Mixed-mode Operating System for Real-time Performance

Hasan M. M., Sultana S., and Foo C.K.

Dept. of Computer Science
Faculty of Science
University Brunei Darussalam
Jalan Tungku Link
Gadong, BE 1410
Brunei Darussalam
Email: mdmahmud@fos.ubd.edu.bn

Mixed-mode Operating System for Real-time Performance

Abstract: *The purpose of the mixed-mode system research is to handle devices with the accuracy of real-time systems and at the same time, having all the benefits and facilities of a matured Graphic User Interface (GUI) operating system which is typically non-real-time. This mixed-mode operating system comprising of a real-time portion and a non-real-time portion was studied and implemented to identify the feasibilities and performances in practical applications (in the context of scheduled the real-time events). In this research an i8751 microcontroller-based hardware was used to measure the performance of the system in real-time-only as well as non-real-time-only configurations. The real-time portion is an 486DX-40 IBM PC system running under DOS-based real-time kernel and the non-real-time portion is a Pentium III based system running under Windows NT. It was found that mixed-mode systems performed as good as a typical real-time system and in fact, gave many additional benefits such as simplified/modular programming and load tolerance.*

1. Introduction

In the past, there has been a lot of research into real-time systems and non-real-time systems, with the clear distinction between them. Mostly, research on them were done on a separate basis. This research aims at combining these two systems to form a “mixed-mode Operating system”. Studies will be carried out on this combination to determine its performance as well as feasibility in implementation, particularly in the context of scheduled real-time events. The benefit will be the real-time performance at the same time full benefits of a mature and conventional GUI based non-real-time system’s facilities. The end result of the research is a working model that simulates a probable mixed-mode configuration and performs measurements of the system performance. One of the greatest strengths of this system is its ability to handle devices with the accuracy of real-time systems and at the same time, have all the benefits and facilities of a matured GUI operating system which is typically non-real-time. Most direct application of this system is to serving the needs in the controlling of electronic video devices such as digital VTRs (Video Tape Recorders), computer-based digital recording systems, video switchers, and many other countless devices in the video broadcasting and editing industry. Other possible applications may be in the sports events to measure participant's times, and even managing photo finishes (for example, in Olympic 100 meters dash) and/or in the manufacturing industry whereby timings of certain predefined operations are important, such as the production of certain time-sensitive chemical by reaction (photo-production), and others.

A basic conventional system consists of basically a program to carry out the tasks, and an operating system (OS) to supply the basic low-level services needed for programs such as memory management, file management, and communication [1]. Typically, such systems have no strict timing requirements and tasks can be carried out at the convenience of the system, and having lateness in performing a particular job will not result in system errors. Most popular operating systems in the market, such as Windows XP/NT, MacOS, Linux, OS/2 as well as BeOS are not real-time [2]. Such operating systems are usually built on the layering model, whereby higher-level code is built on low-level code. This makes the operating system modular, but causes heavy penalty hits. In simple terms, real-time systems are defined as systems that are capable of responding to events within a predefined amount of time, which is usually very small. Such systems are most often custom-made to the needs of a particular application. For a real-time system to work, the time taken to process the incoming information should be less than the rate at which the information is obtained. However, real-time does not merely mean that the system must be able to complete a task within a certain time limit. It must also be able to execute the task at precise moments and must never execute a certain operation too early or too late. An example is a robot arm that picks up objects from a moving conveyor belt. If the robot arm tries to pick up an object too early, there is a possibility that the object is not there. And if the robot arm tries to pick up the object too late, it would have missed the object altogether. Both conditions result in system failures.

2. Mixed-mode System

Mixed-Mode Systems, a term created for use in this research, consists of a combination of Real-Time System and a Non-Real-Time System. This combination exists so that a particular application is able to control and respond to hardware devices in real-time whilst having all the facilities of a mature OS such as Windows NT which is well suited for real-time operations. However, such systems are rarely suited for immediate response to external events since the round trip from the device to the Real-Time System to the Non-Real-Time System and back takes up too much time. Mixed-mode systems are usually used for scheduled real-time events, or when the real-time system is able to handle some of the immediate responses without conferring to the non-real-time system. Performance measurements on such systems usually focus on the performance between the real-time portion with the hardware itself. Less emphasis is put on the performance between the real-time portion and the non-real-time portion.

