A TOOL FOR PERFORMANCE EVALUATION OF REAL-TIME UNIX OPERATING SYSTEMS

B. Furht, A. Boujarwah, D. Gluch, D. Joseph, D. Kamath,
P. Matthews, M. McCarty, R. Stoehr, and R. Sureswaran
Modular Computer Systems, Inc.
Fort Lauderdale, Florida 33309

ABSTRACT

In this paper we present the REAL/STONE Real-Time Tester, a tool for performance evaluation of real-time UNIX operating systems. The REAL/STONE Real-Time Tester is a synthetic benchmark that simulates a typical real-time environment. The tool performs typical real-time operations, such as: (a) reads data from an external source and accesses it periodically, (b) processes data through a number of real-time processes, and © displays the final data. This study can help users in selecting the most-effective real-time UNIX operating system for a given application.

Key words: Real-time UNIX operating systems, Real-time benchmark

INTRODUCTION

Rating a computer's performance and comparing one computer with another has always been difficult. Conventional methods of rating only processor performance do not give a true picture of overall system performance. Therefore, processor performance is not an adequate measure of total system performance, especially in the arena of real-time computing.

The best possible benchmark for determining the performance of a new design is the actual application which will ultimately run on the target system. This is normally not possible. The next best solution is a "synthetic" benchmark which mimics the statistical distributions of instructions and events that will be encountered in the ultimate target application, and thereby simulates the production environment. This is also sometimes difficult since such statistical data is rarely available, and since target applications often change from the time of conception to the time of implementation.

In this article we use the REAL/STONE Real-Time Tester benchmark, which simulates a typical real-time environment, in order to evaluate several popular real-time UNIX implementations. By exercising critical components of real-time computers (such as context switching, process synchronization and communication, and file I/O) the Real-Time Tester provides objective data which allows users to make realistic assumptions about how a particular UNIX-based system will perform in a real-time environment.

The Real-Time Tester has been applied to a number of real-time and non-real-time UNIX implementations, and the results are presented in [1]. The Real-Time Tester can also be ported to other non-UNIX operating systems, such as the DEC VMS operating system [2], and others.

REAL-TIME UNIX OPERATING SYSTEMS

The UNIX operating system has become a standard operating system gaining rapid acceptance because of its superior flexibility, portability, and large number of support tools. However, the UNIX operating system was originally designed for multitasking and time-sharing, and therefore the standard UNIX operating system does not have an adequate response time and data throughput needed to support most real-time applications. Several studies have been done on the application of UNIX-based systems in telemetry [3, 4, 5].

Many attempts have been made to adapt the UNIX kernel to provide a real-time environment [6, 7, 8, 91. Given the diversity of the approaches in that each implementation is dependent on the application code and type of processor, it is very difficult to compare performance among all the versions of real-time UNIX operating systems.

One classification method [8] divides real-time UNIX implementations into six categories. These six categories, along with the companies taking these approaches, are summarized in Table 1.

1. Adding Extensions to Standard UNIX System

This approach to implementing a real-time UNIX system adds extensions to the core UNIX operating system. The real-time extensions can be implemented in the kernel or outside of the kernel. This is the approach taken by some system-level vendors, such as AT&T in its System V.4 UNIX release. Some examples of adding extensions to create a real-time UNIX system can involve implementing priority-based scheduling in a task scheduler, and implementing real-time timers and priority disk scheduling. Although this approach provides some real-time functionality, it has severe drawbacks. The main drawback is that this approach does not allow full preemption in the kernel mode. When an application task makes a system service call, and the task goes into kernel mode, the system has to complete that service call before a higher priority task can gain access to the CPU.

