Measuring the Latency Time of Real-Time Unix-like Operating Systems

Newton Faller

TR-92-037

June 1992

Abstract

With the advent of continuous-media applications, real-time operating systems, once confined to process control and other specialized applications, are coming to the desktop. The popularity of UNIX made this operating system the first choice for use with such real-time desktop applications. However, since UNIX kernel does not provide real-time responsiveness, some software developers have been trying to adapt it to respond to this new requirements, while others have been proposing its total redesign. Though the evaluation of the performance of a real-time operating system depends on many factors, a predictable small latency time in responding to external events is always essential. In this paper, after a discussion about the probable sources of latency, it is presented a method for collecting information about context-switching and interrupt-acknowledge times in UNIX-like operating systems without requiring external measuring tools. It is also proposed, a form of presentation of these data aimed at facilitating the comparison with previously collected data obtained from the same or from other systems. The paper is illustrated with actual results obtained by the application of the method to TROPIX, a real-time UNIX-like operating system, running on a Motorola 68010-based computer. The impact of kernel preemption and some practical measurement interference considerations due to dynamic memory refresh, DMA operation and disk multiblock access are also discussed.

This research was supported by CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brasil) and also by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., Hitachi America, Ltd., Pacific Bell, the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

Measuring the Latency Time of Real-Time Unix-like Operating Systems

Newton Faller

International Computer Science Institute
Berkeley, CA, U.S.A.
faller@icsi.berkeley.edu

Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, Brazil
faller@ufrj.bitnet or faller@rocinha.nce.ufrj.br

Abstract

With the advent of continuous-media applications, real-time operating systems, once confined to process control and other specialized applications, are coming to the desktop. The popularity of UNIX made this operating system the first choice for use with such real-time desktop applications. However, since UNIX kernel does not provide real-time responsiveness, some software developers have been trying to adapt it to respond to this new requirements, while others have been proposing its total redesign. Though the evaluation of the performance of a real-time operating system depends on many factors, a predictable small latency time in responding to external events is always essential. In this paper, after a discussion about the probable sources of latency, it is presented a method for collecting information about context-switching and interrupt-acknowledge times in UNIX-like operating systems without requiring external measuring tools. It is also proposed, a form of presentation of these data aimed at facilitating the comparison with previously collected data obtained from the same or from other systems. The paper is illustrated with actual results obtained by the application of the method to TROPIX, a real-time UNIX-like operating system, running on a Motorola 68010-based computer. The impact of kernel preemption and some practical measurement interference considerations due to dynamic memory refresh, DMA operation and disk multiblock access are also discussed.

1 - Introduction

A real-time operating system may be defined as a system such that, for a defined maximum workload, it is always capable of acknowledging a specific set of asynchronous external events and execute all the processing required to respond to them within a finite and predictable amount of time compatible with the application. Operating system latency time is the real elapsed time between the occurrence of a specific external event and the appropriate system response to it.

Complex computer-based real-time systems have been controlled by real-time operating systems. These systems, once confined to process control and other special applications, are replacing time-sharing operating systems due to the stringent requirements of continuous-media applications. In fact, in the near future, it is expect that they will become commonplace in desktop workstations.

Real-time systems have been divided in hard and soft [StRa88]. They basically differ in the way scheduling is done. While in hard real-time systems a set of tasks with their deadlines are specified, in soft real-time systems the highest priority task should be executed as fast as possible. In this paper we will be concerned with soft real-time systems only.

With the UNIX system being the most popular operating system for workstations, it is easy to understand that many software developers have been trying to fit UNIX to run appropriately with real-time applications. This has been done through adaptation of its kernel, for instance [KhSZ92] [Fual91], or even, sometimes, through the design of a new kernel [FaSa90].

There are many ways of estimating the performance of a real-time operating systems. In [Fual91], for instance, four metrics are described: the Rhealstone metric, the Process Dispatch Latency Time, the Tri-Dimensional Measure, and the Real/Stone Benchmark. In all of them, latency time has always been one of the main factors.

In this paper we suggest a method for obtaining interrupt response and context switching latency times in Unix-like operating systems requiring no external measuring tools. It is assumed, however, that the hardware has a real-time clock. The operating system is not required to be specifically designed for real-time applications but it should have some similarity to the UNIX system. A description of the procedures to be executed and the software instrumentation required for the measurements to be obtained are also described.

As an example, the method was applied to TROPIX, a real-time Unix-like operating system [FaSa90], which source code was readily available to us. The hardware used was a 10MHz one-wait-state Motorola 68010-based computer (EBC-32010) manufactured by EBC, a Brazilian computer company. Though somewhat outdated, it was chosen because it was the only available with a ported version of TROPIX, we already had all the hardware information required [EBCto87][EBCde87], and it fitted very well the purpose of illustrating the proposed methodology.

