

TOWARD A REAL-TIME EXECUTIVE STANDARD

A new standard could enhance the use of UNIX for developing real-time applications

By Dick Vanderlin



Vanderlin is manager of real-time software development for Motorola's microcomputer division (Tempe, Ariz.). He is responsible for developing and supporting real-time operating systems for that division's VMEbus board and system products.

One thing the real-time application software developer does early on, when a new application comes along, is to pick a suitable real-time executive, otherwise known as a real-time operating system kernel. But all the industry executive/kernel offerings present different software interfaces to the outside world. So the knowledge of one interface does little good when another is needed; application porting to new executives is inefficient, and application software portability is minimal. In these days of open systems architectures, this situation is unacceptable.

A solution to the problem is to define a real-time executive/kernel interface standard, much as the IEEE's POSIX effort and AT&T's SVID work define UNIX interfaces. Such an effort would greatly benefit the UNIX community because it would allow close, efficient coupling between UNIX systems and real-time executives. A real-time executive interface standard would allow, for example, the many benefits of the UNIX development system to be applied to real-time software development.

A standardization effort open to all parties interested in real-time executive software, the Real-Time Executive Interface Definition (RTEID) was developed by Motorola with technical input from Software Components Group. It will facilitate the writing of applications programs that are directly portable across multiple real-time executive implementations. RTEID has been submitted to the VMEbus International Trade Association (VITA) for consideration as a standard.

The RTEID defines a standard interface for the development of real-time software. This interface includes source-code interfaces and run-time behavior as seen by an application program. It does not include the details of how a kernel implements these

functions. The RTEID serves as a complete definition of external interfaces so that application code that conforms to these interfaces will execute properly in all real-time executive environments.

A real-time standard interface would have the same strong user advantages that the UNIX standardization effort enjoys; nevertheless, little has been done until now to address the needs of the real-time community for such a standard. What is the reason for this neglect that stands in sharp contrast to the SVID and POSIX efforts for UNIX standardization?

THE QUESTION ANSWERED

The development of real-time executives is a relatively new software industry, with most of the popular executives having been introduced within the last six years. Thus standardization efforts have not had a lot of time to develop.

However, with ever more companies opting for off-the-shelf software rather than in-house designs, it's now time to define the interface to the critical functionality in a real-time system's real-time executive. Furthermore, with the push toward UNIX becoming standardized, a new market environment for UNIX real-time applications has emerged.

Today, among other reasons, UNIX is not used in real-time applications because of the nondeterminacy of the UNIX kernel. One way to move UNIX applications to a real-time UNIX environment is with a SVID-to-RTEID interface library. Applications that are SVID-compliant and run on UNIX may be ported easily to a real-time system with the addition of this interface.

Of course, not all UNIX applications are candidates for real-time execution. But certainly many real-time applications could take advantage of the portability of code developed for a SVID-

to-RTEID environment. For example, applications such as communications servers (that may have marginal performance on UNIX) may achieve acceptable performance running on a separate processor in an integrated UNIX real-time environment.

THE BEST CHOICE

There are many advantages for application developers who adhere to legally defined standards, de facto standards, and even popularly accepted standards wherever possible. For example, by

using a popular operating system like UNIX for software development, the developer has access to a wide variety of tools to enhance the capability of the development environment.

In addition, by using a widely accepted language, the developer has access to a number of compilers, optimizers, and documentation and training aids. All this is well and good for applications developed and running on UNIX. But what about the real-time environment?

Since there are no standards, each new real-time software development project must be ported to a new kernel environment (if for some reason a new kernel has been specified). Clearly, the opportunity is available for a standardization effort that would allow the development of real-time kernel routines that are completely portable between application projects regardless of the underlying kernel.

The idea of the RTEID is simple. With the use of the RTEID, routines that acquire memory segments, create and manage message queues, establish and use semaphores, and send and receive signals need not be redeveloped for a different real-time environment as long as the new environment is RTEID-compliant.