Table 1
Classification of Real-Time UNIX Implementations

TECHNIQUES	OPERATING SYSTEM & COMPANY	
1. Adding extensions to standard UNIX	• AT&T System V.4	
2. Host-target approach	• VxWorks™	(Wind River Systems)
	• OS-9®	(Microware)
	• VRTX™	(Ready Systems)
3. Integrated UNIX and real-time executives (or real-time OS)	• MTOS-UX™	(IPI)
	• RTUX™	(Emerge Systems, Inc.)
	• CXOS™	(Computer X/Motorola)
	• D-NIX®	(Diab Systems)
4. Proprietary UNIX	• AIX®	(IBM®)
	• LynxOS™	(Lynx Real Time Systems)
	• Regulus™	(SBE)
5. Preemption points	• RTU™	(Concurrent/Masscomp)
	• HP-UX	(Hewlett Packard)
	• VENIX™	(VenturCom)
6. Fully preemptive kernel	• REAL/IX™	(MODCOMP®)
	• CX/RT	(Harris)

2. Host/Target Approach

The host/target approach toward developing a real-time UNIX environment is to develop an application on a UNIX host and then download it for execution on a proprietary real-time kernel or a dedicated operating system running on a target system. This approach connects a real-time kernel to a UNIX host via a communications interface. For example, in the case of the VxWorks operating system, developers use the UNIX host for development. This includes editing, compiling, linking, and storing real-time applications. Then the program is tested, debugged and executed on the target machine running VxWorks. The host and target communicate via sockets and the TCP/IP protocol.

3. Integrated UNIX Environment and Real-time Executive/OS

This approach provides a UNIX interface to a proprietary real-time kernel or a proprietary real-time operating system. Both the UNIX system and the proprietary kernel/OS run on the same machine. The MTOS real-time kernel, for example, can log onto a UNIX system. The user can utilize the development tools in the UNIX environment in the development of real-time applications, and then run those applications in the proprietary operating system environment. This approach, similar to the host/target approach, provides the fast real-time response of the proprietary real-time systems, but requires that programmers understand two operating systems. The porting of applications is therefore considerably more difficult

4. Proprietary UNIX-like Operating System

This approach to real-time UNIX implementation consists of developing the real-time kernel from the ground up while, at the same time, retaining standard UNIX interfaces. Since a complete rewriting of the software is required, this approach is referred to as being proprietary. Their internal implementations are proprietary, but the interfaces are fully compatible with a standard UNIX operating system such as AT&T's System V operating system. These interfaces are specified with standards such as SVID and IEEE POSIX. This approach provides relatively high-performance, however porting applications to or from another real-time UNIX implementation often requires rewriting code. Standard UNIX applications will run under these operating systems, but code using the real-time extensions generally must be rewritten.

5. Preemption points

Another real-time UNIX implementation approach is to insert preemption points, or windows into the operating system kernel. Preemption points are built into the kernel, so that system calls do not have to block or run to completion before giving up control. This can reduce the delay before the higher-priority process can begin or resume execution. However, as an impact of preemption points, there is still a preemption delay which may be as high as several milliseconds. This delay corresponds to the longest period of time between preemption points. The preemption points approach is taken by the RTU operating system, HP-UX operating system, and VENIX™ operating system. The RTU operating system includes approximately 100 preemption points and 10 preemptible regions. At the preemption points, the operating system performs a quick check to see if a real-time process is ready to run. At these preemption points a preemption region is initiated in the kernel wherein the scheduler is always enabled. The preemption points approach provides a high degree of determinism and better than normal response times, but it is still not a fully preemptive system. The drawback is similar to the drawback of the adding extensions approach previously described: true real-time performance is not achieved.

6. Fully preemptive kernel

This approach provides for full preemption in the UNIX environment, wherein preemption can occur anywhere in the kernel. The preemptible kernel can be built by incorporating synchronization mechanisms, such as semaphores and spin locks, to protect all UNIX global data structures. By implementing a fine granularity of semaphores, the preemption delay can be significantly reduced to 100 microseconds or less. A fully preemptive kernel provides the system with the ability to respond immediately to interrupts, to break out of the kernel mode, and to execute a high-priority real-time process. This approach is implemented in the REAL/IX and CX/RT operating systems. VENIX and RTU versions of fully preemptive UNIX kernels are under development

THE REAL/STONE REAL-TIME TESTER

The REAL/STONE Real-Time Tester is a benchmark and demonstration system that enables users to compare and evaluate real-time computer systems. By exercising critical components of real-time computers, the Real-Time Tester provides objective data which allows users to make realistic assumptions about how a particular computer system will perform in the user's real-time application, and so helps users to determine the most effective real-time computer system for their application.