PLURIX, TROPIX and UNIX are registered trade marks of their respective owners.

The data obtained are presented in a special logarithmic-exponential percentile graph. This type of graph permits an easy visual comparison between results obtained from the same operating system when running with different generation parameters or under different pre-defined workloads, or from other operating systems when submitted to comparable workloads. This method can be used, and actually has been used, as a gauge to evaluate the improvement of system responsiveness as modifications in the TROPIX kernel are performed.

2 - The Hardware Environment

Real-time performance evaluation of operating systems can be a very elusive task if one does not have a very thorough understanding of the hardware where the operating system is running. Dynamic memory refresh, direct memory access controllers (DMA), caches in disk controllers and in the central processor, to mention just a few examples, may cause severe unexpected interference in the measured data to a point that might turn them difficult to interpret. Therefore, when using the methodology presented here, though not essential, it is rather important to have available for consulting the target computer system manuals that explain its theory of operation [EBCto87], contain its electrical diagrams [EBCed87], and describe its central processing unit with timing [MOTpr84], peripheral circuits [MOTfr90] and peripheral devices.

The methodology described in this paper, was applied, as an example, to an operating system running on a EBC-32010, a 16-bit bus computer based on a 10 MHz Motorola MC68010 microprocessor. This is a simple computer with main memory and all peripheral circuits sharing a single common bus.

Its main memory has 3 Mbytes made up of dynamic memory chips which words (16 bits) can be accessed by the microprocessor with one clock-period wait state. Since the simpler instructions take four clock periods plus one memory access to execute, and a typical instruction might require eight periods plus two memory accesses, the EBC-32010, to allow for further comparisons, can be considered an approximately 1 MIPS machine.

Direct memory access is provided through the four channels of a MC68450 Motorola DMA controller which were programmed to function in the cycle-steal mode. Each channel can also be programmed to transfer data from/to peripherals either one block at a time or multiple blocks through chaining the commands. The TROPIX operating system allows choosing the method of transfer, which has a reasonable impact in latency time.

Dynamic memory refresh is performed by the dummy continuous chained access of a block of memory through one of the DMA channels. Independent measurements showed that the refresh operation takes 11 of every 128 clock periods, which accounted for the consumption of 8.6% of the bus bandwidth. The data obtained during the measurement sessions were not compensated for this wasted refresh time and, therefore, the results presented include this overhead.

Hard disks are connected through a slave (non-disconnect) 8-bit SCSI (Small Computer Standard Interface) interface. Floppy disks have an independent controller but were inactive during the measurement sessions. Hard disks and floppy disks transfer data through different DMA channels. The hard disk connected to the system was a 85 Mbyte Seagate 296N (3600 RPM, 820 cylinders, 6 surfaces, 34 sectors of 512 bytes per track).

Serial I/O is provided by four MC68681 DUART's (Dual Asynchronous Receiver/Transmitter) making possible to have up to eight physical serial lines connected to the EBC-32010. These DUART's were programmed for 8-bit characters, no parity, one start bit and one stop bit during the measurement sessions.

A parallel port is also present in the EBC-32010 but it was inactive during the measurement sessions.

Clock interrupt is assigned to the highest maskable priority, which in the MC680X0 family is 6. An interupt of this level is generated every 10 milliseconds by a MC68230 timer. This timer actually generates this interrupt when one of its previously loaded decreasing internal counter reaches the value zero. At this point, the counter is automatically reloaded with a pre-set value which provides the appropriate generation of the next interrupt. In the EBC-32010, the oscilator that drives the processor is the same that drives the counter and the circuit was designed in a way that this counter, with some caution, despite its ripple carry characteristics, can be read by software any time. The information contained in this counter together with some additional time information stored in the kernel of the operating system, makes it possible to implement a 64-bit real-time clock with a resolution of 3.2 microseconds. This resolution was used in all measurement sessions.

3 - The Operating System Environment

The measurement methodology which is described in the following section, was applied to the TROPIX operating system. TROPIX is a fully-preemptible real-time UNIX-like operating system [FaSa90] which design was made possible through the experience gained in the design of PLURIX, a multiprocessing UNIX-like operating system [FaSa89][Faal84].

In the user level, TROPIX bears a reasonable similarity to the UNIX operating system. Processes are created through fork-execs, I/O is always treated as a sequence of bytes and is performed through open-read-write-close primitives, signals can be sent to processes, there is a kernel process zero (*swapper/pager*), the *init* process is the common ancestor of all other user processes, etc.

