MANAGING A REAL-TIME EMBEDDED LINUX PLATFORM WITH BUILDROOT

J. Diamond#, K. Martin, FNAL*, Batavia, IL 60510, U.S.A.

Abstract

Developers of real-time embedded software often need to build the operating system, kernel, tools and supporting applications from source to work with the differences in their hardware configuration. The first attempts to introduce Linux-based real-time embedded systems into the Fermilab accelerator controls system used this approach but it was found to be time-consuming, difficult to maintain and difficult to adapt to different hardware configurations. Buildroot is an open source build system with a menu-driven configuration tool (similar to the Linux kernel build system) that automates this process. A customized Buildroot [1] system has been developed for use in the Fermilab accelerator controls system that includes several hardware configuration profiles (including Intel, ARM and PowerPC) and packages for Fermilab support software. A bootable image file is produced containing the Linux kernel, shell and supporting software suite that varies from 3 to 20 megabytes large – ideal for network booting. The result is a platform that is easier to maintain and deploy in diverse hardware configurations.

INTRODUCTION

Scientific Linux [2] is a Linux distribution produced by a collaborate effort between Fermilab, CERN and other members of the global Linux community. Scientific Linux has been used successfully as a target platform for data acquisition in the Fermilab accelerator control system but only on commodity PC hardware and not with real-time extensions [3]. When attempting to adopt Scientific Linux to a real-time embedded application some road-blocks were encountered. The first was that Linux is not designed to be a hard real-time operating system. There are however, several options for making Linux real-time and all of these options require a custom kernel be built. At Fermilab, one route chosen was to use the Real-Time Application Interface (RTAI) [4]. The next road-block was the minimum foot-print of the installed system. The minimum Scientific Linux install profile requires 1.5GB of storage. This is a concern because it is desirable to network boot the targets. The requirement for a network boot capability drastically reduces the acceptable footprint size as the entire boot image needs to be transferred over the network. Finally, Scientific Linux is a binary distribution targeted for x86 architecture, thus it would not be possible to use as a platform for ARM and PowerPC targets. These concerns drove the decision to use a custom Linux platform built from the ground-up.

* Operated by Fermi Research Alliance, LLC under Contract No. De-AC02-07CH11359 with the United States Department of Energy

jdiamond@fnal.gov

LINUX "FROM SCRATCH"

The Linux from Scratch website [5] was chosen as the model for the attempt to build a custom Linux platform from source. Linux from Scratch is an on-line resource that explains in step-by-step fashion how to build an entire Linux operating system from source. First the user partitions the target's disk on the build system. Then the necessary source packages (about three dozen) and patches (another dozen) are downloaded onto the build system and unpacked into a special "chroot environment". Then the toolchain packages are built and installed – not a trivial task in its own right. The toolchain is then used to build a complete Linux system from inside of the chroot environment - kernel, boot-loader, shell, etc. Once the system is in place the boot-scripts are installed and configured. Finally, the boot-loader is installed on the target CF disk and an attempt is made to boot the target (things rarely work on the first try).

As illustrated above, the process is exhausting and can take several days to complete. It should be noted that the target platform was x86 so a cross-compile toolchain was not attempted (ARM targets came later). Once developed the root filesystem could be used to boot any target with a similar hardware architecture.

The root filesystems produced with this method were in the range of two to three hundred megabytes. One significant drawback is that a boot image of this size is still too large to network boot. The decision was made to store the kernel, boot-scripts and network configuration on a Compact Flash (CF) disk installed with each target node and the root filesystem on a common network filesystem that could be mounted by each target at boot time. The network filesystem also contains the application programs, libraries and data areas.

One issue that was noticed early on was the failure rate of the CF disks. CF disks are susceptible to failure after many write-cycles. As noted above, application data is written to a network filesystem instead of the CF disk. To reduce the CF write cycles the /tmp and /log partitions were moved to a RAM disk and the system log was redirected to a syslog server. These efforts were able to mitigate the failure rate of the CF disks.

Another significant challenge with this platform was maintainability. Installing new software packages in the root filesystem was easy but deploying to targets already in the field meant physically replacing the CF disks. Furthermore, the root filesystem installed on the network filesystem would creep away from its original state over time without version control system in place.

ISBN 978-3-95450-148-9

CONTROL SYSTEM INTEGRATION

At first the support libraries available for integrating with the Fermilab accelerator controls system (ACNET) [6] [7] only supported VxWorks. Package Exchange Protocol (PEP), a light weight UDP protocol was developed to bridge communication between Linux targets and VxWorks targets (Fig. 1). Support software that ran on a VxWorks target was deployed to bridge communication between ACNET and Linux targets deployed in the field. This method was successful and supported many, but not all of the features expected from an ACNET front end node.