3. Methodology

The system setup basically consists of three units, which are the ***Scheduler Unit*** (to generate schedule of events), the ***Executor Unit*** (to execute events in real-time), and the ***Profiler Unit*** (whose task is to generate a common time base as well as measure performance), as shown in the Figure 1. The Scheduler is a Windows NT based machine, the Executor runs on MS-DOS with a RTK (Real-Time Kernel) to emulate a RTOS (Real-Time Operating System), and The Profiler is a microcontroller-based unit based on the i87C51 chip.

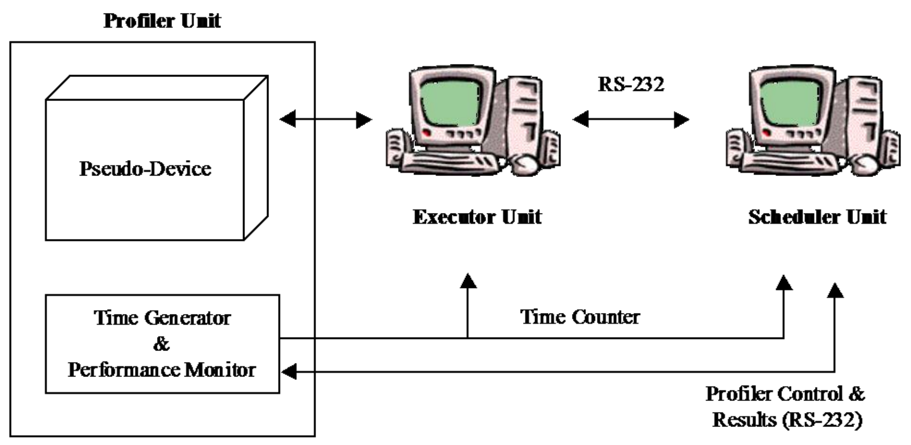

Profiler Unit
Pseudo-Device
Time Generator & Performance Monitor
RS-232
Executor Unit
Scheduler Unit
Time Counter
Profiler Control & Results (RS-232)

Figure 1 - Mixed-Mode System Architecture

Asynchronous serial connections based on RS-232 communication is used for Executor-Scheduler and Profiler-Scheduler communication as such communication is more predictable based on the line baudrate setting than using the protocols based TCP/IP sockets. The Profiler generates timecodes which are 8-bit binary numbers. These numbers are sent to both Executor and Scheduler as a common time base to which event are scheduled. To ensure the lowest possible latency, reading of these binary values are done using a special Digital I/O expansion card called a GPI card.

3.1 Study Method

Having the Profiler as a uniform method of measuring latency (performance), the first task is to verify that the Executor portion is truly real-time. This is done by having the Executor schedule and execute to its own set of events, while the Profiler measures the latency. Latency of *1ms* or below is considered sufficiently real-time for this research project.

In the next step, comparison needs to be done against a mixed-mode system and a conventional system. Therefore, the first part of the study experiments would be conducting latency measurements on the conventional system model which schedules and executes its own events.

Finally, the mixed-mode configuration is tested, and the results are compared to that of the previous two tests. In this test, each unit performs their respective function. It is expected that the mean measured latency of the mixed-mode system will be significantly lower than the measured latency of the conventional system by several folds, and in fact, having (or at least, approaching) real-time performance.

4. Schedular Unit

The Scheduler in this research is used mainly to create and schedule events to be executed on the Executor. It also “executes” events (which is the task of the Executor), purely for comparison purposes. The Scheduler is a PC-based system running Windows NT, a non-real-time operating system by Microsoft.

4.1 Hardware Architecture

The Scheduler is a Pentium-III based system equipped with 1GB SDRAM with two built-in 9-pin serial ports, both free for use. One of these serial ports is connected to the Executor, while the other is connected to the Profiler, via null-modem cable [3]. All communication via serial ports are done at 9600bps, no parity, 8 data bits, 1 stop bit. Besides that, an 8-bit ISA GPI card is installed inside the system to enable it to directly read the 8-bit timecodes from the Profiler [4].

4.2 Software Architecture

The Scheduler was written using a combination of Visual Basic 5.0 and Visual C++ 5.0. Visual Basic was used for its Rapid Application Development (RAD) properties [5]. Visual C++ was used for interfacing. A DLL (Dynamic Link Library), callable from Visual Basic, was made. This DLL contained all these low-level features which Visual Basic lacked, such as a timecode-extension thread (to extend 8-bit timecodes to 32-bit by monitoring for wraparounds) and I/O port access [6]. The Schedular Software Architecture respect to component and task has been shown in the Table 1. And the Figure 2 and 3 shows the Scheduler Software Architecture and Scheduler Screenshot respectively.

