Novi Sad J. Math.
Vol. 33, No. 1, 2003, 67-73

A PROBLEM OF PROGRAM EXECUTION TIME MEASUREMENT

Miroslav Hajduković[1], Zorica Suvajdžin[1], Žarko Živanov[1], Edin Hodžić[2]

Abstract. This paper discusses the problem of program execution time measurement. Program execution time measurement is problematic because the conventional measurement method, based on the real-time clock interrupt counting, is prone to inherent errors. These errors are caused by discrete nature of real-time clock interrupts and by the overhead of clock interrupt processing. Ways of reducing relative magnitude of these errors are described. The program execution time measurement and error estimation are illustrated by the measurement of real-time kernel operations' execution time on the Intel 80386 EX micro-controller.

AMS Mathematics Subject Classification (2000): 68M20

Key words and phrases: software measurement, performance, error estimation, real–time kernel for micro–controllers

1. Introduction

After the development of the real–time kernel for industrial micro–controllers, it is only natural to ask how much "real–time" it is. An answer is obtained by measuring the execution time of the real–time kernel operations, and it depends on the micro–controller the kernel executes on. In order to ensure uniformity of measured results across different micro-controller platforms, the measurement method need to be uniformly applicable to different platforms. One such method is time measurement based on counting clock interrupts of a real-time clock with uniform period [2]. This software–based approach assumes that the measured time is determined as a product of the number of observed clock interrupts and the clock period. The real–time kernel supports software-based time measurement by providing an operation, e.g. *ticks_get*, that returns the number of real–time clock interrupts since the beginning of real–time kernel execution. Software measurement is simplified if the kernel supports execution of the processes and interrupts only directly related to the measurements. This assumption is applicable for micro–controller targeted real–time kernels.

Software–based measurement is illustrated by the example of the function *measure*, dedicated to the measurement of the operation *op*. If the number

[1]Department of Computing and Control, Faculty of Engineering, University of Novi Sad
[2]ehodzic@scudc.scu.edu

```
int measure()
{
    int old_ticks, i, for_ticks;

    old_ticks = ticks_get ();
    for (i = 0; i < N; i++)
        ;
    for_ticks = ticks_get () - old_ticks;

    old_ticks = ticks_get ();
    for (i = 0; i < N; i++)
        op ();
    ticks = (ticks_get () - old_ticks) - for_ticks;
}
```

Figure 1: Function *measure* (the function is simplified by avoiding all details intended to preclude unwanted compiler optimizations).

of clock interrupts is taken before and after the operation, then the difference between the two numbers of interrupts determines the length of execution of the operation *op*. In the case the length of execution of the operation *op* is lower or comparable to the real–time clock period, then it is appropriate to measure $N >> 1$ executions of the operation *op*. In that case, the number of clock interrupts needs to be adjusted by subtracting the overhead of the *for* loop. Figure 1 shows the process *measure*. It is shown as a C function of type *int*. Process *measure* assumes that the function *op* takes no arguments and returns no values. It is assumed that the integer constant N and variable *ticks* are defined outside the process *measure*.

Upon completion of the function *measure*, variable *ticks* contains the number of real–time clock interrupts necessary for N executions of operation *op*. Therefore, the software measured execution time of a single *op* execution is given by:

$$t = \frac{ticks \times PERIOD}{N}, \tag{1}$$

where the constant $PERIOD$ is the real–time clock period.

The question of how close are the software–based measured time based on (1), and the real execution time of an operation, is considered in the sequel of the paper. The software–based measurement error is analyzed, its significance discussed, and an example of software-based real–time kernel operation measurement is presented.

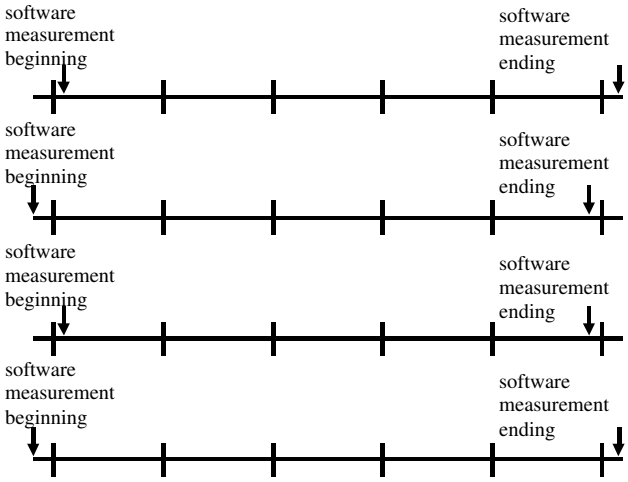

Figure 2: Possible relations of the interrupt counting and software–based measurement events.

