OVERVIEW OF REAL-TIME KERNELS AT THE SUPERCONDUCTING SUPER COLLIDER LABORATORY

K. Low, S. Acharya, M. Allen, E. Faught, D. Haenni, C. Kalbfleisch
SSC Laboratory *
2550 Beckleymeade Ave.
Dallas, Texas 75237

Abstract

The Superconducting Super Collider Laboratory (SSCL) will have many subsystems that will require real-time microprocessor control. Examples of such sub-systems requiring real-time controls are power supply ramp generators and quench protection monitors for the superconducting magnets. We plan on using a commercial multitasking real-time kernel in these systems. These kernels must perform in a consistent, reliable and efficient manner. Actual performance measurements have been conducted on four different kernels, all running on the same hardware platform. The measurements fall into two categories. Throughput measurements covering the "non-real-time" aspects of the kernel include process creation/termination times, interprocess communication facilities involving messages, semaphores and shared memory and memory allocation/deallocation. Measurements concentrating on real-time response are context switch times, interrupt latencies and interrupt task response.

I. INTRODUCTION

The process of evaluating real-time kernels from different vendors can be a confusing experience. One is faced with a plethora of performance numbers from the individual vendors' information packages, each displaying superiority and advantages over their competitors. Each vendor invariably measures performance numbers in different ways and on different hardware platforms thus making comparisons almost meaningless.

To compare and evaluate the different offerings, we performed our own tests in a controlled environment. Products from the four vendors that met our base requirements were tested on the same hardware platform. The platform on which all four vendors is supported is the MVME147S-1 [1]; a VME based, single board computer with a 25MHz 68030 from Motorola. The four kernels selected, listed in no particular order, were pSOS+ from Software Components Group [2], VRTX32 from Ready Systems [3], VxWorks (v4.02) from Wind River Systems [4] and LynxOS (v1.21) from Lynx Real-Time Systems [5].

It should be stressed that these tests only provide quantitative measurements of a particular system's performance. Qualitative aspects such as development environment, debug capabilities, connectivity, compliance with industry standards, technical support and host/target availability will be addressed at the end of this paper.

Each test was executed a number of times in order to compute the average time to complete a test. The entire measurement is then repeated several times to measure the variance of this average value in the form of maximum and minimum average values. Clock resolution, number of iterations and cache conditions were identical for all four kernels.

II. THROUGHPUT MEASUREMENTS

Throughput measurements are tabulated in Table 1 and what follows is a brief description of each test as it appears in the table. Idiosyncrasies of each kernel will also be noted. An asterisk means that a particular test could not be performed on that kernel.

1. Create/Delete Task This test measures the time it takes to create and delete a task. A task deletes itself as soon as it is created. The created task has a higher priority than the creator, so the time quoted actually includes a create, start, delete and two task context switches.

2. Ping Suspend/Resume Task A low priority task resumes a suspended high priority task. The high priority task immediately suspends itself. This measurement includes two task context switches and the time it takes to suspend and resume a task. There is no facility to suspend and resume a task on LynxOS apart from using signals. So this test was not performed under LynxOS.

3. Suspend/Resume Task This is identical to previous test except that a high priority task suspends and resumes a suspended lower priority task so that there is no context switching.

4. Ping Semaphore Two tasks of the same priority communicate with each other through semaphores. Task A creates a semaphore, gets the semaphore and then creates Task B which blocks when it attempts to get the semaphore. Task A then releases the semaphore which immediately unblocks Task B. Task A then attempts to get the semaphore which causes it to block until Task B releases it. The two tasks then alternate ownership of the semaphore thereby causing context switches. In our version of VxWorks, two separate semaphores are required because round-robin scheduling is not supported.

5. Getting/Releasing Semaphore The time reported includes the time it takes to get and immediately release a semaphore within the same task context.

6. Queue Fill, Drain, Fill Urgent We first time how long it takes to fill a queue with messages and then we time how

*Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC02-89ER40486.