Component	Task
Event Schedule List	Hold the list of scheduled events, as well as their measurement results.
Timecode Timer	Update timecode display as well as “execute” events in “Normal mode” operation.
Scheduling Timer	Periodically create event schedule.
Serial Communication	Perform RS232 messaging with Executor & Profiler.
Direct I/O unit	Give the VB program access to I/O ports
Timecode Thread	Extend the 8-bit timecode into more usable 32-bit values

Table 1 - The Schedular unit’s Software Architecture respect to component and task.

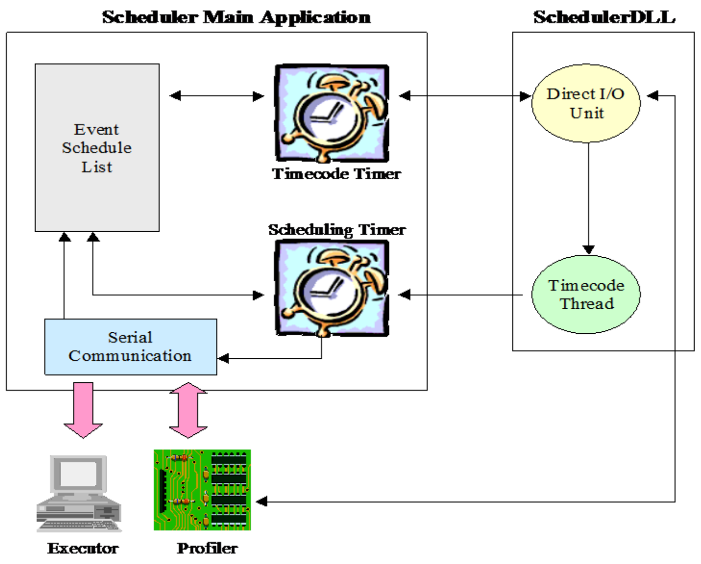
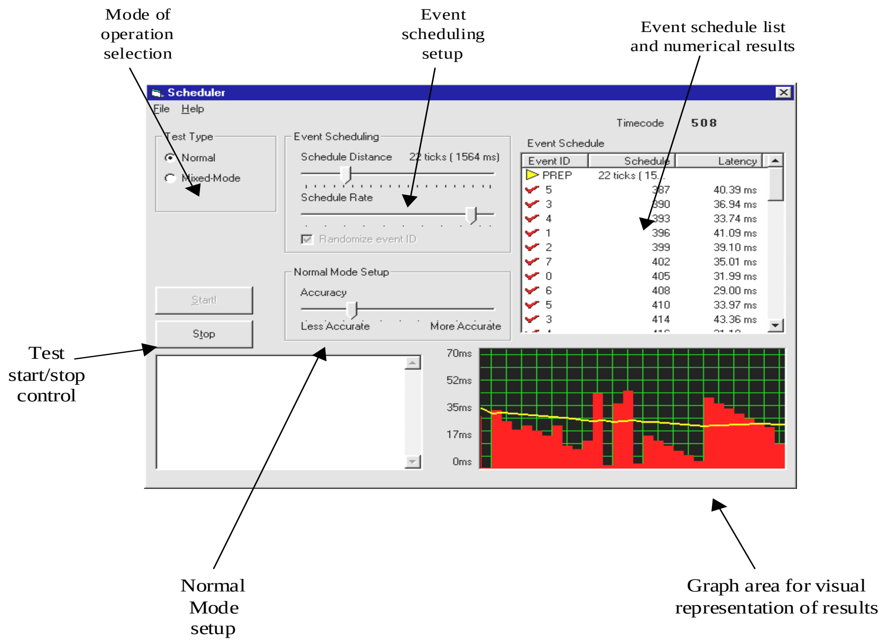


Figure 2 - Scheduler Software Architecture


Mode of operation selection
Event scheduling setup
Event schedule list and numerical results
Scheduler
File
Help
Start!
Stop
70ms
52ms
35ms
17ms
0ms
Test start/stop control
Normal Mode setup
Graph area for visual representation of results

Timecode 508

Test Type: [✓ Normal] Mixed-Mode

Event Scheduling

Schedule Distance 22 ticks (1564 ms)

Schedule Rate

[✓ Randomize event ID]

Normal Mode Setup

Accuracy: Less Accurate — More Accurate

Event Schedule

Event ID	Schedule	Latency
PREP	22 ticks (15...	
5	387	40.39 ms
3	390	36.94 ms
4	393	33.74 ms
1	396	41.09 ms
2	399	39.10 ms
7	402	35.01 ms
0	405	31.99 ms
6	408	29.00 ms
5	410	33.97 ms
3	414	43.36 ms