The programmer need only concentrate on the hardware dependencies of the real-time system. And most hardware dependencies for real-time applications can be localized to the device drivers.

An RTEID-based software design will have major advantages in the overall software project development cycle. There are still other advantages in the testing phase of a project. Since the re-used code has been fully tested and debugged, it saves time in testing.

INSIDE THE RTEID

Table 1 identifies the RTEID functionality. For example, with the RTEID's set of task-management functions (directives), it's possible to implement a wide variety of real-time applications that involve manipulating tasks. What's more important, there is considerable flexibility available to the software developer concerned with task management.

For example, using `t_setpri` and `t_mode`, the real-time programmer has sufficient latitude to change the real-time characteristics of an entire system. In most real-time systems, it is common to have several tasks competing for processor time, depending on the relative importance or priority given to each task.

continued

TABLE 1: RTEID functionality

Name	Input Parameters				
<code>t_create</code>	name	superstk	userstk	priority	flags
<code>t_ident</code>	name	node			
<code>t_start</code>	tid	mode	saddr		
<code>t_delete</code>	tid				
<code>t_suspend</code>	tid				
<code>t_resume</code>	tid				
<code>t_setpri</code>	tid	priority			
<code>t_mode</code>	mode	mask			
<code>t_getreg</code>	tid	regnum	®val		
<code>t_setreg</code>	tid	regnum	regval		
<code>q_create</code>	name	count	flags		
<code>q_ident</code>	name	node			
<code>q_delete</code>	qid				
<code>q_send</code>	qid	buffer			
<code>q_urgent</code>	qid	buffer			
<code>q_broadcast</code>	qid	buffer			
<code>q_receive</code>	qid	buffer	flags	timeout	
<code>ev_send</code>	tid	event			
<code>ev_receive</code>	event	flags	timeout		
<code>as_catch</code>	asraddr	mode	flags		
<code>as_send</code>	tid	signal	flags		
<code>as_return</code>					
<code>s_create</code>	name	count	flags		
<code>s_ident</code>	name	node			
<code>s_delete</code>	sid				
<code>s_wait</code>	sid	flags	timeout		
<code>s_signal</code>	sid				
<code>tm_set</code>	timebuf				
<code>tm_get</code>	timebuf				
<code>tm_delay</code>	ticks				
<code>tm_tick</code>					
<code>v_return</code>					
<code>m_create</code>	name	length	logpage	paddr	flags
<code>m_ident</code>	name				
<code>m_delete</code>	mid				
<code>m_getseg</code>	mid	size	flags	timeout	
<code>m_retseg</code>	mid	paddr			
<code>pt_create</code>	name	bsize	bnum	paddr	flags
<code>pt_ident</code>	name	node			
<code>pt_delete</code>	ptid				
<code>pt_getbuf</code>	ptid				
<code>pt_retbuf</code>	ptid	bufaddr			
<code>mm_12p</code>	tid	laddr	&pages		
<code>mm_pmap</code>	tid	laddr	paddr	pages	
<code>mm_vmap</code>	tid	laddr	flags		
<code>mm_unmap</code>	tid	laddr			
<code>mm_pread</code>	paddr	laddr	bytes		
<code>mm_pwrite</code>	paddr	laddr	bytes		
<code>m_ext2int</code>	external				
<code>m_int2ext</code>	internal				

continued

Moreover, the use of additional software registers further extends the hardware register set and allows tasks to exchange pointers or global data variables quickly and conveniently through the use of the `t_getreg` and `t_setreg` directives.

MESSAGE, EVENT, AND SIGNAL MANAGEMENT

One of the most important features of a real-time system is its ability to respond to external signals and events commonly generated by hardware interrupts. The system must also respond to messages, events, and signals generated by other tasks executing in the real-time environment.

Many real-time applications are built entirely on a message-passing protocol. Here, hardware interrupts indicate the existence of some external event that must be handled immediately. Clearing and arming the interrupt are of primary importance; processing the data associated with the interrupt may be handled some time later by another task. An efficient and

deterministic message-passing mechanism is necessary to allow one task to send a message or an event to another (which could cause a task switch).

