accurately measuring device characteristics, a high resolution clock is often indispensable, as one cannot measure service time *distributions* without one. This note describes a microsecond clock that we designed and built for Sun 3 and Sun 4 workstations[1]. One can measure average service times without a high resolution clock. This paper explains how to measure average times with high precision in the absence of such a clock. We pose and answer the question: "how many measurements are needed to report timing data to three significant digits?"

1. Introduction - Who Needs a Microsecond Clock

Beginning with its Sun 3 workstations, Sun Microsystems substituted an Intersil, battery backed up, time-of-day clock chip for the microsecond resolution clock chip present in their earlier models. The new clock interrupts the processor every ten milliseconds. By default, the Sun operating system discards every other interrupt, degrading the clock resolution from ten to twenty milliseconds. Sun kept an I.C. socket for a data encryption chip (DES), but chose to leave it and up to three other support sockets empty[2]. As we had no use for the DES chip, we designed a high resolution clock to plug directly into the DES chip's socket. To install the clock, one only needs to insert it and the support chips into their associated sockets and add a device driver to the operating system. In October 1989, we had three dozen of these clocks in use at U.C. Berkeley and other universities and laboratories.

In the next section we describe our clock's design. In Section 3 we derive the number of measurements needed to accurately report average timing data as a function of the clock's resolution. We show that without a microsecond clock, it may require several hours or days to report average timing data to three significant digits. We draw conclusions in Section 4.

2. Our Design

In this section we describe our clock's design. Because Sun guards its workstation's schematics, we designed our clock to meet the timing requirements and eight-bit interface of the Advanced Micro Devices (AMD) Am9518 DES chip (also known as the Zilog Z8068). We built the clock around AMD's Am9513a counter chip because its five, sixteen-bit counters can be atomically saved with a single instruction yet read over an eight-bit bus. While other counter chips have multiple sixteen-bit timers, the Am9513a is the only chip that can save more than one timer with a single instruction. Although the timer and DES chips carry similar designations, their pin-out, interface protocol, and timing needs are quite different.

Both chips have a data port and a control port which can be written or read. The DES chip selects the appropriate port with separate data-strobe and control-strobe pins. The timer chip's single strobe serves for both

[1] Contact us to obtain the schematic diagram, SunOS device driver, or completed timer boards. Note that we cannot support Sun 3/80, 386i, or Sparc Stations, but Sparc Stations have an internal microsecond timer.

[2] Sun 3/50 and 3/60 workstations do not need additional support chips. Sun 3/75, 3/140, 3/150, 3/160 systems require a 74ALS245 octal-buffer and a PAL22V10. Sun 3/260 and 3/280 systems require a 74ALS245, a PAL16R4, and a PAL16R8. Sun 4/110, 4/150, 4/260, 4/280 systems require a PAL22V10.

ports; its data/control pin selects between the two ports. The DES chip has a single read/write pin; the timer chip has separate read and write pins. We placed a programmable logic array (PAL) on our timer board that converts the DES chip's control signals into the timer chip's control signals. One cannot meet the timer chip's data/control pin's setup and hold requirements given the DES chip's published timing specifications. As we could neither modify the Sun's hardware nor firmware, yet wanted the board to work in all Sun 3 and Sun 4 processors, we chose a solution that adds a few instructions to the sequence of instructions necessary to read the timer. We drive the timer's data/control pin from a set-reset flipflop built from two of the PAL's gates. The data port is selected when this flipflop is set and the control port when it is reset. We precede accesses to both ports by appropriately setting or resetting this flipflop from the DES chip's data strobe and read/write signal.

The timer board's device driver sets the timer chip's fifth timer to divide the 4.0 megahertz oscillator frequency by four, concatenates the timer chip's lower four timers, and drives the lowest of these with the output of the fifth timer. The board can return a simple binary count or a 64-bit UNIX *timeval* structure. The *timeval* mode is useful for compatibility with the UNIX system call *gettimeofday()*. The clock appears as device */dev/tmr0* and can be read through the file system or through a system call. It can also be mapped into the user's address space, giving user programs quick access to the timer's registers and the microsecond time.