Figure 3 - Scheduler Screenshot

There are two basic modes of operation for the Scheduler. Firstly is the "Mixed Mode" operation. This is the main mode of interest in this research. In this mode, each of the 3 units of the system (Profiler, Executor and Scheduler) performs their respective task. Secondly is the "Normal mode" of operation. In this mode, the system does away with the need for the Executor, and instead, responds to events directly. This mode is present for comparison purposes against the "Mixed Mode" operation. Results can be viewed on screen as well as saved to disk in Comma Separated Values (CSV) format (loadable from Excel). The "Schedule Distance" slider determines "how far ahead" the event's scheduled timecode should be. The "Schedule Rate" slider determines how much time will elapse between two events that are scheduled. "Randomize Event ID" checkbox determines whether events should be scheduled in sequential order or randomized order. "Accuracy" slider works in normal mode by determining how much processor power should be dedicated to event execution.

5. Executor Unit

The Executor is the real-time portion of this system. Its task is basically to "execute" scheduled events accurately in real-time. Schedule of events are obtained from the Scheduler unit. The Executor comprises of the following components

- An IBM PC-AT compatible machine (AMD 80486DX 40MHz).
- A GPI card to read in time counter and send out signals for the events to the Profiler.
- A 9-pin serial port to receive schedule of events from Scheduler.
- A Real-Time-Kernel (RTK) written in Assembly & C (as a simple RTOS replacement).
- An "Executor" program to handle the events, running on top of the RTOS and written using C++.

5.1 Hardware Architecture

A 80486 system at 40MHz, equipped with 8MB of SIMM RAM was used. It runs MS-DOS as its primary operating system. It has one built-in 9-pin serial port free for use. This serial port is connected to the Scheduler via null-modem cable to receive event schedules from it [7]. All communication via serial ports are done at 9600bps, no parity, 8 data bits, 1 stop bit. A 486-based system was chosen because it relatively cheap and easy to source. Furthermore, hardware and software support for this "scheduled based real-time event" architecture is aplenty. An 8-bit ISA GPI card is installed inside the system to enable it to directly read time-code from the Profiler as well as output "responses" to the Profiler unit.

5.2 Software Architecture

Being a study of mixed-mode system, a RTOS was naturally required for this research. A small real-time multitasking kernel (RTK) was written using a combination of assembly language and C++. This kernel runs on top of MS-DOS and gives the system prioritized multithreading capability. It must be noted, however, that RTK is merely a real-time multitasking kernel, not an OS by itself. It relies on MS-DOS to perform basic functionalities such as file access and memory management (QNX Software Systems) [8]. However, MS-DOS is not designed to be multi-threaded, therefore, there are certain limitations to what can be done with it [9]. The Table 2 and Figure 4 shows the Executor software architecture.

Component	**Task**
Event Schedule List	Hold the list of scheduled events, as well as their measurement results.
GPI interface	Perform I/O access with the GPI card.
Timecode Thread	Extend the 8-bit timecodes into 32-bit ones as well as execute events. Runs at highest priority.
Serial Communication Thread	Perform the reception of the event from Scheduler.
Serial Communication Library	Perform direct I/O serial communication with the Scheduler.

Table 2 - The Executor unit's Software Architecture respect to component and task.