The Real-Time Tester simulates a typical real-time environment by:

- Reading data from an external source
- Processing the data through a user-specified number of real-time processes
- Displaying the processed data

The data transferred by the Real-Time Tester is an image consisting of thousands of pixels. While the benchmark is running, the monitor displays progress through screen refreshes, each of which adds pixels to the image. The computer system with the best deterministic real-time performance displays the image fastest without any image distortions or missing pixels. The demonstration provides even non-technical users with an easy-to-understand comparison of each tested system's real-time performance.

The Real-Time Tester, the structure of which is shown in Figure 1, consists of a master process, which runs a selected real-time priority, and a number of lower priority real-time processes that are generated by the master process. The real-time processes serve as a pipeline for the data being acquired and displayed.

The pipeline consists of a head process, a tail process, and the real-time processes. The user selects the number of real-time processes in the pipeline. The logical flow of the Real-Time Tester is as follows:

1. The head process reads the data from an external source (a file or a device) and places the data into an input buffer in a shared memory region.

2. The data is then transferred through the process pipeline. The head process activates the next process in the pipeline (P1) through the use of semaphores, then P1 activates P2, etc. Each transfer of control from one process to another causes a full real-time context switch.

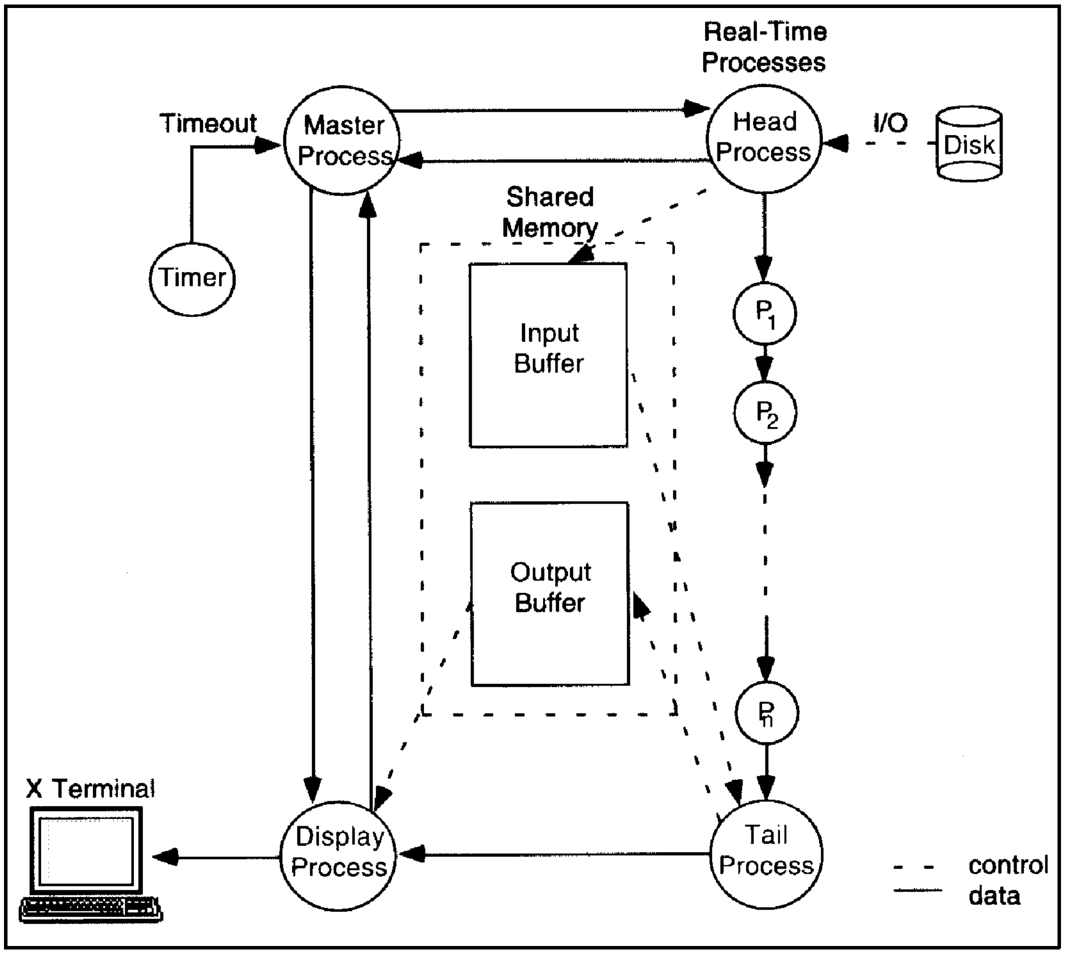


Figure 1 - The structure of the REAL/STONE Real-Time Tester benchmark

3. The last process in the pipeline, the tail process, transfers the data from the input buffer to the output buffer, and then displays the data on an X terminal or on a character-based terminal if an X terminal is unavailable.

