REAL-TIME COMPUTER SYSTEMS: CURRENT AND FUTURE TRENDS
plenary presentation

Borko Furht

Department of Computer Science and Engineering
Florida Atlantic University, Boca Raton, Florida 33431

ABSTRACT

In this paper we present the state-of-the art of real-time computing systems by emphasizing those developments likely to have impact on future real-time systems. The real-time systems based on industry standards are introduced, and the future trends in operating systems development are discussed.

1. OVERVIEW OF REAL-TIME APPLICATIONS

Response time and data throughput requirements of a real-time system depend on the specific real-time applications. For example, in an energy management system, the main function is to monitor and control environmental factors such as temperature and air flow. The computational requirements for this application are modest because of the relatively slow sampling rates and slow overall response of the mechanical devices affecting changes within the system. In an electrical power plant, however, data acquisition and calculation become more critical. Plant equipment and system functions must be closely monitored and controlled on a continuous basis to ensure safe operation and prevent costly unscheduled outages and shutdowns. A slight deviation from optimal performance even for a short period of time can significantly impact the cost of electrical energy produced by the plant. Figure 1 illustrates various response time requirements for several typical real-time applications.

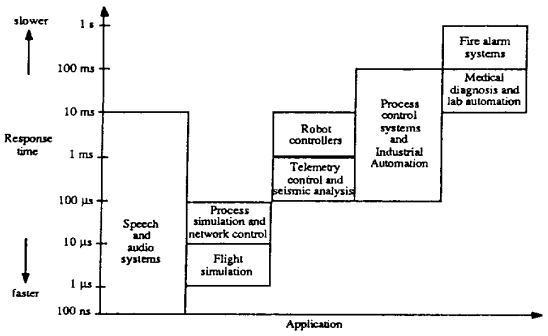

Figure 1 - Response-time requirements for real-time applications

2. THE TRANSITION FROM PROPRIETARY TO OPEN REAL-TIME SYSTEMS

Traditionally, real-time computing has been a realm of proprietary systems. Applications written for these systems in high level languages often require assembly level code and depend on the unique characteristics of the proprietary operating system, hardware architecture, and instruction set. With the escalating cost of software development and the massive conversion efforts required for porting real-time applications to state-of-the-art hardware, there is a need for highly portable real-time applications. Therefore, the next generation of real-time systems will be based on open systems, which incorporate industry standards. Open systems reduce system cost and time to market, increase availability of software packages, increase ease-of-use, and facilitate system integration.

Open real-time computer concept is based on hardware architectures which use off-the-shelf standard micro-processors, standard real-time operating systems, standard communication protocols, and standard interface buses.

In planning the development of new real-time computer systems, in addition to functionality, there are two key elements which determine the success of the product: (1) cost, and (2) timely availability of the product. Figure 2 shows the curve integration level versus design time for four design strategies: custom design, use of standard logic, use of standard boards with value added, and finally, use of standard boards with no value added.

Proprietary systems are typically based on custom designs and their development requires a minimum of three years. We foresee that the winning strategy in designing modern real-time systems is a strategy based on standard boards and adding values in real-time. For embedded real-time systems, a strategy based on standard boards with no value added can be the winning one.

Using the inverse pyramid model, shown in Figure 3, the contemporary architecture of an open real-time system based on industry standards consists of "open" components and "value-added" components. At the present stage, the computer developers have begun developing open real-time systems using standard components, however, there are still many custom elements, that make these systems