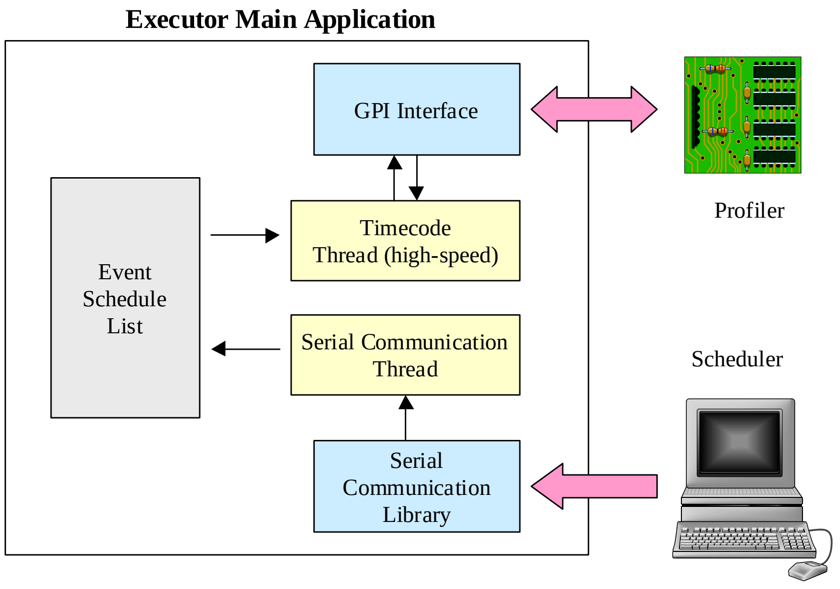
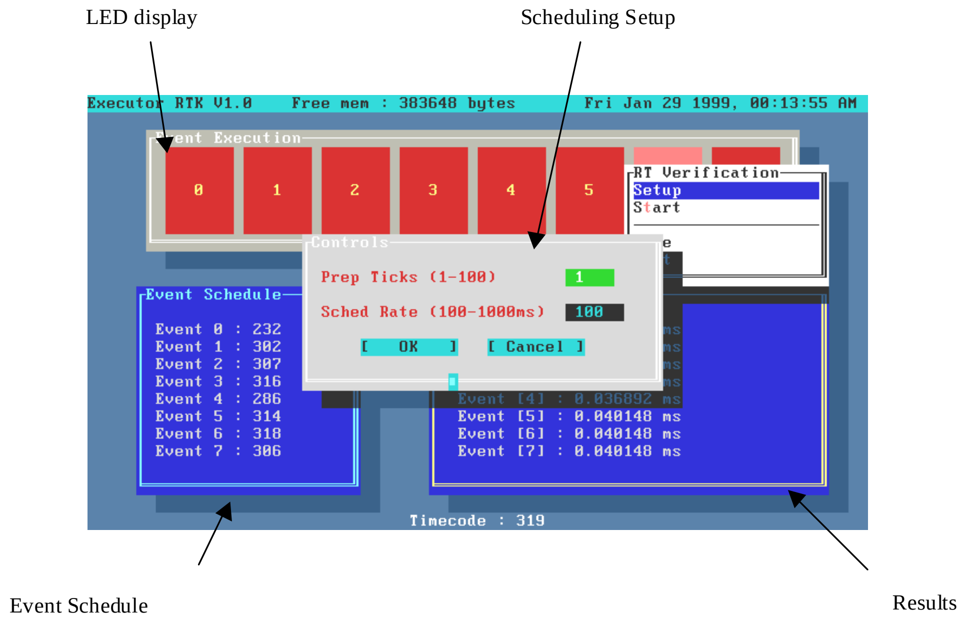

Executor Main Application
GPI Interface
Event Schedule List
Timecode Thread (high-speed)
Serial Communication Thread
Serial Communication Library
Profiler
Scheduler

Figure 4 - Executor Software Architecture

The application itself is a GUI-based application (shown in the Figure 5) . The top portion of the screen contains large "LEDs" to display the events the Executor is executing. The blue window on the left is the event schedule list, whereas the blue window on the right (partially covered) is the results of the measurements. Test parameters can be modified by changing the input fields in the gray color window.

There are two basic modes of operation for the Executor. The first is the Real-Time Verification mode. In this mode, the performance of the Executor is verified to be real-time or not. In this mode, the Executor schedules and executes its own set of events. Results can be saved in CSV format (loadable under Excel). The second mode is the actual mixed-mode configuration. In this mode, the Executor does not create events. Instead, the Scheduler will be having this task, and the Executor will merely Execute the scheduled events at precise moments in time.



```
Executor RTK V1.0     Free mem : 383648 bytes     Fri Jan 29 1999, 00:13:55 AM

Prep Ticks (1-100)          1
Sched Rate (100-1000ms)     100

Event 0 : 232
Event 1 : 302
Event 2 : 307
Event 3 : 316
Event 4 : 286
Event 5 : 314
Event 6 : 318
Event 7 : 306

Event [4] : 0.036892 ms
Event [5] : 0.040148 ms
Event [6] : 0.040148 ms
Event [7] : 0.040148 ms

Timecode : 319
```

Figure 5 - Executor Screenshot

6. Profiler Unit

The Profiler is a microcontroller-based system developed for the specific purpose of measuring the latency of the mixed-mode system in this system, which relates to the overall system performance, as well as to generate a common time base for all event schedules [10]. Being a unit of its own independent of both Executor and Scheduler, performance measurements are accurate and unbiased [11].

6.1 Hardware Architecture

The Profiler is powered by an Intel 87C51 microcontroller with 4KB EPROM and 128 bytes of RAM, and runs at 11.0592MHz clock rate. A quartz crystal of this frequency was chosen because it gave the best settings for 9600bps serial communication shown in the Figure 6.

