A NEW REAL-TIME OPERATING SYSTEM AND PYTHON SCRIPTING ON ALADDIN*

D.E. Eisert#, R.A. Bosch, K.D. Jacobs, K.J. Kleman, J.P. Stott, Synchrotron Radiation Center, University of Wisconsin-Madison, 3731 Schneider Drive, Stoughton, WI 53589-3097, USA

Abstract

We are in the process of upgrading the VME processors on the Aladdin electron-storage-ring control system. The last major redesign of the control system occurred in the mid 1980's. At that time we converted to VME microcomputers and VAX/VMS workstations communicating via Ethernet. This is the second upgrade since then of the VME processor. As upgrades of the Motorola 680x0 processor are no longer available we have decided to switch to the Intel Pentium III. This change allowed us to reconsider our use of the rather primitive μC/OS kernel and implement a commercial real-time OS. We decided to use QNX primarily as it was a good match to our existing software and was zero cost. In addition to upgrading the CPUs we have also added a new scripting language to our main control application. We used SWIG (Simplified Wrapper and Interface Generator) to create wrapper code for the scripting software. SWIG can create wrapper code for many scripting languages so our initial choice of a scripting language was not critical. We decided to start by using Python due to the many available add-on libraries and the apparent ability to support larger projects. We will discuss our evaluation process and the challenges we encountered.

INTRODUCTION

Our present control system utilizes several generic desktop PCs running Windows 2000, eight VME systems running QNX [1] and ten dedicated power-supply controllers running μC/OS [2,3]. The network is primarily 100 Mbps switched Ethernet with a 1 Gbps over copper link between the storage ring and the control room. Our new VME processor boards (VMIC 7750) use a 733 MHz Pentium III processor and have 64 MB of Compact Flash storage, 64 MB of SODIMM RAM and dual 100 Mbps Ethernet ports. These boards replaced processor boards based on the Motorola 68030. The dedicated power-supply controllers are built into our dipole and quadrupole power supplies. They were updated with new VME processors based on the Motorola MC68360 in 2000. These controllers are actually the oldest processors in our control system.

The size of our control system is smaller than many facilities [4]. As a result we no longer have a controls group. Several years ago the controls group was trimmed from three staff members down to one. The responsibilities were also reduced and no longer include facility computing and network support. The operations group supplies staff for cabling and transition panel work as needed by the control system.

* Work supported by the U.S. National Science Foundation under Award No. DMR-0084402.
#deisert@src.wisc.edu

REAL-TIME OPERATING SYSTEMS

For the last ten years we have been using μC/OS for simple task management. It doesn't include any support for networking or memory management. We wrote our own networking software and simple heap management software. Typically our budget is consumed by hardware purchases so we are forced to write our own software or find an open source solution.

While investigating many possible real-time operating systems, we discovered that QNX had recently changed its licensing policy. The software was now offered free for non-commercial use. We verified with their sales department that this new licensing policy allowed us to use QNX free of charge.

Some possible operating system choices included μC/OS, RTEMS, several versions of real-time Linux and QNX. μC/OS has served us well but we desired a more complete package. RTEMS appeared to be an improvement over μC/OS but its support is not as strong as real-time Linux.

The choice was narrowed down between the various real-time Linux systems and QNX. QNX has many advantages for us:

- Drivers are separate from the kernel and they run in their own protected address space. Rebuilding the kernel is not required for every driver change.
- Drivers can be started and stopped independently from the kernel.
- QNX's built-in message passing functions closely matched our existing software's need for interprocess communication.
- QNX has a cleaner implementation of its networking software. Linux switches out of its real-time kernel for network communication.
- QNX has a much shorter learning curve for people unfamiliar with Linux. There is no need to learn how to build the kernel or how to write kernel drivers.

PROCESSOR SELECTION

With the availability of no cost real-time OS's the decision of processor architecture became less important. The decision could now be made based on cost vs. performance. We requested bids for a VME processor board with a Pentium III, Celeron or Motorola MPC750 processor. We also specified some form of on-board flash memory, 100 Mb Ethernet and a PMC expansion site.

