Real-Time Linux at MLRS

Randall L. Ricklefs
The University of Texas at Austin / Center for Space Research
Contact: ricklefs@csr.utexas.edu

Article
13-Po25

Abstract

The McDonald Laser Ranging Station (MLRS) has used a proprietary, unix-like real-time operating system to control tracking, ranging, and timing for 2 decades. With the availability of a 7 year duration Long Term Support version of Ubuntu Linux, as well as the Real Time Application Interface (RTAI) and Real Time Device Driver interface (RTDM) add-ons for real-time control, an opportunity presented itself to convert to a stable, modern, open-source alternative. We will review the architecture and development of this system and describe recent experience with its installation and operation.

Introduction

In the early 1990's, MLRS as well as the rest of the NASA SLR network converted their real-time ranging control computers from a set of disparate minicomputers to commodity PCs running the Unix-like real-time operating system LynxOS. Real-time functionality was needed to service interrupts in various subsystems in a timely fashion. After a market survey, there were only a couple of affordable contenders that provided a portable interface using POSIX and X11 libraries to provide a real-time platform with a portable graphical user interface. [1] Recently, as LynxOS became more expensive for one station (we are no longer able to share a development environment with the rest of the NASA network), we needed to find a viable alternative.

We have had a great deal of experience working in a Linux environment, and became acquainted with the various open source real-time options for Linux. Ultimately, RTAI (Real Time Application Interface) was chosen.[2] It includes patches to the Linux kernel and supplies a set of real time libraries and schedulers. Later, when it became clear that the device driver for our primary bus system (CAMAC) needed to have real-time capability, we also adopted RTDM (Real Time Device Model). This is a library of wrapper routines to make writing a real-time device driver simpler and more transportable between real time implementations like RTAI and Xenomial.

The chosen Linux distribution, after experimenting with a couple others [2], is **Ubuntu 12.04 LTS**. The boot and shutdown times are shorter than for most of the others distributions, and this version will be supported with security and bug patches until 2017. Shorter boot times help if the system must be rebooted during a pass, and the long term support will insure that the system will remain safely patched for some years – and time will not need to be spent upgrading to a newer distribution every 12-18 months.

The architecture of the software remains the same for the Linux version as for the LynxOS version. There is a multi-threaded main program, called the Monitor, that handles all interactions with the station hardware – mount, timing, ranging hardware, meteorological sensors, xy stage, and so forth. This program forks several other programs, only one of which can be active at a single time, and communicates with them through shared memory. These are the star calibration program, the satellite tracking program, and programs to calibration the TD811 ranging timer and the absolute and incremental encoders. Each of these programs is also multi-threaded. All were converted for real time use with RTAI. All programs are written in c, and heavy use is made of the X11/Motif graphical user interface libraries.

This conversion was undertaken as a low priority project without interference with the usual ranging activities. There were 3 phases to creating the real-time Linux controller system. Each brought us closer to a real-time response.

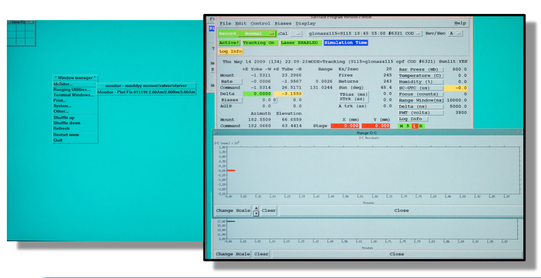
Production Experience

The Linux real-time system has been in use at MLRS since late August, 2013. Observer training took little time, since there is virtually no difference in the graphical user interface of the ranging programs. In addition, the Linux desktop environment is familiar to the observers because the data analysis computer also runs Linux. Instead of launching the Monitor program from the root window menu under LynxOS, the observer double clicks an icon on the desktop.