The 87C51 was chosen because it contained built-in USART capability, many I/O ports, as well as built-in high-precision timers useful for performance measurement. An ICL232 serial transceiver was used to convert TTL voltages to RS-232 voltage levels (vice versa). This IC is essentially pin compatible with the more popular MAX232 by Maxim. A 10-segment BAR LED was also used as a simple display to ascertain the workings of the microcontroller. Of the10, only 8 of the LEDs were used, with each LED corresponding to one event ID (or one “virtual device”). An 8-bit octal buffer (74HC244) is used to drive these LEDs.

Port 1 is used for timecode output. The output of this port is an 8-bit binary up counter which increments at a rate of 14.0625 times per second (derived by dividing clock rate of 11.0592MHz by 12 and by 65536), which is the 16-bit timer overflow rate. Executor’s response is received from Port 0, whereby the lower 3-bits of the port denotes the event ID. Port 2 is used for LED outputs to light up the LED corresponding to the event being executed by the Executor. Serial communication within the Profiler performs no flow control whatsoever as there are only two pins available (Rx and Tx) for serial communication.

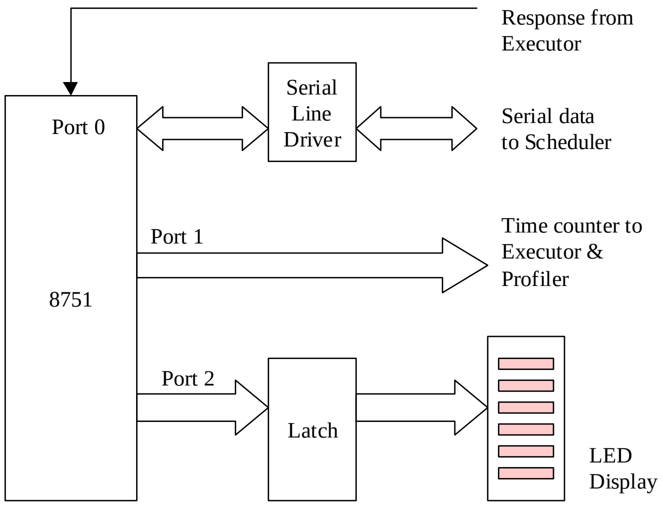


Figure 6 - Profiler Hardware Architecture

6.2 Software Architecture

The Profiler software can be broken up into foreground and background tasks. The foreground task polls Port 0 to monitor for any changes (execution of events) to measure latency. For the background tasks outputs timecodes (14.0625 times per second) and also performs the serial I/O communication. The Figure 7 shows the Profiler architecture.

Event schedules are stored in a 16-byte buffer, each element being 2-bytes scheduled timecode. The measurement results are stored in a 32-byte circular buffer which the serial interrupt clears upon transmission. During startup, the Profiler performs a simple LED POST (Power-On Self Test) that runs through all the LEDs one by one before initializing the microcontroller.

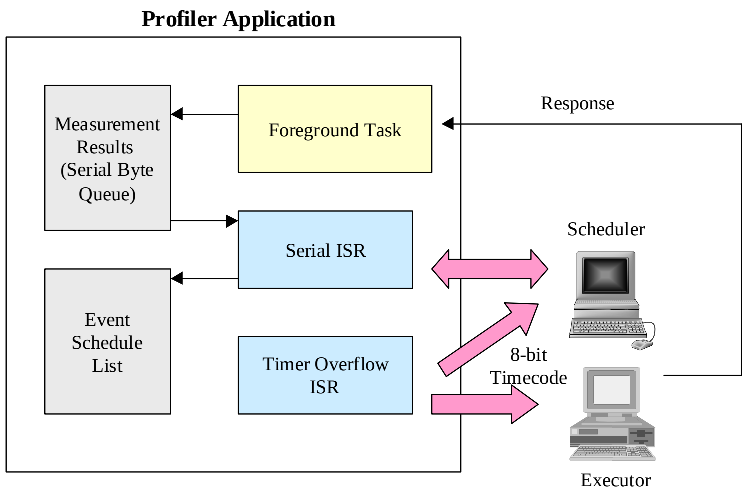


Figure 7 – Profiler Architecture

7. RESULT & DISCUSSION

7.1 RTK performance

Full context switch takes 14.245 μs, which is a pretty good number, and in fact, on par, if not, higher than QNX's performance. Removing FPU context switching (useful when not more than a single thread uses floating-point code), RTK managed to perform context switch took in only 6.41 μs. Interrupt latency was not measured for the system, as RTK does not implement soft interrupts and all interrupts are as fast as the hardware permits.