Unfortunately it seems that many vendors don't like to respond to the generic bids required by our purchasing department. But we did receive a reasonable selection of low bidders with prices that varied by less than about 10%. The VMIC board with the Pentium III clearly had the most computing power. Other benefits included removable/upgradeable RAM and Compact Flash, two Ethernet controllers and available on-board 32-bit timers.

One concern is that the heat generated by the Pentium III is greater than that generated by the PowerPC processor. VMIC claimed that with proper cooling the Pentium III processor would not have problems. To test this a VMIC board was initially configured to run Windows 2000. A CPU thermal stressing program was downloaded from the Internet. The processor temperature climbed to 50 °C in about 5 minutes. The BIOS was configured to throttle down the processor by 50% at that temperature. In the throttled down state it would cool off and then the BIOS would return the processor to full speed. This cycle would continue indefinitely. Unfortunately the early BIOS must have been configured to set a flag if it reached the throttle down temperature. When attempting to reboot after this test, the board indicated a problem by flashing a LED. VMIC's documentation instructed that the board must be returned to the factory. The service record from the factory only indicated that the BIOS firmware had been updated to fix the flashing LED. This would not be suitable for number crunching but it probably does not represent a problem for real-time control applications.

CODE DEVELOPMENT

The first task in porting our software was writing a driver for the PCI-VME bus interface (Tundra Universe II). At that time VMIC had not written the driver for the latest version of QNX. In fact they suggested they might be interested in supplying the prior version's source code to us if we would update it for them. We declined since the source code was available for a Linux driver. Creating the QNX version did not prove to be an overly ambitious project.

Most OS driver protocols typically require procedures that are similar to the standard file processing functions. These procedure calls allow only a limited number of parameters to be passed. Our original drivers assume that the interface would allow a significant number of parameters to be passed with the function calls. Initially we thought of trying to bypass the built in driver support library and communicate via a lower level raw message format. But one of the many technical articles available on QNX's web site detailed an easy method to extend the driver call to pass the parameters we required.

One of the unique features of our drivers is their ability to be restarted on the fly. When the new driver is started, it requests all of the information needed to restart the driver from the old process. The old driver then exits and the new driver is able to initialize without modifying the contents of the board's registers.

We were concerned that the architecture change would cause byte order problems. The Motorola and Intel processors use different integer byte ordering. The VMIC board has on-board hardware to automatically correct byte order problems. As a result, all but one of our drivers were ported without any special consideration to byte order. The driver that had difficulty was a stepping motor driver that required four byte integers to be written to a byte wide register.

Polling the analog input boards requires several percent of the CPU. This may seem curious at first since the task does not require much processing. Upon further investigation it appears to be caused by bus throughput limitations. Our older 16 bit VME bus can only transfer data at about 1.2 MB per second.

INSTALLATION

The VME processor boards were upgraded over a nine-month period. During that time both the new and old processors were used together in the control system. Normally the processor would be changed during the beginning of a ring development period coinciding with other work on the ring. After initial testing the system would be monitored for the rest of the development period. If no problems were detected, it would remain on the ring during the user period.

One of the VME systems developed problems after installation. Approximately once a week the system will stop servicing interrupt requests on the VME bus. The system is nearly identical to several other systems that do not have this problem. So it may be a very subtle software bug or hardware related problem. The problem is too infrequent to isolate during development periods. The work around is to monitor interrupts in the PCI-VME Tundra Universe driver. When a lockup occurs, a shell script is spawned to kill all of the affected drivers. The drivers are then restarted within in about a second. This is done without losing any of the stored beam. A small delay may occur in some of our feedback loops but it should be nearly impossible for the users to detect. We hope to have this problem fixed soon but the ability of QNX to restart drivers without rebooting is quite impressive.

CONTROL SOFTWARE

The primary program used to operate our facility is called the Page Program. The program is very easy to use and its tabular appearance allows a significant amount of information to be neatly displayed on the screen. Much of its basic look and feel has not changed in over a decade. Operators like the consistency of the user interface. But for many years there have been requests to improve its primitive built-in scripting language. The amount of work involved seemed considerable due to the program's internal dependence on its own scripting language.