For each block of data, two cycles must be executed by the computer. The first cycle transfers the block of data through the pipeline of tasks. This transfer must occur within a deadline time limit, which is controlled by the internal timer. If the whole block is not transferred within the deadline time, then the remaining data in the block is lost, simulating what occurs in real-world applications. The second cycle displays the data (image) that was successfully transferred through the pipeline. These two cycles are performed continuously until all data is transferred and the image is complete.

The Real-Time Tester examines the computer system's real-time performance in the following areas:

- Context switch time
- Interprocess communication and synchronization mechanisms, such as semaphores and shared memory
- File I/O system
- UNIX system calls

The user-friendly interface is based on the X Window System™ and the OSF/Motif™ graphical user interface (see Figure 2). The menu-driven system consists of the following user-selectable fields:

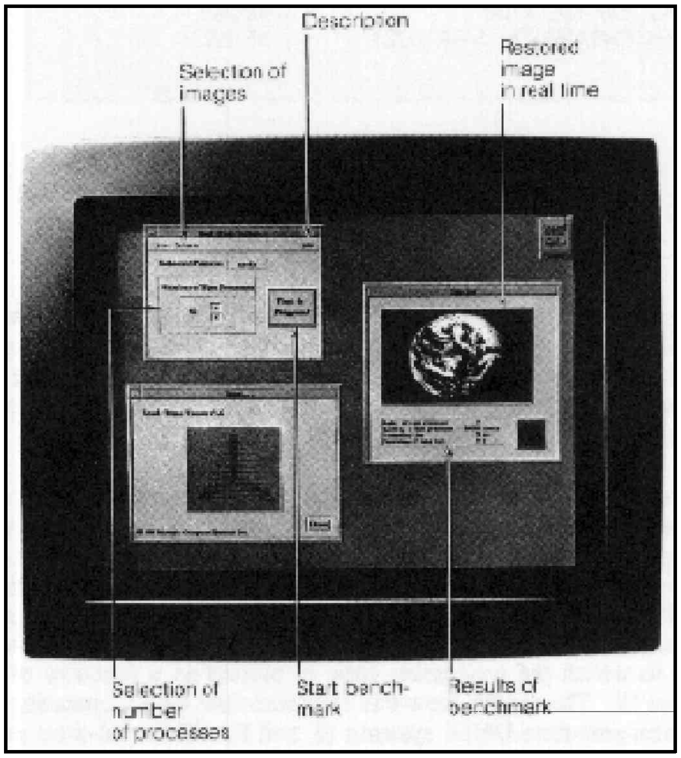


Figure 2 - User interface

- Description - Allows the user to display or print the description and structure of the Real-Time Tester.

- Select process - Allows the user to select the number of real-time processes in the pipeline (1 to 12) to simulate different process loads.

- Select image - Allows the user to select the image to be displayed during the demonstration. The images are different sizes and represent different real-time applications.

- Start and stop - Allows the user to initiate or interrupt the execution of the Real-Time Tester.

Completion of the benchmark results in an image being generated on the X terminal. Better real-time computers (with faster response time and higher data throughput) will generate the image in shorter times.

As the user increases the number of real-time processes in the pipeline, the computer's workload increases and image generation slows. Computers with poor real-time performance will lose data because they are unable to transfer each block within the specified deadline time. Consequently, the image displayed by these systems shows blank spots which represent lost data.