7.2 Executor-Only Performance (Real-Time Performance Verification)

Result shows in the Figure 8, that it is impossible for the Executor to respond to immediate events (as expected). This holds true for any similar real-time systems and cannot be avoided. However, system performs well above expectation of 1ms when handling scheduled events. Average latency is 0.0391ms with variance of 0.004809.

Spikes are due to execution of GUI threads which may temporarily lock up system. (This is not a good thread to have in actual applications).

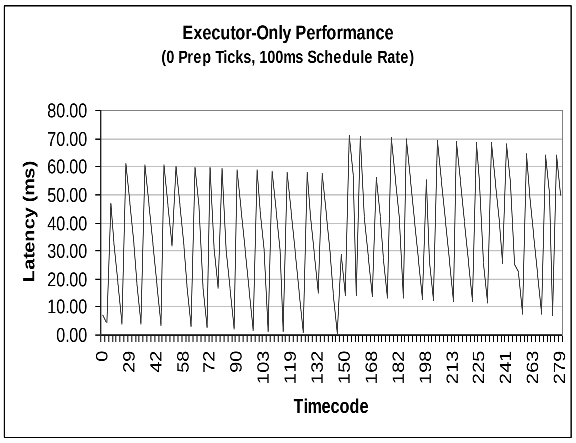


Figure 8 – Executor performance, immediate event (0 tick ahead)

7.3 Scheduler-Only Performance (Conventional System Test)

As can be seen, the scheduler performs badly with scheduled events. The average latency is 14.465ms with a large variance of 36.52 ms. This is clearly unacceptable for this system's deterministic 1ms requirement. The results are the same if events are scheduled further ahead (> 1 tick), as shown in the Figure 9.

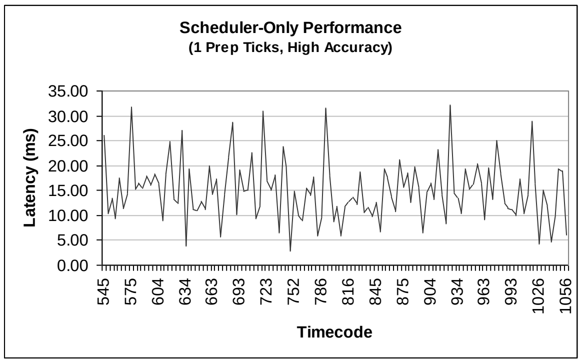


Figure 9 – Scheduler performance, Scheduled event (1 tick ahead)

7.4 Mixed-Mode Performance

It is expected that mixed-mode system's performance should be quite good. However, with events scheduled 1 tick ahead, the results are quite unacceptable even though half of the readings were below 1ms as shown in the Figure 10. This is due to the propagation delay of messages from the Scheduler to the Executor and Profiler (as this delay itself is sometimes over 71ms long).

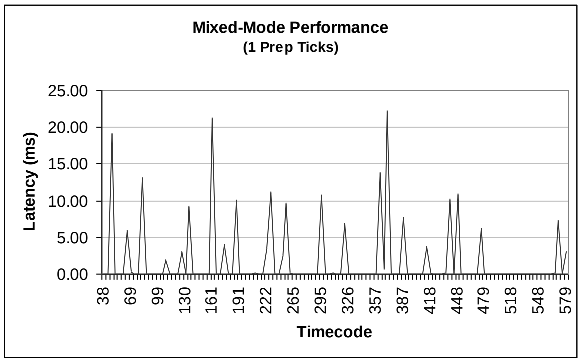


Figure 10 – Mixed-mode performance, 1 tick ahead

Extending the schedule-ahead time to 2 ticks (Figure 11) yield much better real-time results well within the system's requirements. Occasional spikes still exists. Mean latency is 0.0430ms and variance is 0.000378.