The message queue is the data structure supporting intertask communications and synchronization. One or more tasks may send messages to the message queue, and one or more tasks may request messages from the queue. Typically, with the RTEID, the `q_create` and `q_ident` calls are done at


Many real-time applications are built entirely on a message-passing protocol.

task initialization time and, as such, have no real performance criteria. The `q_delete` call is needed for cleanup purposes when a task is either going away or no longer needs the message queue.

The functions of the RTEID's message manager that have real-time requirements are `q_send`, `q_receive`, and `q_urgent`. The performance characteristics of these functions must be deterministic for a given run-time environment. The message queue could be a global object that is not specifically connected to any task or processor.

The RTEID, in addition to its message management function, provides a very fast synchronization method via its events mechanism. Events are simply bits encoded into an event mask. A task can send one or more events to another task using the `ev_send` directive. The target task receives events with the `ev_receive` directive.

Another important function sometimes used in real-time applications is the asynchronous routine. Asynchronous intertask communications is commonly used to process software signals that may occur at a random rate but that must be acted upon when they occur. Similar in function to hardware interrupts, asynchronous signals allow



NUTSHELL HANDBOOKS™

What's in them for you?

X11 Programming Manual \$60
Hardly a nutshell! Over 700 pages in two volumes—the most complete discussion of programming with Xlib available.

DOS Meets UNIX \$15
DOS/UNIX Merge, PC-Interface, VP/ix, MKS Toolkit, and more. An overview of what's out there and how it can help you. New!

Managing UUCP and Usenet \$18
Don't even TRY to install UUCP without it!!
—Usenet posting 456@nitrex.UUCP

...and many more titles!

1-800-338-NUTS
O'Reilly & Associates, Inc.
981 Chestnut Street Newton, Massachusetts 02164
in MA 617-527-4210 • e-mail to uunet!ora!nuts

3B2 Disk Drives

• **Internal Mounting** - User installable or coordinated with AT&T if required.

	Drive Only	W/ XM Kit
72Mb Miniscribe	\$1650.00	\$2040.00
72 Mb CDC	\$1800.00	\$2190.00
170 Mb Priam	\$4000.00	\$4390.00


Reel To Reel 9 Track Tape

• **SCSI Interface** - 7" and 10.5" reel capacity units; 1600/3200 bpi; and 6250 bpi configurations. Integratable with other AT&T SCSI units and controllers.

	W/O SCSI 3B Host Card	W/SCSI 3B Host Card
1600/3200 bpi	\$5500.00	\$7500.00
6250 bpi	\$9100.00	\$11100.00

HDM™ Units for 3B2s

External Hard Disk Module Units that allow 4 - 72 Mb disks, or 3 - 170 Mb disks, or removable winchester cartridge 10 Mb or 20 Mb disks installed without the addition of an extra drive card in the 3B2. Call for quotes and configurations available. Now HDM's are available in SCSI format. Endless configurations available.



(206) 868 - 6500
P.O. Box 3098,
Redmond, WA.
98073-3098

HDM is a registered trademark of SIMCO Computer Applications Inc.

CIRCLE NO. 158 ON INQUIRY CARD

CIRCLE NO. 48 ON INQUIRY CARD

tasks to do normal processing while awaiting a signal.

When a signal occurs, the task is activated at an appropriate routine to handle the signal and then return to normal processing. A task can send one or more signals to another task using the `as_send` directive.

If the receiving task has set up an asynchronous signal routine using the `as_catch` directive, the task will be dispatched to the signal routine. The `as_return` is executed at the completion of signal processing to return the task to its previous dispatch address.

SEMAPHORES TOO

Semaphores are a widely accepted method of arbitrating access to shared resources. The RTEID semaphore manager defines a set of directives to implement this functionality. The semaphore primitives provided meet different sets of requirements.