At the end of execution, the displayed image gives a visual measurement of the tested system's real-time performance. For additional quantitative comparisons, the Real-Time Tester automatically calculates and displays a summary of results, as shown in Figure 3.

```
NUMBER OF PROCESSES              5
QUANTITY OF DATA PROCESSED       120,256 PIXELS
PROCESSING TIME                  45 SEC
PERCENTAGE OF DATA LOST          15.2%
```

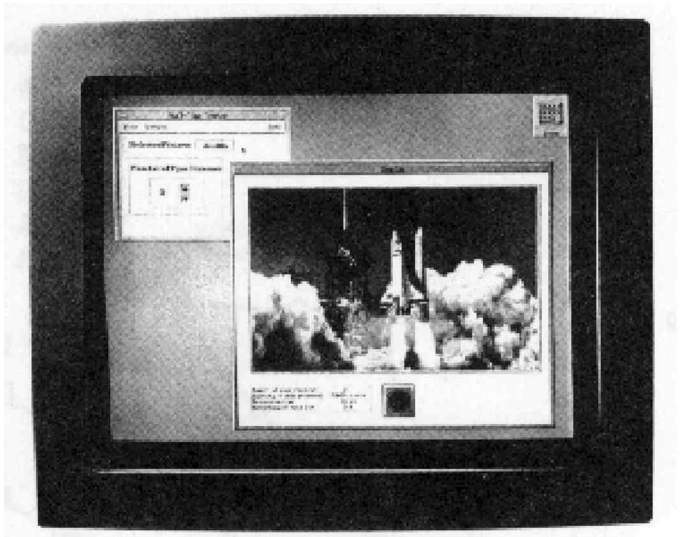
Figure 3 - Sample summary of results

RESULTS

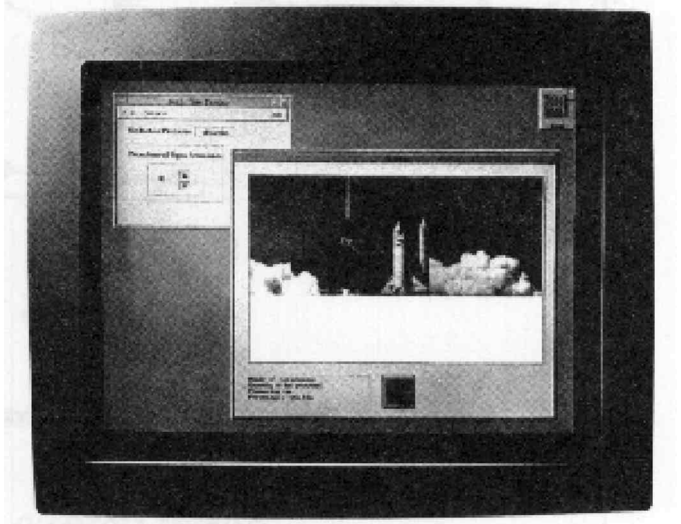
The results presented here are provided to demonstrate the capabilities of the Real-Time Tester and are not intended as a competitive evaluation of the performance of various systems. In Figure 4, visual measurements of three tested systems' real-time performance are shown. System A provides fastest response, while System B is slower than System A, however, it is generating the image without distortion. System C is the slowest system and it displays a distorted picture due to lost data.

For an additional quantitative comparison, the Real-Time Tester has been applied to four real-time UNIX operating systems and two non-real-time UNIX operating systems. These results are based upon operating systems A, B, D, and E executing on hardware platforms which used a 25 MHZ Motorola 68030 processor. Operating System C was executing on a platform which used a 33 MHZ Motorola 68030 processor and Operating System F was executing on a platform which used a 20 MHZ Motorola 68020 processor. A total of 1,500 data characters were transferred and processed in each test. The number of real-time processes in the pipeline was varied from 1 to 12. A sample of results is shown in Figure 5 in which the processing time is plotted as a function of the number of real-time processes over the range of 1 to 12. The results show that four real-time UNIX operating systems (A, B, C, and D) perform better than the two non-real-time UNIX systems (E and F). This

System A provides the fastest response, completing the image fastest and without distortion.