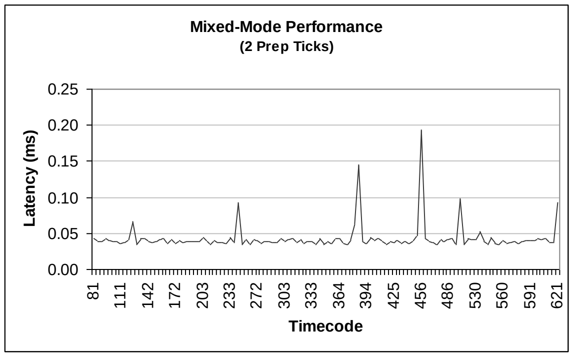


Figure 11 – Mixed-mode performance, 2 ticks ahead

7.5 Impact of Non-Real-Time System on Overall Performance

Under heavy load on Scheduler, the mixed-mode system configuration performs still exceptionally well, as shown in the Figure 12. Mean latency is 0.04703ms with variance of 0.001035. Heavy loads were simulated by running many applications concurrently, as well as performing background disk access.

7.6 Significance of Schedule Time on the Mixed-Mode Performance

Having a mixed-mode system involves a “middle-man” in managing the actual hardware introduces message propogation delay. Therefore, to be truly effective, the mixed-mode system should not handle immediate events, but rather, only scheduled events. The value of the propagation delay depends on the transmission medium. With baudrate 9600 bps (line baudrate) the RS-232 gives pretty much deterministic values.

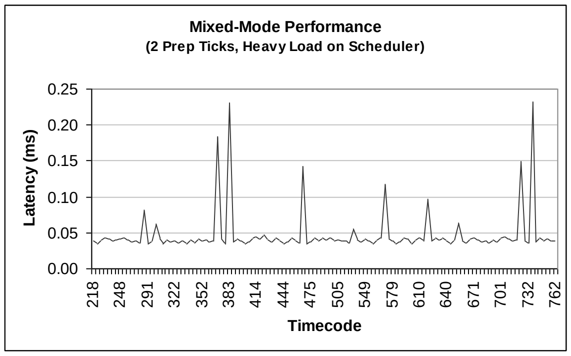


Figure 12 - Mixed-mode performance, with heavy load on Scheduler

7.7 Advantages of Mixed-Mode Systems

Firstly, real-time performance is guaranteed from the mixed-mode system, even during heavy loads on the main application. Secondly, it is probably cheaper to have two normal machines to handle specific tasks rather than one fast super-machine to handle everything. Thirdly, the system, being broken up into two distinct applications with well-defined tasks makes the code easier to write.

8. Conclusion

Mixed-Mode operating system should be used more widely since it gives more benefits than limitations. It observed that a well-designed mixed mode system is capable to delivery in the real-time performance with all the benefits and facilities of a matured Graphic User Interface (GUI) operating system. Which is an unique advantage of this system. Most of the measurement results in the system can be, in fact, improved. It is found that mixed-mode system provides one of the best balance of real-time performance and OS features. It also makes programming simpler and easier to manage as the system is broken down and separated into distinct functional groups. The executor / RTK may undergo the following changes to further improve its performance.

- Implement soft-interrupts (so that spikes would not exist)
- Increase scheduling rate
- Use an optimizing compiler
- Avoid unnecessary threads
- Use a faster processor

Refferences

[1] CAXTON C. FOSTER (1981), *Real Time Programming - Neglected Topics*, Addison-Wesley Publishing Company.

[2] CMX COMPANY (1998), *CMX 80x86 RTOS Performance Sheet*, http://www.cmx.com/trget27.htm

[3] HARRY GARLAND (1979), *Introduction to Microprocessor System Design*, McGraw-Hill Book Company.

[4] INTEL CORPORATION (1983), *Embedded Applications Volume 1*, Intel Literature, ISBN 1-55512-242-6

[5] DAVE WILLIAMS (1993), The Programmer Technical Reference For MS-DOS And IBM PC, *Tech Publications Pte Ltd*.

[6] PETER NORTON AND HAROLD AND PHYLLIS DAVIS (1995), *Peter Norton's Guide To Visual Basic 4 For Windows 95*, Sams Publishing.

[7] JAMES L. ANTONAKOS (1990), The 68000 Microprocessor - Hardware and Software Principles and Applications, *Merril Pubslishing Company*.

[8] QNX SOFTWARE SYSTEMS LTD. (1998), *QNX News*, Vol. 11 No. 3.

[9] S.T. ALLWORTH AND R.N. ZOBEL (1988) *Introduction to Real-time Software Design (2nd Ed.)*, Macmillan Education.

[10] LEO CHIN SIM, HEIKO SCHRODER and GRAHAM LEEDHAM, "MIND-SIMD hybrid system – towards a new law cost parallel system", *Parallel Computing 29, 2003, 21-36,* Elsevier.

[11] A. SINGH, K. JEFFAY, "Co-Scheduling Variable Execution Time Requirement Real-time Tasks and Non-Real-Time Task", *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 2007, pages 191-200.