2. The software–based measurement quantization error

Software–based measurement inaccuracies are caused by clock interrupts. One of inaccuracies is the quantization error. Software–based measured time is expressed as an integer multiple of clock periods although the real execution time might include only a fraction of a period. As a consequence, the software–based measured time and real execution time could differ by up to one clock period. Similar difference could also be caused by interrupt counting error. It is dependent on the relation between the events of interrupt counting and software-based measurement beginning and ending. The four possible scenarios are illustrated in Figure 2.

Figure 2 assumes that the beginning and ending of software–based measurement events are almost aligned with interrupt counting events, i.e. it assumes that the real execution time is close enough to multiple of the clock period, for example close enough to 5 clock periods. In the first two cases, software–based measurement is equal to the real execution time. The other two cases show a positive or negative difference of one clock period, as a result of counting error. Table 1 summarizes the software–based measured time compared to the real execution time.

It is important to note that the counting error is an upper bound to the quantization error, i.e. these two errors do not accumulate. Based on this analysis, it follows that the software–based measured time of operation *op* is

Clock interrupts	software	real	real – software
5.00	$5 \times PERIOD$	$5 \times PERIOD$	0.00
5.00	$5 \times PERIOD$	$5 \times PERIOD$	0.00
4.00	$4 \times PERIOD$	$5 \times PERIOD$	$+PERIOD$
6.00	$6 \times PERIOD$	$5 \times PERIOD$	$-PERIOD$

Table 1: Difference between software–based measured time and real execution time.

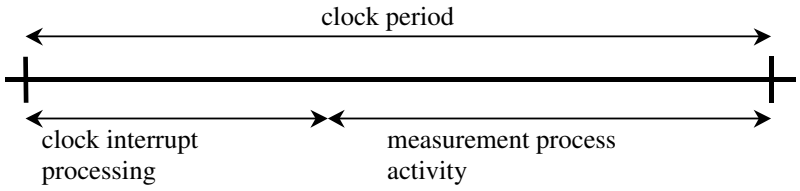

Figure 3: Clock period division.

given by:

$$t = \frac{ticks \times PERIOD}{N} \pm 2\frac{PERIOD}{N}. \tag{2}$$

The coefficient 2 in (2) is a result of calculating variable *ticks* as a difference of two software–based measurements, each of which can have one clock period error. The discussed error can be ignored if the value of variable *ticks* is much greater than 2, which is achieved for large values of N. In that case, we can use equation (1) instead of equation (2).

3. The software–based measurement overhead error

Another type of software–based measurement error is caused by the clock overhead. Clock interrupts are processed by the same processor, and thus only a fraction of the clock period is spent on processing the measured operation. Figure 3 illustrates a single period division between clock interrupt processing and measured operation processing.

Based on Figure 3, the software measured time of a single operation *op* is given by:

$$t = \frac{ticks \times (PERIOD - OVERHEAD)}{N}, \tag{3}$$

where $OVERHEAD$ is the clock interrupt processing time. Therefore, it is necessary to know exact value of the constant $OVERHEAD$. It is important

```
int overhead()
{
    int old_ticks, i;

    old_ticks = ticks_get ();
    for (i = 0; i < N; i++)
        ;
    ticks = ticks_get () - old_ticks;
}
```

Figure 4: Function *overhead* (the function is simplified by avoiding all details intended to preclude unwanted compiler optimizations.)

to know the value of $OVERHEAD$ because its ratio to the value of $PERIOD$ determines the processor utilization. Assuming $PERIOD > OVERHEAD$, the processor utilization is expressed as:

$$\frac{PERIOD - OVERHEAD}{PERIOD}. \tag{4}$$

Value of the constant $OVERHEAD$ can be calculated using the results of the function *overhead* shown in Figure 4.

It is assumed that the integer constant N and variable *ticks* are defined outside the function *overhead*. The time T, needed to execute loop *for* in function *overhead*, is independent of the clock period, however, for different values of clock period we will observe different number of clock interrupts. If we ignore for the moment the quantization error, for the clock period $PERIOD1$ we may observe $ticks1$ clock interrupts:

$$T = ticks1 \times (PERIOD1 - OVERHEAD). \tag{5}$$

and for the clock period $PERIOD2$, we would observe $ticks2$ clock interrupts:

$$T = ticks2 \times (PERIOD2 - OVERHEAD). \tag{6}$$

From equations (5) and (6), we can determine the clock interrupt overhead:

$$OVERHEAD = \frac{ticks1 \times PERIOD1 - ticks2 \times PERIOD2}{ticks1 - ticks2}, \tag{7}$$

where we assume $PERIOD2 > PERIOD1 > 0$ and $ticks1 > ticks2 > 1$.