0-8186-4130-4/93 $03.00 © 1993 IEEE

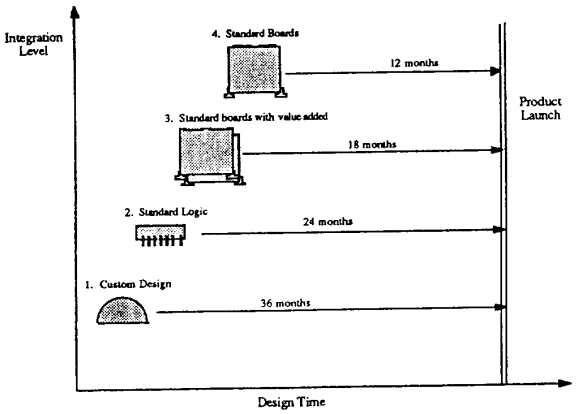

Figure 2. Strategies in designing real-time computers: integration level versus design time

hybrid. We foresee that in the near future a typical real-time system will be based on industry standards and consisting of off-the-shelf standard components. However, there will still be space to add value in real-time, in order to produce higher-performance system. This will be a major challenge for real-time system developers. We also expect further emphasis on software, and we may see real-time computer developers incorporating application tools into their products, thus providing more integrated solutions.

In summary, the trends in designing modern real-time computing systems can be characterized with the following concluding remarks: (1) time to market rather than performance, (2) software solutions rather than hardware, (3) computer vendors will ultimately become system integrators, and (4) value-added to nice real-time markets.

3. REAL-TIME OPERATING SYSTEMS

In the arena of real-time operating systems, we are currently witnessing the battle of three distinct approaches: (a) real-time executives for embedded real-time systems, (b) a full-blown standard (UNIX) real-time operating system for more complex centralized real-time applications, and (c) distributed real-time operating systems for distributed real-time applications. The spectrum of real-time operating systems from these three categories is presented in Figure 4.

We foresee that the UNIX operating system will soon become the standard operating system for centralized complex real-time applications. A number of attempts have been made to adapt the UNIX kernel to provide real-time environments [1]. Also, a number of attempts have been made to standardize the user level interfaces of the UNIX operating system. The two major are AT&T's System V Interface Definition (SVID) and the IEEE committee 1003 (POSIX). The POSIX 1003.4 is a a real-time standard with the purpose of developing a set of interfaces that allow portability of applications with real-time requirements. The POSIX 1003.4a is the the extension of this standards which incorporates using threads in real-time.

Due to its complexity and some inherent drawbacks, a real-time UNIX cannot be yet used in embedded real-time applications. Figure 5 illustrates a typical response time and size of typical real-time UNIX systems versus real-time executives and micro kernels. However, real-time UNIX developers are trying to build small UNIX kernels with a very fast response, which can be used in embedded real-time applications. This issue is discussed in Section 5.