Internally, TROPIX kernel structure [FaSa90] is quite different from UNIX's [Bach86]. TROPIX has a fully-preemptible kernel and many specialized system calls to manipulate and coordinate the execution of real-time processes. Real-time processes coexist with their time-sharing counterparts but they can run at higher priorities and have many other privileges.

Besides its *swapper/pager*, TROPIX kernel standard processes include a unique *dispatcher* process per processor. When running in a multiprocessing environment, this scheme greatly facilitate the implementation of different scheduling strategies to be followed by different processors. Fine-grain parallel processing within executing processes is also possible since TROPIX implements threads at the supervisor level.

For the understanding of the measuring methodology described in the next section, however, there are just a few characteristics present in the TROPIX operating system, and common to most UNIX-like systems, which are important to pinpoint. These characteristics are described in the following paragraphs.

The TROPIX operating system process zero is executed once a second and starts some housekeeping operations related to main memory allocation. This process runs at the highest priority of all time-sharing processes. Real-time processes, however, run, in general, at higher priorities.

In the TROPIX operating system, besides the data structures sizes which can be dynamically allocated at boot time, there are a few parameters that can be modified at any time through special system-calls. From those, the ones, which are important for our measurement sessions, are the preemption control and the multiblock control parameters. When preemption is turned off, context switching to a new process, due to the occurence of an external event, is not executed immediately upon the occurence of the corresponding interrupt if the process is executing in supervisor mode. It is delayed until the current process willingly relinquishes the control of the processor. When preemption is on, context switching may occur immediately following the occurence of an external event, no matter in which mode the current process is presently executing. Actually, there are a very few points where preemption is always inhibited. One of them, for instance, is when the current process is already switching its context.

In the TROPIX operating system, the size of disks blocks is always 512 bytes. When multiblock is turned off, blocks are read from disk one at a time. This means that the DMA and the device controller are programmed and interrupt processing takes place each time a block is read. When multiblock is turned on, the DMA and the device controller are programmed once and many blocks can be read in one single operation. When no limiting parameter is given, an entire file can be read with a single DMA-controller programming operation. This procedure substantially improves I/O performance when requested disk blocks are physically contiguous in the media and a substantial amount of data in a disk track can be read in a single disk revolution. Multiblock transfers, however, may monopolize the bus for an unreasonable period of time and may become one of the main sources of latency. Anyway, the multiblock control parameter has influence just in block reads since block writes, except for swapping or paging, are always performed as single block operations.

4 - The Sources of Latency

In this paper, latency is defined as the delay presented by an operating system to start a processing triggered by an external event. It is the time it takes between the occurence of an external event and some action, related to this event, that the computer system is supposed to execute. Though this action is ultimately a physical action, it has been interpreted in different levels of abstraction and are treated with different priorities. Therefore, initially, it is important to understand the three levels of priority with which a computer system gives attention to external events: DMA operations, interrupt operations and process operations.

When attention to external events is extremely critical, it is essential to use DMA operations. The priority of DMA operations is the highest among all other above mentioned operations in a computer system. In addition, each channel of each DMA controller can be further prioritized to garantee that the processing required by the occurence of an external event will not be lost. The latency of a high-priority DMA operation is determined by the bus arbitration protocol, which is executed in hardware, and by the way the bus is used (reservation and release). When multiple equally important events can occur basically at the same time and must be treated without delay, it is reasonable to consider the use of a multibus architecture with multiport memories. This is actually the typical configuration of hardware designed for critical real-time applications.

One may argue that the processing of a DMA controller is too simple (read/write) to be considered as a task on a computer system. However, between a simple read/write operation that a DMA controller can perform and the complex data manipulation a full processor can execute, there is a full range of operations that a specialized processor may take advantage of using the shortest latency time a computer system can provide through direct access to its main memory. Actually, it has been reported that some high speed communication boards when connected to workstations do not rely on interrupts or user processes to perform some of their critical tasks. They manipulate data structures directly on workstation memories, for example.

When attention to external events is not extremely critical, one can rely on interrupt routines to process them. Like DMA operations, interrupt operations may be further prioritized. The latency of the highest level interrupt operation in a computer system may be influenced by three factors: instruction time execution, DMA operations, and software masking.

Since, by definition, interrupt operations can only start after the current instruction is fully executed, instructions that require a long execution time may cause an unreasonable delay to interrupt operations. There is nothing an operating system can do about this. Main processors with very long execution time instructions may save intermediate instruction status (similar to what is done by some processors when a page fault occurs in the middle of a instruction execution) and allow an interrupt operation be executed before the current instruction is completed. This procedure, however, is costly and is rarely used.

DMA operations have a higher priority over interrupt operations because DMA controllers have a higher priority in holding the memory bus than main processors. There