To control access to a single resource (either available or not), the

user creates a semaphore with a unity initial value. To control access to a resource pool (at any moment some resources are available and some are not), the user creates a semaphore with an initial value equal to the available resources.

The set of directives provided by the semaphore manager are `s_create` to create the semaphore, `s_ident` to obtain the id of a semaphore, `s_delete` to delete a semaphore, `s_wait` to wait for a semaphore, and `s_signal` to free it.

The RTEID supports two different memory allocation schemes. A region provides for allocation of variable sized memory areas or segments. And a partition provides for allocation of fixed sized buffers.

With these two memory-management schemes, applications have the flexibility to acquire various memory areas, depending on their needs and performance requirements. Large, variable-sized memory areas may be allocated for data processing, or small,

fixed-sized buffers may be acquired for implementing packetized communication protocols.

SOFTWARE DEVELOPMENT

Today's software developer may be reluctant to generate code for a real-time kernel interface because all the real-time kernels on the market have different interfaces. Still, most real-time kernels implement a basic set of functions for applications such as task support, memory support, and message semaphore, event, and time support.

These basic functions differ little between implementations, but their calling sequences are different. Moreover, some implementations have assembly language interfaces and some have C interfaces.

Within the software industry and, in particular, in the UNIX community, there is substantial movement toward automating the software development environment. For example, Computer Aided Software Engineering (CASE) tools are available that aid in the specification and documentation of the software project, the specification of code modules, data definition, and, finally, even the automatic generation of system code.

In essence, the software industry is evolving toward a standard software project-development environment. What is missing in this project environment are the tools necessary for the development of real-time software. True, there are efforts underway to integrate the higher level functionality of CASE environments with real-time executives. But these are for specific executives and CASE tools.

In the future, we must consider the benefits to the project manager, sys-

The software industry is evolving toward a standard software project-development environment.

tem designer, and programmer of a real-time system if a highly automated CASE tool could generate the real-time code necessary to implement the creation of tasks, message queues, semaphores, signals, and all the other functions needed by most real-time applications.

The programmer, under a well-known and comfortable environment, could develop documentation such as control flow and data definition, construct the logic diagram for the pro-



ACL ADVANCED COMMUNICATION LINK

Intelligent crossroads for multiusers

Nobody ever told your PC how to deal with eight channels of data traffic. But now, it's got some help. It's got the ACL Serial Card that takes the panic and frustration out of multiuser systems... and replaces them with speed and orderliness.

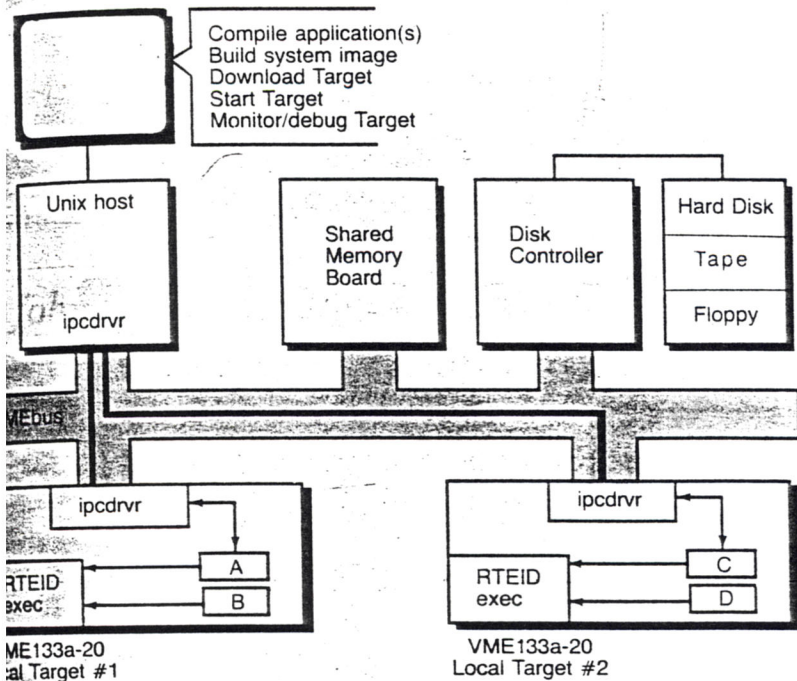
- **NO DATA PILE UPS!** High speed dual port memory teamed with a PC compatible CPU gives you a high performance communication coprocessor.
- **MAXIMUM ADAPTABILITY!** Eight independent channels of RS-232, with full hardware modem control handshakes, or software flow control. Also, RS-422 and RS-485 available, to interface with industrial process controls.
- **CREATE AND RUN YOUR OWN PROGRAMS!** Download custom programs and have ACL run them.