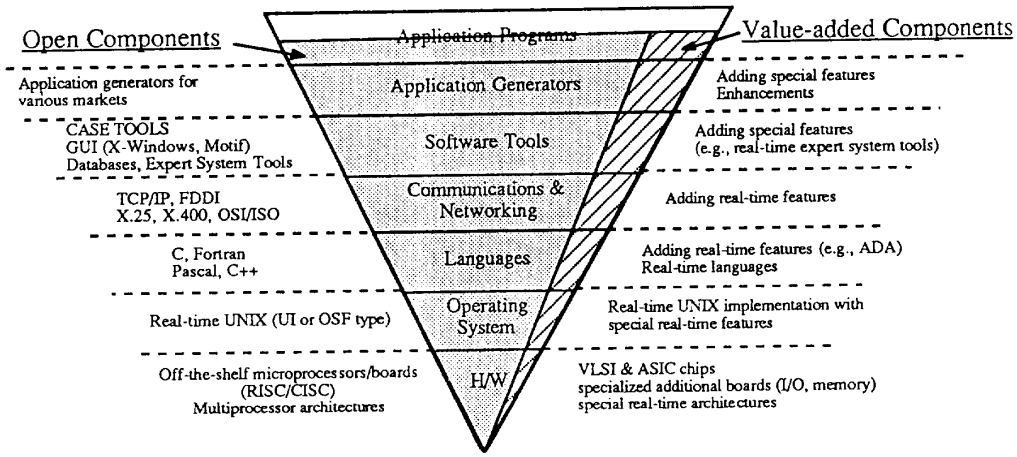

Figure 3. The architecture of a real-time system based on industry standards

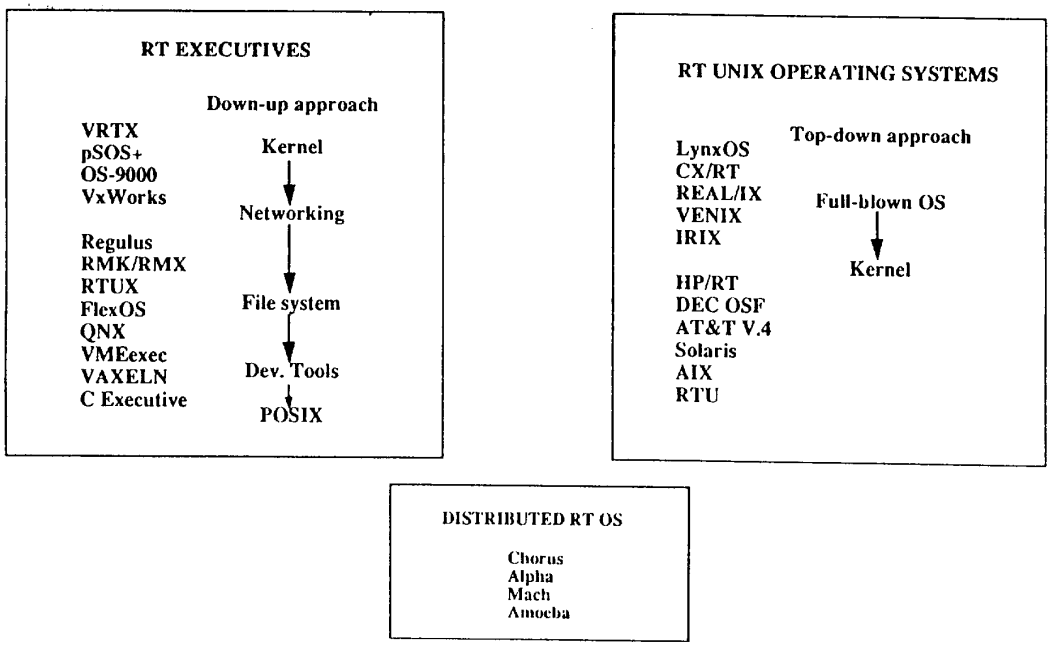

Figure 4. Real-time OS/Executives/Distributed systems: Which way to go?

Today, there is a clear difference in real-time UNIX operating systems and real-time executives, as illustrated in Table 1.

TABLE 1: Real-Time Executive vs Real-Time UNIX

RT Executive	RT UNIX
Scalable	Less scalable
Faster response	Slower response
Micro kernel/kernel	Kernel
Kernel does not include a file system	Kernel includes a file system
Custom dev. tools	UNIX dev. tools
Host/target development	Host development
No standards	Standards: POSIX, SVID
Poor GUI	Powerful GUI (X, Motif)
Loosely-coupled multiprocessing	Tightly-coupled multiprocessing

We discuss the future trends in developing RT executives and RT UNIX systems in Sections 4 and 5.

4. REAL-TIME UNIX IMPLEMENTATIONS

One classification method, that we propose, divides real-time UNIX implementations into six categories. These six categories, along with the companies taking these approaches, are summarized in Table 2. These six approaches are briefly discussed next.

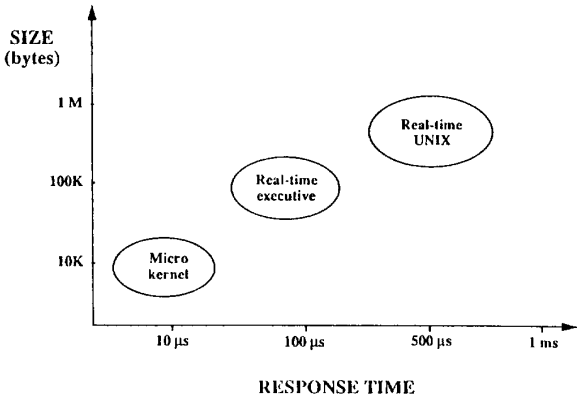

Figure 5. Real-time operating systems: response time versus size

Adding extensions to the standard UNIX operating system. Some extensions are added to the core UNIX operating system, such as priority-based scheduling, real-time timers, and priority disk scheduling in AT&T V.4. This approach does not provide a full preemption in the kernel mode.