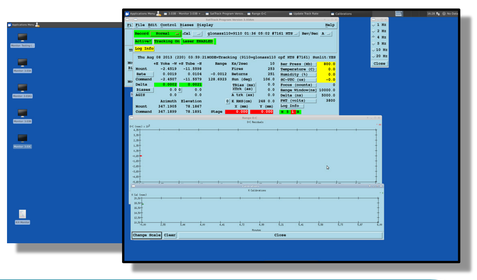
Weather and some software problems limited the number and quality of passes taken early on. Later, a couple of annoying problems in the software that proved to limit data and frustrate the observers were remedied, and tracking is now proceeding normally. These problems involved identifying lost interrupts and limiting their detrimental effects. Although the problem of missed interrupts is not new to this real time Linux implementation, there may still be some tuning of the RTAI/Linux ranging code that can further reduce them.

Although Ubuntu boots fairly quickly compared to other distributions, it is slower than LynxOS. If there are software or hardware glitches that require a reboot in the middle of a pass, added time rebooting can be frustrating for the operators and can result in the loss of data. To mitigate this slowness, the hard disk drive was replaced with a 240GB Solid State Drive (SSD). The boot time was cut almost in half, and is comparable to LynxOS boot times.

The choice of desktop environment is evolving. Only a fairly simple desktop is needed to launch the ranging programs and carry out system administration. Production use started with Gnome Classic. This has a bug which often prevents shutdown and reboot from the usual desktop menus. The XFCE desktop may be used in its place.



Old
versus
New environment



Working with RTAI/RTDM

As is often common when beginning to use a library that is new to the programmer, a fair amount of experimentation was required to learn what set of commands perform the required task, and RTAI and RTDM are no exceptions. The learning curve for RTAI/RTDM was substantial, mainly because there is no up-to-date or in-depth instructional book. Everything that is available is on the web, although RTAI is discussed in one programming book[6]. Much of the HOWTO and User Manuals, though quite useful, are somewhat out-of-date. Some documentation comes from an earlier era before the product was mature, leaving the reader to wonder about the current state and procedures. However, the RTAI web site has excellent and up-to-date API (Application Programming Interface) documentation. There is also a wealth of working code on the RTAI and related web-sites. An active RTAI forum and archive provide needed access to RTAI expertise. If a question isn't answered in the archive or elsewhere, a query to the forum will bring a quick response.

System Development

Phase 1 – Conversion from LynxOS to Linux

Converting the c code from LynxOS version 4.2 to Linux was a straight forward task. The main changes were due to differences in header names, such as replacing #include <shm.h> with #include <sys/shm.h>; absence of certain defines, such as SIGUDEF29; and the addition of certain headers. Other minor changes were due to the serial port device names for the Blue Heat 8250 PCI 8-port serial board. The serial devices are named, for example, /dev/ttyS1 (CentOS) or /dev/ttyCTI0 (Ubuntu), instead of /dev/cbh0.0 for LynxOS. In the code that initializes serial ports, the define "V_B9600" was replaced with "B9600". Also, a call to cfsetispeed and cfsetospeed were needed to set the baud rate.

The big change was replacing the CAMAC kernel device driver and its interface in the user space controller program (Monitor). MLRS (and the rest of the NASA network) uses the Hytec CAMAC TURBO Personal Computer Interface Type 1331 (which takes up 2 slots in the CAMAC crate) and the Hytec PCI5331 PCI "personality card", which fits in a slot in the controller computer. The Linux device driver and user space interface library, written by Stefan Ritt, was downloaded and installed [2]. The LynxOS device driver used an ioctl call in user space to wait for interrupts. This Linux driver uses an interrupt service routine in user space to wait for kernel driver interrupts (triggered by kernel level "wake_up_interruptible" and "kill_fasync" calls).