System B is slower than System A; however, System B, like System A, is generating the image without distortions.



System C is the slowest system and it displays a distorted picture due to lost data.

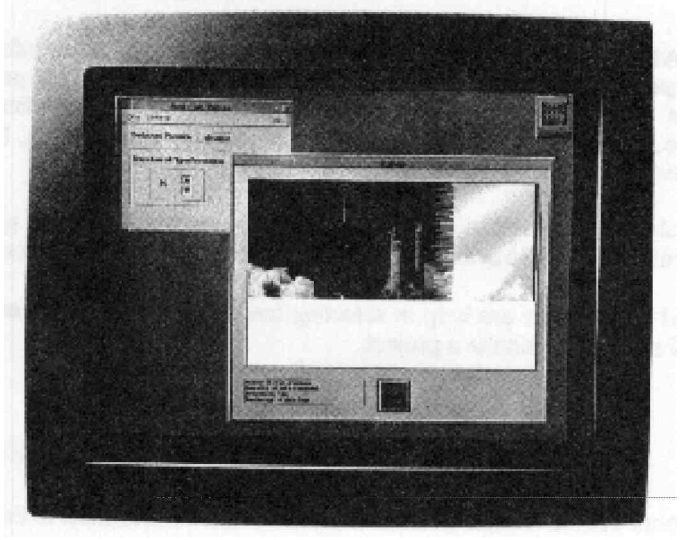


Figure 4 - Demonstration of visual measurements for three real-time UNIX systems

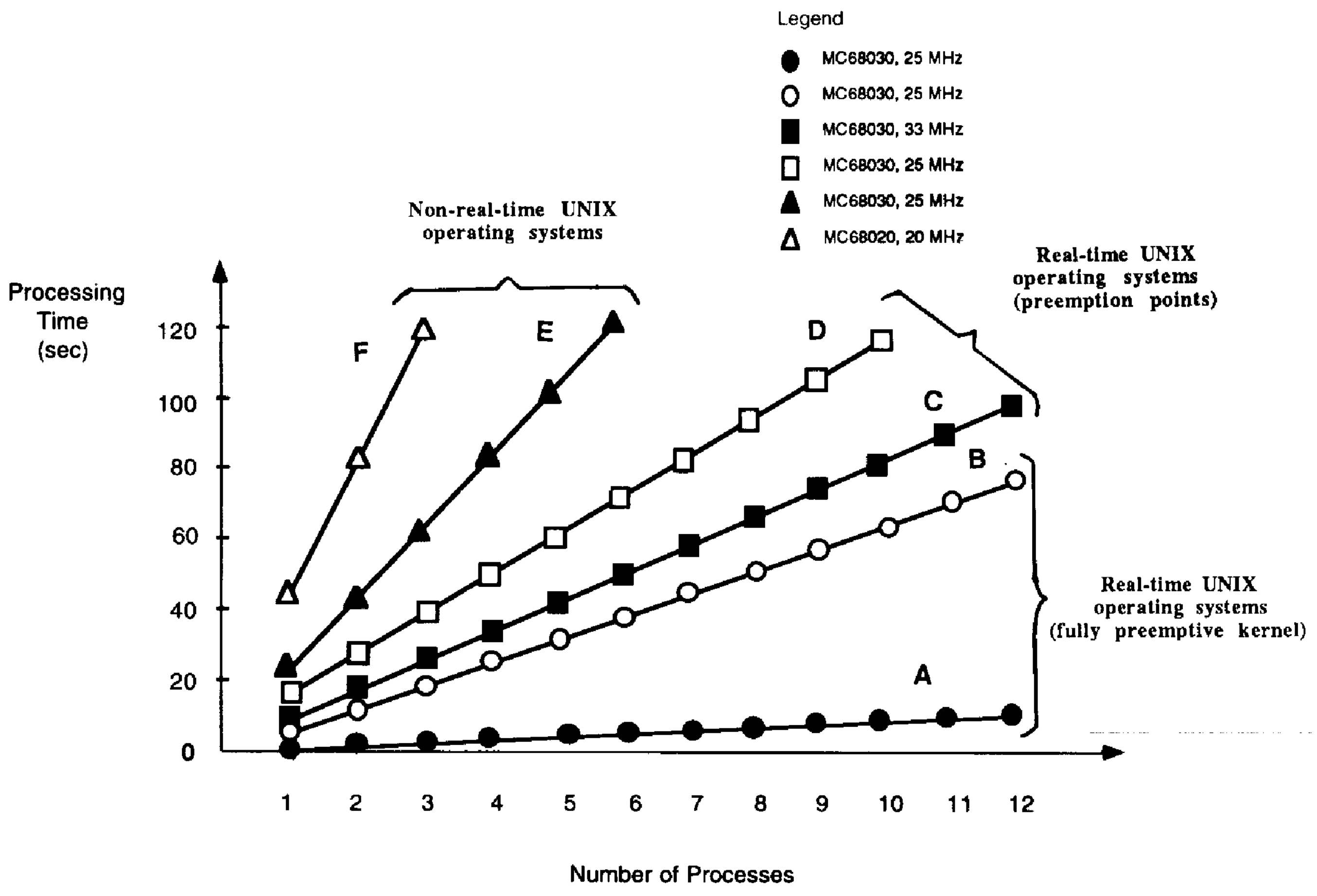


Figure 5 - Performance of UNIX operating systems

real-time performance superiority of real-time UNIX systems (even recognizing the slower hardware platform of System F) is attributable to their preemptive kernels, fast context switch, fast interprocess communication facilities, and other real-time features. However, it should be pointed out that there are significant performance differences between real-time UNIX implementations in that the real-time UNIX systems with a fully preemptive kernel (A, B) generally performed better than those which use preemption windows [C, D]. In addition, System A, which uses ultra fast interprocess communication facilities, performed better than any of the other systems tested.

CONCLUSIONS

The REAL/STONE Real-Time Tester provides substantially more information about total system performance than general-purpose benchmarks, which typically only measure CPU performance, floating-point performance, and compiler maturity. Features and capabilities tested by this benchmark include: operating system features for real-time, operating system interprocess calling and semaphores, timer scheduling features, real-time library capabilities, and disk I/O subsystem.