We did have another method to develop programs without having to write traditional software. In the mid 1990's LabView had been extended to allow access to the control system. LabView could create great looking GUI

screens and had built in plotting and data analysis. Our beamlines had been using it for years but most staff are reluctant to use it.

THE TOOLS

There are several open source scripting languages to choose from. Tcl/Tk seems to be very popular but the syntax is very similar to shell scripts. Python's syntax looked appealing due to its greater similarities to a traditional programming language [5]. It also appeared to have more libraries, including matrix libraries.

The initial choice of a scripting language soon became irrelevant. SWIG claims to allow applications to be written for most of the popular scripting languages with only minor changes [6]. Being a small facility we only wanted to support one language. SWIG allowed us to initially pick any language we wanted without the risk of making the wrong choice.

EMBEDDING THE LANGUAGE

The first step was to write an extension for Python using SWIG. After only a couple of days most of the common operations were available, such as reading and writing device settings and device database queries. The software was installed on one workstation and a notice was e-mailed to a small group of staff members. The response was immediate and very enthusiastic. Several staff members dropped their current projects and started experimenting with the new software. Another week or two was spent testing, documenting and adding new functions to this extension. The DLL extension was also used to enable control system access for IGOR and MATLAB.

The next step was to replace the primitive scripting in the Page Program. This proved to be quite simple but time consuming due to the number of functions that were created for Python. Redirecting Python error messages posed a slight difficulty. Python documentation explains the procedure but the task could be made a bit more straightforward. It is also possible to create multiple threads in the program all running Python scripts. It was decided to limit the program to only one thread running a Python script due to the possible confusion it could create for the operators.

The final step was to translate all of our existing scripts into Python. The original scripting language was modeled after VAX/VMS's Digital Command Language. It allowed command options to be placed anywhere in the command string. In only a few hours a Python script was written that did the translation. More than 100 scripts were translated to Python in a single afternoon.

IMPLEMENTATION

Surprisingly most of the initial scripts that were written performed operations that were already available in existing software. But eventually these scripts became building blocks for larger measurement operations. Scripts were written to change lattices, apply quadrupole corrections and start orbit feedback and undulator compensations with the click of a button. Many scripts are so complex that they require several hours to execute. Scripts have also been used to create feed forward control loops for newly installed hardware.

The original work on creating a Python extension proved to be very useful. Initially it was written only to experiment with the scripting language. But it now allows us to spawn Python scripts that are independent from our main control program. It has also been useful for developing new scripts during user periods. Access to the control system is prohibited from most PCs therefore initial debugging can be performed on them without affecting the ring. Access control will be expanded in the future to allow limited real-time access to the control system from these systems.

Excessive reliance on scripting for normal operation could create some problems. Operators could become complacent and would be less likely to notice problems unless the script or other monitoring software detects them. Operators could lose the ability to recover from problems if a routine configuration script fails. Operators need to review the procedures frequently and stay informed about changes to maintain their skills.

CONCLUSION

Replacing our older VME processor boards has brought increased reliability and future expandability. Some staff were disappointed that they did not detect any change after the upgrade. Actually the primary goal of the project was to implement the change without disruption. The initial software was created to be identical to the old system while creating a base of expansion for years to come.

The new scripting software appears to be serving the needs of the accelerator development group. This clearly demonstrates that ease of use is very important to the acceptance of a tool. An added benefit is the ability to distribute software related work across more individuals. The focus of the controls related work can now be more concentrated on supplying tools for the operation and accelerator development groups rather than end user applications.

REFERENCES

[1] http://www.qnx.com
[2] Jean J. Labrosse, μC/OS The Real-Time Kernel, R&D Publications, (1992).
[3] D. E. Eisert, Fermilab Report CONF-96/069, 851 (1996).
[4] J. P. Stott and D. E. Eisert, Nucl. Instr. and Meth. A293, 107 (1990).
[5] http://www.python.com
[6] http://www.swig.org