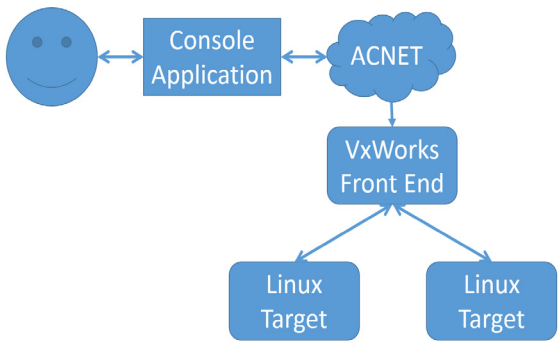

Figure 1. Method for using PEP to bridge ACNET request to Linux Targets.

Later, a suite of software was developed for interfacing Linux front end nodes with ACNET [8]. This ACNET support software is written in Erlang and runs inside of the Erlang virtual machine as its own process (Fig. 2). Using this software the user can write data acquisition drivers in Erlang or using a C++ API. The challenge in this model is how to attach the data acquisition driver, running in its own process to the real-time application running in another process. The decision was made to again utilize the PEP protocol to forward ACNET requests to the real-time application process. In this architecture the communication was done with local sockets instead of over Ethernet.

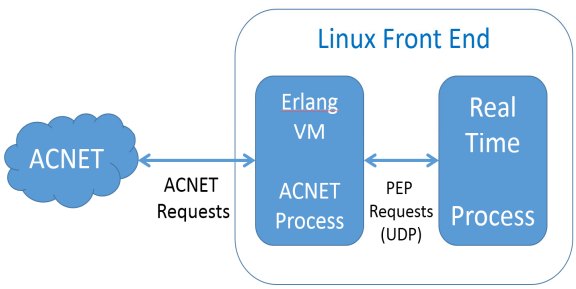

Figure 2. ACNET process utilizing PEP for inter-process communication.

Using this architecture the existing applications using the bridge method were adapted to the new Linux ACNET platform without much effort (Fig. 3). The interface remained the same and the bridge software was moved into the Erlang framework. An alternate method was developed that replaced the PEP protocol with RTAI message queues and shared memory. This method allows large amounts of data to be shared between the ACNET process and the data acquisition process without transferring through a UDP socket.

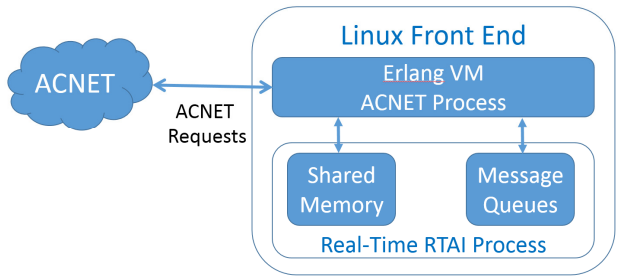

Figure 3. Replacing PEP with RTAI IPC methods.

LOOKING TO IMPROVE

Out of the experience of building a real-time embedded Linux platform from scratch came three desired improvements:

1. A smaller footprint operating system that could be packaged into a network-deliverable boot-image
2. An automated system for building kernel and root filesystem that could support multiple target architectures and hardware configurations
3. An integrated system for building and deploying application software

To address these improvements an effort was begun to look to the open source community for a “best-practice” approach to deploying an embedded Linux platform.

BUILDROOT

Buildroot is an open source build system with a menu-driven configuration tool (similar to the Linux kernel build system) that completely automates this process (Fig. 4). It supports ucLibc [9], a low-footprint alternative to the GNU standard C library and Busybox [10], which combines many of the standard UNIX utilities and a shell into a single low-footprint executable.

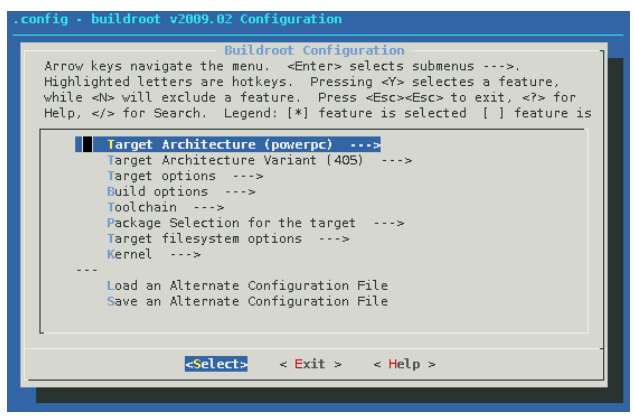

Figure 4. Buildroot configuration menu.

ISBN 978-3-95450-148-9

In addition to providing a kernel, shell and basic utilities, hundreds of software packages are supported. The user simply selects which software package to include in the boot-image and the build system downloads, unpacks, configures, compiles and installs it. The Buildroot configuration system includes an option to combine the kernel and root filesystem into a single bzImage at the end of the build process. A boot-loader and this bzImage file are the only components needed to boot an embedded Linux target.

The first build usually takes a couple of hours, depending on the processing power and network bandwidth of the build machine. This is primarily because of the sheer size of the tool-chain and kernel builds. Once these packages are built, subsequent builds can be done in a matter of seconds.