Host/target approach. This approach consists of developing an application on a UNIX host and then downloading it to a proprietary real-time kernel on a target system. It combines the advantages of the UNIX operating system with those of a proprietary real-time kernel.

Table 2. Classification of Real-Time UNIX Implementations

TECHNIQUES	OPERATING SYSTEM & COMPANY	
1. Adding extension to the standard UNIX operating system	• AT&T System V.4	
2. Host-target approach	• VxWorks • OS/9 • VRTX	(Wind River Systems) (Microware) (Ready Systems)
3. Integrated UNIX and real-time executives (or real-time OS)	• MTOS-UX • RTUX • CSOX • D-NIX	(IPI) (Emerge Systems) (Computer X/Motorola) (Diab Systems)
4. Proprietary UNIX operating system	• LynxOS • Regulus	(Lynx Real Time Systems) (Alcyon)
5. Preemption points	• RTU • HP-UX • VENIX	(Concurrent/Masscomp) (Hewlett Packard) (VenturCom)
6. Fully preemptive kernel	• REAL/IX • CX/RT • HP-RT • Solaris 2.0 • AIX • DEC OSF/1 • IRIX	(MODCOMP) (Harris) (Hewlett-Packard) (SUN) (IBM) (DEC) (Silicon Graphics)

However, this approach requires two operating systems and porting of application software to other platforms is difficult.

Integrated UNIX environment and real-time executive. This approach provides a UNIX interface to a proprietary real-time kernel, both running on the same machine. This approach provides the fast response of the proprietary real-time system, however it requires two operating systems, and porting of applications is difficult.

Proprietary UNIX operating system. This approach consists of developing a proprietary real-time kernel from the ground up while maintaining the standard UNIX interfaces. The internal implementation of these systems is proprietary, however the interfaces are fully compatible with UNIX operating systems, based on POSIX standards.

Preemption points in kernel. This approach is to insert preemption points to a traditional UNIX 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 preemption time, however there is still a preemption delay which may be as high as several milliseconds.

Fully preemptive kernel. A fully preemptive real-time UNIX operating system allows full preemption anywhere in either the user or the kernel level. The preemptible kernel can be built by incorporating synchronization mechanisms, such as semaphores and spin locks, to protect all global data structures. It provides the system with the ability to respond immediately to asynchronous events, to break out of the kernel mode, and to execute a high-priority real-time process.

Table 3 summarizes the key contrasts between a traditional and a fully preemptive high-performance real-time UNIX and identifies the implementation issues involved in the development of a real-time UNIX kernel.

Table 3. Contrast Between Traditional and Real-Time UNIX Operating Systems

Traditional UNIX	Real-Time UNIX	Implementation Issues
Non-preemptive kernel	Full kernel preemption	Protection of global data and code
Time-sharing scheduler	Fixed priority time-sharing scheduler	Extend scheduler functions
Lengthy interrupt latency times	Guaranteed interrupt response time	Optimize code and minimize interrupt disable times
Basic interprocess communication	High performance reliable IPC	Optimize traditional and include enhanced mechanisms
Basic file I/O	High-performance deterministic file I/O	Pre-allocate and fast file system files
Basic I/O support	Asynchronous I/O	Extend I/O interface options
Minimal resource control	Memory, CPU capabilities and I/O control and preallocation	Incorporate user control facilities into the kernel

5. NEXT GENERATION OF REAL-TIME OPERATING SYSTEMS

Two major driving forces in developing the next generation of RT operating systems are: (1) the real-time standards such as IEEE POSIX 1003.4 and 1003.4a, and (2) big players, such as IBM, DEC, HP and SUN have entered into real-time UNIX battle by developing their real-time UNIX versions.