In Figure 1 we report the overhead associated with reading 32-bit timestamps and 64-bit *timevals*. Note that a 3/50's display steals cycles from main memory, as it does not have a separate frame buffer. Hence we give two sets of overhead figures for it, one for when the display is blanked and another for when it is active. When the display is active, the machine slows down by more than twenty percent. The overhead varies a few microseconds from read to read due to the speed of the memory and memory contention. Infrequently, when reading the clock from a user process, the process may be descheduled within the instrumented code, resulting in large times. This is, unfortunately, unavoidable, but easily detectable; any clock would suffer the same inconvenience. Interrupts can also increase the measured time. Since user programs cannot disable interrupts they cannot read the clock atomically when it is mapped into the user's address space, and it may return nonsensical values if other user processes or the operating system also read the clock. (User programs can always read the clock atomically through the system call). This occurs because the competing process may read the timer, which resets the timer chip's internal pointer. The original process, when it resumes, will continue reading bytes from where it left off, unaware that these are not the bytes that it wants.

3. Profiling Code with a Low Resolution Clock

Perusing the operating system's literature, we often see tables of performance measurements collected on computers with poor clock resolution [1,2,4,5]. The highest possible resolution of an IBM PC/RT is 125 microseconds; the clock resolution of microVAX-II workstations is 20 milliseconds, and, as we have mentioned, the highest possible resolution of Sun 3 and Sun 4 workstations is 10 milliseconds. Often practitioners report times as short as 10-300 microseconds to three decimal places based upon the average of a few dozen to a million iterations through the code. Instrumenting code and making measurements can be quite time consuming. For example, measurements of network code are usually repeated for several sizes of messages, and measurements of transaction systems are usually repeated for various numbers of participants. Let us consider the process by which we collect measurements and then pose the following question. How many iterations suffice to report our measurements to two or three significant digits given our hardware clock advances every Δ milliseconds?

We profile code by recording the difference in the clock's values upon entering and exiting each instrumented code segment. For example, these segments could correspond to the various layers of a communication protocol stack. Since the clock time only advances every Δ milliseconds, it may not advance between entering and exiting a

Timestamp Type	3/50	3/50 Blanked	3/60	3/260
Kernel 32-bit	24.0	19.5	16.5	11.
Kernel 64-bit timeval	38.2	30.2	27.	18.
User 32-bit	14.0	11.3	11.1	7.
User 64-bit timeval	27.0	23.5	21.	13.
System call 32-bit	238.	179.	140.	87.
System call 64-bit timeval	254.	190.	162.	91.

Figure 1. Measured overhead in microseconds to read the high resolution clock (we do not state the precision and degree of confidence of the measured overhead because overhead is machine dependent).

code segment of duration less than Δ milliseconds. Without loss of generality, assume that we want to measure a code segment of duration $\delta \le \Delta$ milliseconds. First we must assure ourselves that the measurements are not initiated by (or otherwise synchronous to) clock ticks. During a code segment of duration δ the clock advances with probability $p = \delta/\Delta$. If the clock advances we record a one; if not, we record a zero. We define the event σ_i to be one if the clock advances during iteration i and zero otherwise. After n iterations the clock advances S_n ticks:

$$S_n = \sum_{i=1}^{n} \sigma_i \, .$$

After n iterations, the average time through this code segment is

$$\hat{\delta} = \hat{p}\ \Delta = \frac{S_n\ \Delta}{n} \, ,$$

and, in our research papers, usually report this value to two or three decimal places.

Let us consider the precision of measurements made in this manner. Notice that the σ_i's approximate the two possible outcomes of a sequence of Bernoulli trials [3]. We say approximate because the outcomes of subsequent trials may not be totally independent. That is, if we know that the clock advanced during the last measurement, this may increase the probability that the clock does not advance during this measurement. Assuming independence, we want to know how many iterations are required to report these times to two or three significant digits. Since our measurements result from randomized experiments, we can only make probabilistic statements about their precision. Conceivably, but with low probability, we may never observe a single clock tick. Let $\delta = p\ \Delta$ be the true interval length and $\hat{\delta}$ be our estimate of it based upon n observations. We would like to report our experimental data as

$$P\left[-\varepsilon \le (\hat{\delta} - \delta) \le \varepsilon \right] \ge \alpha \, . \tag{1}$$