Table 1: Throughput Measurements

Test Description	pSOS+ min/max/avg μsec	VRTX32 min/max/avg μsec	LynxOS min/max/avg μsec	VxWorks min/max/avg μsec
Create/Delete Task	540/600/591	370/380/371	*	1378/1446/1423
Ping Suspend/Resume Task	120/130/128	140/150/142	*	174/182/177
Suspend/Resume Task	80/90/83	80/90/87	*	68/74/69
Ping Semaphore	210/220/219	230/250/239	390/400/397	228/234/232
Getting/Releasing Semaphore	63/64/63	55/56/55	73/76/74	33/34/33
Queue Fill	40/50/46	20/30/26	136/146/140	19/21/20
Queue Drain	40/50/43	20/40/29	126/136/132	21/25/22
Queue Fill Urgent	40/50/47	20/30/27	166/175/170	70/76/72
Single Queue Fill/Drain	90/93/91	50/70/59	270/290/278	43/48/44
Alternate Queues Fill/Drain	230/240/238	250/260/252	860/900/867	366/376/371
Allocate Memory	40/40/40	20/30/27	34/79/57	67/71/68
Deallocate Memory	30/40/38	30/40/33	20/21/20	82/86/83

long it takes to drain the queue. Finally we repeat the two tests with priority messages i.e. messages are sent to the head of the queue. VxWorks 4.02 does not support message queues but ring buffers with semaphores gives the functionality of a message queue. LynxOS uses SysV message queues with priority messages handled differently.

7. Queue Fill/Drain A single task sends a message to a queue which the task immediately receives on the same queue. There is no task context switch nor is there any pending queue operations. The next test consists of two tasks with two queues. The two tasks alternate execution by sending to the queue that the other is blocked waiting to receive from. The total time now includes context switches, queue pends and sending plus receiving a message.

8. Allocating/Deallocating Memory We measure the time it takes to allocate a number of buffers from a memory partition and the time it takes to return those buffers to the partition.

III. REAL-TIME RESPONSE

The Motorola MVME147S-1 includes an auxiliary timer capable of generating interrupts. A driver for the timer was written for all four kernels. We quantify the real-time response of the kernels by measuring the interrupt service response and the interrupt task response. The interrupt service response is the time it takes to execute the first instruction of an interrupt service routine (ISR) from when the interrupt occurs. The task response is the time it takes for a user task to resume execution from when the interrupt occurs. These measurements were taken over a large number of times and the maximum, minimum and average times are reported over the span of the test. The LynxOS was the only kernel with a SCSI disk attached to it and all kernels had network attachments and a real-time clock as other sources of interrupts. The source of interrupts for the actual measurement was an auxiliary counter on the MVME147S-1 and the measurement task runs at the highest priority.

Typically, a user task is blocked waiting for a semaphore to be released by the ISR. The counter is programmed to start counting up from a preset value to a maximum value when it will generate an interrupt, resets itself to the preset value and begins counting up again. Each count corresponds to 6.25 μs. The ISR then immediately reads the counter, which gives the interrupt response time, and then releases the semaphore. When the kernel reschedules the user task after completion of the ISR, the user task becomes unblocked, reads the counter which then gives the task response time.

IV. OBSERVATIONS

pSOS+ is a robust real-time kernel. Code can be developed on a number of different host platforms and downloaded to the target with the final application stand-alone in ROM. Software Components Group (SCG) supports pSOS+ on many target systems and provides source to drivers making ports to specialized boards easier. The XRAY+ debugger, based on the popular XRAY debugger from Microtec [6] is capable of debugging target resident optimized C source code across ethernet or RS-232. There is also an X11 interface which offers increased versatility. In addition to task-level breakpoints, system-level breaks can also be set at the system-level; stopping all tasks. This allows access to the onboard monitor and the state of all pSOS+ objects. Optional components provide UNIX-compatible network facilities and an ANSI standard run-time library. Field support was excellent.