The first phase of RT UNIX development (1987-91) is characterized by the fact that small- and average-size companies (Concurrent, Harris, Modcomp, Lynx) have developed the first real-time UNIX versions. In the second phase (1992-present), big companies have developed their RT UNIX versions, as well. We foresee that in the next few years these big players will dictate the future development in RT UNIX.

Another trend is in the development of scalable real-time operating system architectures. RT executives have

Another trend is in the development of scalable real-time operating system architectures. RT executives have applied "bottom up" approach by developing first a micro kernel, then adding additional functions the micro kernel has been expended to a kernel, and later on the top of the kernel the file system, networking, and various tools have been added. The next natural step would be adding the POSIX interfaces on the top of RT executives, which will transform them into POSIX-like operating systems.

In the RT UNIX arena, the applied approach can be characterized as "top down". First, from the full-blown UNIX, real-time versions with POSIX interfaces have been developed. Currently, there are several attempts to make the RT UNIX scalable (LynxOS, HP/RT), as illustrated in Figure 6.

In summary, we foresee that there will be less and less differences between RT executives and RT UNIX operating systems. Referring to Table 1, RT executives are trying to add features where a full-blown RT UNIX has been superior in the past (such as standard interfaces, tools, and GUI). On the other hand, RT UNIX systems are trying to become faster, more scalable, and of a smaller size, so they can be used in embedded applications as well. Very soon, both systems RT executives and RT UNIX operating systems may look alike, and will be characterized by a common scalable real-time operating system architecture, shown in Figure 7.

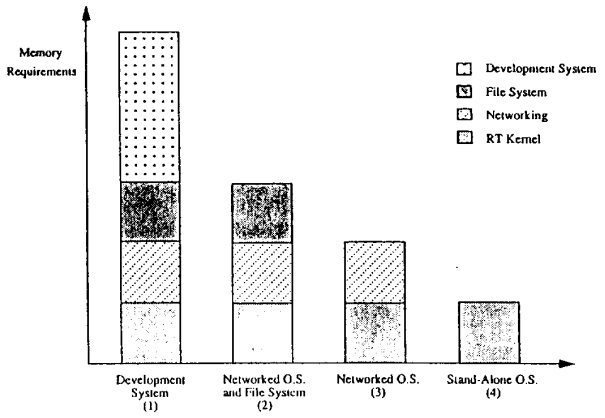

Figure 6. Scalable real-time UNIX OS

6. CONCLUSION

Real-time systems, based on standards, present a new direction in real-time computing. Most real-time UNIX operating systems already comply with POSIX standards, and soon we can expect that RT executives begin complying with them. Besides complying with standards, the next generation of real-time operating systems will use scalable architectures allowing users to tailor the system according to their needs. All three approaches, based on RT executives, RT UNIX operating systems, and distributed real-time OS, will coexist. The winners will be those who will provide scalable architectures with the highest real-time performance.

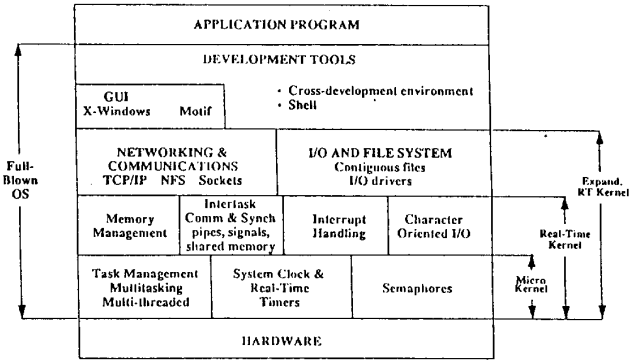

Figure 7. Scalable RT operating system architecture