Equation (7) does not account for the quantization error nor for the counting error, discussed earlier. This error is accounted for by adding ± 1 to each of *ticks* variables:

$$OVERHEAD = \frac{(ticks1 \pm 1) \times PERIOD1 - (ticks2 \pm 1) \times PERIOD2}{(ticks1 \pm 1) - (ticks2 \pm 1)}. \tag{8}$$

A total of 9 different values for the value of $OVERHEAD$ can be calculated based on equation (8), each of them corresponding to one of the cases of measured and modified values of $ticks1$ and $ticks2$ in equation (8). Among the 9 values, the largest value is:

$$(9)\ OVERHEAD = \frac{(ticks1+1)\times PERIOD1 - (ticks2-1)\times PERIOD2}{(ticks1+1)-(ticks2-1)}.$$

It can be shown that (9) represents the largest of 9 values of the constant $OVERHEAD$ in (8) by starting with the inequality:

$$(10)\qquad \frac{(ticks1+a)\times PERIOD1 - (ticks2+b)\times PERIOD2}{(ticks1+a)-(ticks2+b)} \leq \frac{(ticks1+1)\times PERIOD1 - (ticks2-1)\times PERIOD2}{(ticks1+1)-(ticks2-1)},$$

where a and b are variables that take values from the set $\{-1, 0, 1\}$. Inequality (10) reduces to:

$$(11)\qquad \begin{aligned} & PERIOD1\times((b+1)\times ticks1 + (1-a)\times ticks2 + a + b) \\ \leq\ & PERIOD2\times((b+1)\times ticks1 + (1-a)\times ticks1 + a + b), \end{aligned}$$

under the assumption $PERIOD1 < PERIOD2$ and $ticks2+2 < ticks1$. Since $(1-a)$ is nonnegative, the left side in (11) is smaller, the inequality holds, and thus (9) is the largest of the 9 possible values in (8).

Error in the measurement of the constant $OVERHEAD$ gets smaller as the value of $OVERHEAD$ gets closer to $PERIOD1$, and as T gets longer, since then the difference in the numerator of (7) gets larger and thus less sensitive to the counting and quantization error (when T is getting longer and $OVERHEAD$ is closer to $PERIOD1$, more clock periods are needed to execute function *overhead* and consequently $ticks1$ is getting larger).

Once the largest value of the constant $OVERHEAD$ is determined, the constant $PERIOD$ can be selected in such a way that the effects of the constant $OVERHEAD$ are insignificant. In that case, equation (3) can be substituted by the simpler equation (1).

4. Software–based measurement error estimation example

This example describes software–based measurement of real–time kernel [1] operations' execution on a platform based on Intel 80386 EX micro–controller (33MHz, 16 bit bus).

To determine value of the constant $OVERHEAD$, two software–based measurements using the process *overhead* were made. With $PERIOD1 = 100\mu sec$ the measured $ticks1 = 147059$, and for $PERIOD2 = 1000\mu sec$ the measured $ticks2 = 11198$. The value for $PERIOD1$ was determined by trials, while

the value for $PERIOD2$ was influenced by the potential application of the real–time kernel. Based on equation (9), the value of overhead found was $OVERHEAD = 25.827487\mu sec$ (this value differs from the value based on (7) by $0.007716\mu sec$ which is insignificant). Since the determined value of the constant OVERHEAD is 2.58% of the constant $PERIOD = 1000\mu sec$, we can expect the same relative error when using equation (1) instead of (3).

During the real–time kernel operations measurement, the measured clock interrupt counts ranged from 52 to 631 ($N = 2000$). Since the counting and quantization errors in the amount of two clock interrupt counts account for less than 4% relative to the range of measured counts, we can use equation (1) instead of (2) with the same relative error.

5. Conclusion

This paper describes error estimation for a software–based measurement of execution time of real–time kernel operations. Based on the estimated error, it is possible to accurately estimate execution time of real–time kernel operations. That provides the foundation for kernel optimization (reduction in real–time kernel operations' running time), as well as for evaluation of a kernel's suitability to a real–time application domain (evaluation whether its operations can satisfy the timing constraints). The presented approach to measurement of execution time is important when an expensive equipment for exact measurement is not available.

References

[1] Hajduković, M., Concurrent Programming Using Programming Language CONCERT, author's eddition, Novi Sad, 1996.

[2] Tanenbaum, A.S., Modern Operating Systems, Prentice Hall, Englewood Clifs, NY, 1992.

Received by the editors January 5, 2002