Phase 2 – Adding real-time support into user space

The Linux-only version worked, but failed, as expected, in guaranteeing deterministic response times. Next came installing RTAI kernel patches and libraries following the setup guidelines [4] and users manual[5]. This involved downloading RTAI 3.6 (available from rtai.org and through the Ubuntu Software Manager) and a compatible generic kernel from kernel.org. The latest kernel that was supported by RTAI 3.6 was 2.6.32.59, and is used here.

The test suite that accompanies RTAI shows quite good performance in kernel space tests, with about 5 µsec maximum latency and 4 µsec preemption latency. As expected, the user space tests were not as good, with 21 µsec maximum latency and 12 µsec preemption latency. There may be some "latency killers" that have not yet been located. These results were obtained using our existing controller computer, a Dell Precision 380 with a 3.2 GHz Pentium 4.

Once RTAI was installed, the process of replacing or augmenting Linux system calls with RTAI routines began. A little functionality was added each time until a usable system emerged. There are 2 real-time flavors, soft and hard. Soft real-time has better characteristics than the typical Linux scheduling, but not deterministic enough for critical applications. Hard read-time is used in portions of the code where there are critical timing needs, such as interrupt or timer handling. The key is to avoid page swaps and Linux system calls within the hard real-time portion of any thread. Each real-time thread needs to be locked into memory with adequate memory so that it does not try to grow and therefore cause a time-consuming page-fault. A helper thread is then created to interact with the RTAI scheduler.

As it turns out, the downfall of this approach was that the CAMAC driver interrupts and calls were not set up for hard real time. That brings us to the final phase.

Phase 3 – Adding real-time support into kernel space – the CAMAC device driver

The only device in our ranging system that requires hard real time responses to interrupts is the CAMAC crate. The CAMAC contains the system clock and the ranging timer, both of which produce interrupts which must be serviced immediately. The meteorological system and XY stage, both attached to serial ports, do not have time-critical response requirements. As mentioned above, the modules in the CAMAC crate are accessed through a Hytec 1331 CAMAC module and a 5331 PCI card. Initially the open source device driver and wrapper libraries described above were used. However, while these are fine for low speed work, the response time was inadequate for our laser ranging system. Thus, it was necessary to convert this kernel-level driver to real time using RTAI and RTDM techniques and calls. Almost all CAMAC services, including interrupts, are accessed through ioctl calls, similar to the LynxOS technique. Once this conversion was accomplished, ranging tests could begin.

It should be noted that since the MLRS is 500 miles away from the software developer in Austin, and since no spare CAMAC hardware is available, initial development in each of these steps required using a CAMAC device driver that simulated, to some crude extent, the response of the real hardware. Creating this driver required different techniques from those in the operational driver, further distracting from the job at hand.

Conclusion

The conversion from LynxOS to RTAI/Linux at MLRS was a low priority/ low risk venture that bore fruit after a couple years of part time work. To gain experience and control, the project was broken up into several phases, each getting closer to the goal of real-time response. Although the learning curve for RTAI and RTDM was challenging, it was comparable to other unfamiliar libraries. Now that the real-time Linux software is in place, it provides an effective, low cost, stable, and tool-rich environment for further development and maintenance.

References

[1] Ricklefs, R., Check, J., McGarry, J.F., *"Upgrading NASA/DOSE Laser Ranging System Control Computers", 8th* International Workshop on Laser Ranging, Annapolis, MD USA, May 18-22, 2002.
[2]Hoffman, E., Ricklefs, R, *Controlling Laser Ranging with RTAI-based Real-Time Linux,* 17th International Workshop on Laser Ranging, Bad Kötzting, Germany, May 16-20, 2011.
[3] https://github.com/cjpl/midas/tree/master/drivers/kernel/khyt1331_24 .
[4] *RTAI Installation Complete Guide*, Joao Monteiro, Feb 27, 2008, from www.rtai.org .
[5] *RTAI 3.4 Users Manual, rev 0.3*, October 2006, from www.rtai.org .
[6] *Linux for Embedded and Real-time Applications*, D. Abbott, Elsevier, 2006.