The footprint depends in large part on the choice of software packages and system libraries. Using ucLibc, Busybox and a streamlined kernel the resulting footprint was 3.5 megabytes – a 99% reduction in size from the existing Linux platform. When the build configuration was expanded to include Erlang, ACNET support libraries and application software the footprint grew to 20 megabytes. With this footprint size, the entire kernel and filesystem could be downloaded by the boot-loader and loaded into a RAM disk on start-up. In order to store the network configuration locally, the /etc partition was all that was left to store on the CF disk. The boot-time of one target was clocked at under nine seconds from power-on to login prompt.

REVISION CONTROL

The Buildroot project is maintained in a public Git repository. A Fermilab clone of the master branch was made and is merged with the project's repository when upgrades or new packages are desired. Custom Buildroot and Kernel configurations for each target are maintained in the repository. Custom application packages are maintained in the repository as well.

APPLICATION PACKAGES

Buildroot supports hundreds of software packages out of the box as well as custom packages. A Buildroot package consists of two parts: a menu description file and a Makefile that is included as part of the build process. Buildroot uses variables defined in the Makefile to find dependencies, to find out where to download the software from and what specific configuration and build options to use.

Dozens of custom software packages were developed for applications and support libraries specific to Fermilab. Buildroot downloads the software directly from a version control server using a specific tag or branch identified in the Makefile.

NETWORK BOOT

The GRUB [11] boot-loader supports downloading the bzImage file from a network filesystem at boot time. GRUB must be built with the support for the network adapter on the target. The network configuration is stored in GRUB menu.lst file, which resides on the flash disk's boot partition.

APPLICATION BUILD ENVIRONMENT

All software that runs on the target must be built with the toolchain that Buildroot builds as the first step in the build process. A set of Makefiles was developed that could be included in a developer's C/C++ project that builds using this toolchain. Deploying to remote targets is supported using scp and ssh or by mounting a remote filesystem accessible from the target and the build machine.

ARM SUPPORT

Recently, Buildroot was utilized to deploy a real-time embedded Linux platform for the ARM Cortex A-9 Hard Processor System on the Altera Cyclone V FPGA. Cross compiling was done by specifying the pre-built Linaro ARM toolchain [12] as an external toolchain. A community kernel configuration file for the Cyclone 5 FPGA SOC, available through RocketBoards.org [13], was utilized to build the Linux kernel.

One notable difference between ARM and x86 targets is that they use different boot loaders. The x86 targets use GRUB and the ARM targets use U-Boot [14]. Like GRUB, U-Boot is a flexible boot loader that allows for network booting. Another difference due to U-Boot though is that it is unable to boot a kernel image that is integrated with a root filesystem. Thus, Buildroot was configured to output the kernel and the root filesystem as separate files. Also, ARM based Linux systems require a device tree file. Fortunately a device tree file for the Cyclone 5 FPGA SOC target was also available pre-made from RocketBoards.org.

RESULTS

Buildroot has been successfully used to deploy real-time embedded Linux systems in the Fermilab accelerator control system for several projects running on x86 and ARM hardware with applications such as magnet quench protection, power supply regulation and beam instrumentation. An effort is currently underway to extend support for legacy PowerPC targets such as the MVME 5500. We have confidence that adding support for additional targets can be done with minimal effort thanks to the efforts of the open-source embedded Linux community.With Buildroot we were able to reduce the operating system footprint and introduce version control to the platform build process. We have found that managing multiple architectures and hardware configurations much easier than building platforms from source would be.

ISBN 978-3-95450-148-9

ACKNOWLEDGEMENTS

The ACNET Erlang framework was implemented by Rich Neswold, Dennis Nicklaus, and Jerry Firebaugh from the Femilab's Controls Group.

REFERENCES

[1] http://buildroot.uclibc.org/
[2] https://www.scientificlinux.org/
[3] C. Briegel, J. Diamond, "Acsys Camera Implementation Utilizing an Erlang Framework to C++ Interface", ICALEPCS (2013); San Francisco, CA, USA.
[4] https://www.rtai.org/
[5] http://www.linuxfromscratch.org/
[6] J. Patrick, "ACNET Control System Overview," Fermilab Beams-doc-1762-v1, http://beamdocs.fnal.gov/AD-public/DocDB/ShowDocument?docid=176
[7] K. Cahill, L. Carmichael, D. Finstrom, B. Hendricks, S. Lackey, R. Neswold, J. Patrick, A. Petrov, C. Schumann, J. Smedinghoff, "Fermilab Control System," Fermilab Beams-doc-3260-v3, http://beamdocs.fnal.gov/AD-public/DocDB/ShowDocument?docid=326
[8] D. Nicklaus, "An Erlang-based Front End Framework for Accelerator Controls," ICALEPCS (2011); Grenoble, France.
[9] http://uclibc.org/
[10] http://www.busybox.net/
[11] https://www.gnu.org/software/grub/
[12] https://www.linaro.org/
[13] http://www.rocketboards.org/
[14] http://www.denx.de/wiki/U-Boot

ISBN 978-3-95450-148-9