VRTX, from Ready Systems, provides a full complement of support software in addition to the VRTX/32 real-time kernel. These include packages for I/O file management, networking, multiprocessing and a run-time library. VRTX is supported on several commercially available target boards with supporting documentation for porting

Table 2: Real-Time Response

	pSOS+ min/max/avg μsec	VRTX32 min/max/avg μsec	LynxOS min/max/avg μsec	VxWorks min/max/avg μsec
Interrupt Service Response	6/6/6	6/6/6	13/88/13	6/56/6
Interrupt Task Response	100/169/163	169/343/169	163/262/175	119/319/125

VRTX to customized boards. Host support currently exists only for SUN3/SUN4 with Sun's own proprietary windowing environment. The source level debugger (RTsource) and the symbolic debugger (RTscope) can function across an ethernet/serial link between the host and target. Like pSOS+, breakpoints can be set at task as well as system level. Tasks may be stopped and information about kernel data structures displayed. A run-time shell with dynamic linking capability is available for quick prototyping of applications. Although somewhat daunting to the first-time user, VRTX is an extremely flexible and versatile system to the initiated.

VxWorks includes a proven real-time kernel and a UNIX cross-development package with extensive UNIX-compatible networking facilities. Version 4.02 supports only a preemptive priority scheduling kernel while V5.0 offers in addition round-robin scheduling. Version 5.0 also promises better performance with some compliance to Posix 1003.4 Real-Time Extensions. VxWorks currently is ported to a number of different target boards with the host support fully implemented only on the SUN3/SUN4 systems. The source-level debugger is a remote debugger based on the Free Software Foundation GDB [7]. The debugger can only debug single tasks and currently does not have an X11 interface. A symbolic debugger with some system status displays is also standard. Dynamic loading of objects over the network or from a disk together with an interactive C-interpreter interface can be useful during the development cycle.

LynxOS provides a complete Unix development environment. It can also be used for a cross-development system like the other three kernels. It offers good real-time performance with memory protection. LynxOS 1.21 currently offers compliance to Posix 1003.1, SVID 4.2 and BSD 4.3 with future releases complying with 1003.4 Draft 9 (Real-Time Extensions). It has been ported to four different computer architectures. It has a Unix System V.3 binary compatible interface built into the LynxOS kernel so that binaries work under LynxOS and the standard Unix for that architecture without modification. The debug environment consists of GDB as the source-level debugger. There is presently no kernel debugger.

V. CONCLUSIONS

It has been our experience that a compile-download-debug cycle common with all the embedded systems is not a major problem for us, Ethernet and NFS links make this a speedy process.

It has become apparent the importance of compliance with standards. Standards adherence makes code more portable. We had to effectively rewrite all the tests for all the kernels because of the interface differences.

Another conclusion is the importance of having a mature debugging environment, a source-level remote debugger with a X11 Windows interface that can debug optimized code is extremely useful. A good kernel debugger is also very important, allowing the user to halt all tasks and examine states of any individual task with relationship to other tasks.

After we factor in the hardware differences between our environment and the individual vendors' test bed, most of the timing results we obtain agrees surprisingly well with the respective vendors' published values.

Furthermore, we realize that differences in compilers can contribute to the overall performance of the kernels and will require further investigation.

Finally, the more hosts and targets that a given cross-development kernel supports, the more attractive it will be, especially in a vastly heterogeneous environment like the SSC.

References

[1] Motorola, Inc., Technical Systems Division, P.O. Box 2953, Phoenix AZ 85062

[2] Software Component Group, Inc., 1731 Technology Drive, San Jose, CA 95110, (408) 437-0700

[3] Ready Systems, Inc., 470 Potrero Ave., P.O. Box 60217, Sunnyvale, CA 94086

[4] Wind River Systems, Inc., 1010 Atlantic Ave., Alameda, CA 94501, (415) 748-4100

[5] Lynx Real-Time Systems, Inc., 16780 Lark Ave., Los Gatos, CA 95030, (408) 354-7770

[6] Microtec Research, Inc., 2350 Mission College Blvd., Santa Clara, CA 95054, (408) 980-1300

[7] Free Software Foundation, 675 Massachusetts Ave., Cambridge, MA 02139