That is, with probability α or greater our measurement is within ε of the true value. We can apply the DeMoivre-Laplace limit theorem to find the minimum number of iterations required to make such a statement. The DeMoivre-Laplace theorem states

$$P\left\{ z_1\sqrt{npq} \le S_n - np \le z_2\sqrt{npq} \right\} \tag{2}$$

$$\rightarrow \Phi(z_2) - \Phi(z_1) \, ,$$

where $q = 1 - p$ and $\Phi(x)$ is the normal distribution function. We will need the following identity concerning $\Phi(x)$:

$$\Phi(-x) = 1 - \Phi(x) \, . \tag{3}$$

We can combine (1), (2), and (3) into a single expression:

$$P\left\{ \frac{-\varepsilon n}{\Delta} \le S_n - np \le \frac{\varepsilon n}{\Delta} \right\} \rightarrow 2\Phi(z_\alpha) - 1 \, .$$

Equating $\varepsilon n/\Delta$ with $z_\alpha\sqrt{npq}$, we arrive at an expression for n:

$$n = \frac{z_\alpha{}^2 \Delta^2 pq}{\varepsilon^2} \, , \tag{4}$$

for which we must evaluate z_α:

$$2\Phi(z_\alpha) - 1 = \alpha \, . \tag{5}$$

For the remainder of this paper, we use an arbitrary confidence level of $\alpha = 0.95$. We invert (5) from tables of the normal distribution:

$$z_{.95} = \Phi^{-1}(0.975) = 1.96 \, .$$

Below we summarize the number of iterations n_3 and n_2 required to report times to two and three decimal places such that the least significant digit is within one unit of the true value with probability $\alpha = 0.95$.

Consider the problem of instrumenting a UDP/IP protocol stack implementation. Conceivably each iteration could take about 10 milliseconds. In Figure 2 we see that measuring to three significant digits a code segment of about a millisecond duration requires about $7 \cdot 10^5$ iterations. At 10 milliseconds per iteration, we will need to run this experiment for about two hours. If some profiled section of this code is on the order of a hundred microseconds, then the experiment could have to run for an entire day!

4. Conclusions

Reporting performance data to two or three significant digits with high confidence may require many iterations, depending upon the clock resolution. When we report performance numbers of intervals shorter than our clock's resolution, we should perform enough iterations to achieve a high confidence level for the number of significant digits that we report (see (4)). Without a high resolution clock, this may require many hours of measurements.

When it is necessary to measure a distribution rather than an average, a microsecond resolution clock may be essential. The microsecond resolution clock described in this paper will work in all Sun workstations equipped with the DES socket. A rudimentary device driver for the clock can be written in three pages. Finally, we note that it is possible to build a clock like ours in place of non-essential chips in other computers.

References

1. Beilner, H., "Measuring with Slow Clocks", *ICSI-Technical Report-88-003*, Berkeley, California, July, 1988.
2. Carter, J. B. and Zwaenepoel, W., "Optimistic Implementation of Bulk Data Tranfer Protocols", *1989 ACM SIGMETRICS Conference* , Berkeley, CA, May 23-26, 1989, 61-69.
3. Feller, W., *An Introduction to Probability Theory and Its Applications, Vol. 1*, John Wiley & Sons, New York, NY, 1970.
4. Haskin, R., Malachi, Y., Sawdon, W. and Chan, G., "Recovery Management in QuickSilver", *Trans. Computer Systems 6*, 1 (Feb., 1988), 82-108.
5. Renessee, R., Staveren, H. and Tanenbaum, A. W., "Performance of the World's Fastest Distributed Operating System", *Operating System Reviews 22*, 4 (Oct., 1988), 25-34.

δ	n_3	n_2
$1 \cdot 10^1$ μSec	$8 \cdot 10^7$	$8 \cdot 10^5$
$1 \cdot 10^2$ μSec	$8 \cdot 10^6$	$8 \cdot 10^4$
$1 \cdot 10^3$ μSec	$7 \cdot 10^5$	$7 \cdot 10^3$
$1 \cdot 10^4$ μSec	$4 \cdot 10^4$	$4 \cdot 10^2$

Figure 2. Required number of iterations to measure code segments of duration δ to three and two significant digits given a clock with $\Delta = 20$ millisecond resolution to a confidence level of $\alpha = 0.95$.