The Real-Time Tester can easily be ported to any open system, regardless of the system's supplier, CPU, or configuration. It is also available for some proprietary real-time systems.

The Real-Time Tester can help in selecting the right computer system for any real-time application, which may be the most critical decision in a project.

ACKNOWLEDGEMENTS

The authors wish to thank Ms. Terri Stettner for her contribution to the production of this manuscript

REFERENCES

1. "Real-Time Tester", MODCOMP publication, Fort Lauderdale, Florida, November 1990.

2. "REAL/STONE Real-Time Tester for DEC Machines Running DEC/VMS™ Operating System", MODCOMP publication, Fort Lauderdale, Florida, March 1991.

3. J. J. Pfeiffer, Jr., "The UNIX Operating System for Telemetry Applications", Proc. of the International Telemetry Conference, Las Vegas, Nevada, October 1990, pp. 381-388.

4. S. B. Crabtree and B. J. Feather, "Real-Time UNIX Telemetry Support System", Proc. of the International Telemetry Conference, Las Vegas, Nevada, October 1990, pp. 389-398.

5. R. Querido, "UNIX and Real-Time Telemetry", Proc. of the International Telemetry Conference, Las Vegas, Nevada, October 1990, pp. 731-738.

6. H. Falk, "Developers Target UNIX and ADA with Real-Time Kernels," Computer Design, April 1, 1988, pp. 55-70.

7. D. Simpson, "Will the Real-Time UNIX Please Stand-up", System Integration, December 1989, pp. 46-52.

8. B. Furht, D. Grostick, D. Gluch, J. Parker, G. Rabbat, and M. McRoberts, "Real-Time UNIX Systems: Design and Application Guide", Kluwer Academic Publishers, Boston, 1991.

9. P. Matthews, "The Merger of UNIX and Real-Time", MODCOMP publication, Fort Lauderdale, Florida, March 1990.