DRIVERS FOR PC, MOS, XENIX, UNIX, QNX AND THEOS! Give us a call. Find out how the ACL Serial Card can help your PC reach its full potential.



STAR GATE TECHNOLOGIES INC.
Cleveland, Ohio
1-800-STAR GATE

Figure 1: This application development system example shows how to combine the capabilities of UNIX with those of a real-time executive through the use of the Real-Time Executive Interface Definition (RTEID).



...n, generate the code, and analyze critical path for the target real-time em. Currently, this scenario is highly likely because of the wide variety of different interfaces to real-time systems. The RTEID could be the mechanism by which CASE tools developers generate highly reusable and portable code for real-time systems. The potential of this method for real-time system designers is great. They could have the ability to define the functional requirements, design the

VMEbus boards make a convenient platform for developing integrated UNIX and real-time system applications.

...ware components, generate code that has a very high probability of working correctly, and document the entire real-time project. With the use of the RTEID, there are many advantages to having UNIX installed in the target application system for certain application classes). The target is a real-time system that re-

quires a processor and some specialized I/O process control board and that needs a connection to some higher-level computation power. It turns out that more applications (even in the real-time process control world) could use the capabilities UNIX offers—especially in the area of data processing, report generation, and user interfaces.

THE VME EXEC PROJECT

VMEbus boards make a convenient platform for developing integrated UNIX and real-time system applications. But there are a number of problems that must be overcome before a successful application environment can be set up.

The system shown in Figure 1 illustrates the "VMEexec" UNIX-to-real-time executive strategy and is one example of how the RTEID may be used. The real-time development system comprises standard Motorola VME boards. The UNIX processor with sufficient memory and a disk controller make up the host development system.

The target system is built from two 68020-based processor boards (each with 1 megabyte of memory). The purpose of this configuration is to dem-

onstrate a real-time executive/UNIX integrated system with application development for real-time systems convenient for the system programmer.

From the UNIX system console, the system programmer first compiles the application program. The programmer then builds the target system image by executing a make file that links the applications with the RTEID kernel (and device drivers) into a single object file. At this point, the system image has been developed and stored on the UNIX disk.

The image is then loaded into the memory of one of the target processors. This is accomplished by starting a UNIX process (from the console), which transfers the system image file to the target processor. The starting address and dual-ported memory offset for the target board are needed to transfer the image.

This specific configuration has the target real-time processors in the same chassis as the UNIX host. Further enhancements will allow the software download to take place over an RS-232 link or a network connection to the target system.

The system start involves sending a command to the resident firmware monitor on the target processors to execute the downloaded code at the specified address. Run-time communication between tasks running on the target processors and processes running on the UNIX host is through I/O calls to the Inter Processor Communication Driver (ipcdrv). Tasks on the target make use of the RTEID executive for real-time functions.

This configuration is useful for CPU-intensive applications requiring little I/O and minimum use of the system bus. Any results of calculations are passed to a UNIX process via the ipcdrv for subsequent storage.

Monitoring or debugging the target system involves sending specific debug commands to a target-resident component of a UNIX-based debugger. These commands are translated to cause breakpoints, and the tracing and display of target task memory occur on a multi-windowed debugger running on the UNIX host. □

Please express your interest in this article by circling the appropriate number on the reader inquiry card.
High 719 Med. 720 Low 721

