Faculdade de Ciências da Universidade de Lisboa

Instituto Superior Técnico

DARIO: Distributed Agency for Reliable Input/Output

Project FCT POSC/EIA/56041/2004

Bullet-Linux: a Real-Time Platform for Industrial Embedded Systems

DARIO Technical Report RT-08-03

J. Craveiro, J. Rufino, C. Almeida, R. Covelo, P. Venda

March 2008

Faculdade de Ciências da Universidade de Lisboa

Instituto Superior Técnico

Bullet-Linux: a Real-Time Platform for Industrial Embedded Systems

To be submitted for publication: please do not distribute

Technical Report: DARIO RT-08-03
Authors: J. Craveiro, J. Rufino, C. Almeida, R. Covelo, P. Venda
Date: March 2008

This work was partially supported by the FCT through Projects POSC/EIA/56041/2004 (DARIO) and the Large-Scale Informatic Systems Laboratory (LASIGE).

Bullet Linux: a Real-Time Platform for Industrial Embedded Systems

João Craveiro*, José Rufino*, Carlos Almeida†, Rui Covelo† and Pedro Venda†
*Faculdade de Ciências da Universidade de Lisboa,
Campo Grande - Bloco C8, 1749-016 Lisboa, Portugal.
Tel: +351-217500254 - Fax: +351-217500084.
†Instituto Superior Técnico - Universidade Técnica de Lisboa,
Avenida Rovisco Pais, 1049-001 Lisboa, Portugal.
Tel: +351-218418397 - Fax: +351-218417499.

***Abstract*— In the field of industrial embedded systems one has to cope with (on the one hand) the scarcity of resources typical in such devices, and (on the other hand) the diversity of existent hardware (processors, network interfaces). A balance between how much featured an operating system solution should be to address both issues must be reached. In that sense, we are evaluating Linux as a solution for embedded systems that addresses this balance; for such, we show the genesis of such a solution, the Bullet Linux, in three main vectors: kernel, system library, and system tools. We also extend the suitability of Bullet Linux to embedded systems with real-time requirements, through the integration of RTAI, and to architectures presenting spatial and temporal partitioning, like ARINC 653, a standard for aeronautical and, hopefully, aerospace applications.**

I. Introduction

Embedded systems, and particularly industrial embedded systems, are often physically small in order to fit in small places, to be easily carried or to be hidden. One of the problems the world of embedded systems faces is the lack of computing resources relatively to larger computers. These small systems can also be composed of a wide variety of hardware devices, like different CPU architectures, different network interface controllers, different types of mass storage and a set of other application-specific devices. A wise choice of operating system must be made in order to balance functionality and available resources.

In this work, a case is made for a GNU/Linux system (the Bullet Linux) to serve this purpose, and the foundations are laid to extend the suitability of this system to more demanding environments.

The paper is organised as follows: the Bullet Linux and its build process are described in Section II, and the size of different configurations is analysed against a typical GNU/Linux distribution in Section III. The use of Bullet Linux is then studied, in Section IV), for its suitability to satisfy real-time requirements, including the incorporation in an architecture compliant with the ARINC 653 standard. Conclusions are presented in Section V.

II. Bullet Linux

Linux is an open source operating system kernel available free of charge and maintained by developers from all the world. The source code is accessible for everyone and people are encouraged to contribute with their own code. For this reason, the Linux kernel is extremely portable between computer architectures and supports a massive variety of hardware devices. And there's always space for more.

There are some GNU/Linux distributions available for embedded systems but none of those are exactly what we need. Some are commercial or targeted at a specific type of device. Others simply have too much unnecessary features or already have their own kernel modifications. Balancing that and the ease of setting up a new Linux operating system, we analysed how to build a specific (and further ahead, real-time ready) version of a Linux operating system targeted for embedded systems and applications. The main vectors for achieving an effective balance between functionality and available resources are: configuration of the Linux kernel, use of a smaller system library, and provision of the standard Unix utilities and tools in a more resource-efficient way.

A. Linux kernel 2.6

The Linux kernel 2.6 brought fundamental improvements over 2.4 for embedded system and towards real-time systems. A new preemption system allows the preemption of kernel tasks, where user applications are no longer locked until the end of all pending system calls before they can continue executing. This significantly reduces the latency of user applications and increases the overall system responsiveness. Along with a new $\mathcal{O}(1)$ scheduler, the preemption of kernel tasks resulted in a performance improvement and better user interactivity.

An increased modularity allows one to easily select the smallest set of features required for each system, avoiding unnecessary code. For systems with limited resources this is very important. Additionally, the added support for a wider variety of hardware devices, computer architectures and the improved build tools help enhance the pace of development of the kernel itself. This flexibility makes Linux, and specially Linux 2.6, a good choice for real-time embedded systems. An example of the configuration process for the Linux kernel is depicted in Figure 1 [1].

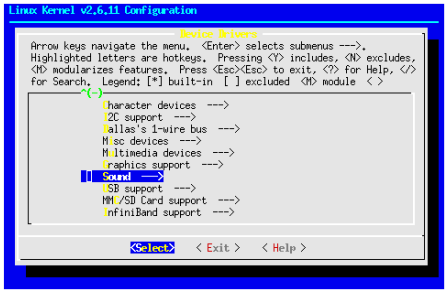

Fig. 1. Kernel configuration screen

Exploiting the configurability of the Linux kernel, it became possible to produce a small kernel image for Bullet Linux, as shown in Figure 2.

Kernel	Generic	Bullet Linux
Size	1690K	596K

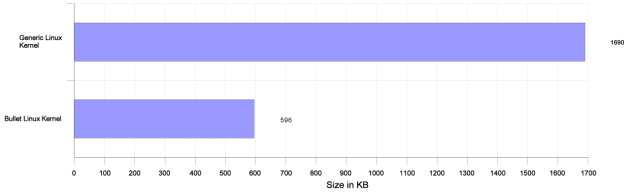

Fig. 2. Size comparison between a generic Linux Kernel and Bullet Linux Kernel

The kernel for Bullet Linux is built from a standard, unpatched source tree, exactly as distributed by the developers. The absence of customised patches ensures easier upgradeability and less compatibility issues between different versions.

Independently to building a working minimal Linux distribution for embedded systems, work was made to account how much size could be spared or required depending on how the kernel is configured. Thus, a handful of kernel configuration profiles were surveyed. The base, what we'll call here a "bare kernel", is a kernel with the bare minimum to function — basic device support, no network or any kind of peripheral connection, no peripherals or multimedia device drivers, and support for no more than the basic file system types.

The additional features that were tested for kernel size footprint were:

- network support (with basic/generic network interface drivers);
- 802.11 stack support (no WPA though, only WEP);
- Bluetooth support;
- USB (Universal Serial Bus) support;
- IrDA support (including some infrared dongle drivers).

The results obtained are that on Table I. For comparison sake, they refer only to `vmlinuz` — the statically linked executable file that contains the Linux kernel image.

B. A small system library

One of the most important components of a UNIX like system is the system library. The most used system library

TABLE I
KERNEL (VMLINUZ) SIZE COMPARISON

Configuration	Size
Bare	596K
Bare + Network	872K
Bare + Network + 802.11	911K
Bare + Network + Bluetooth	939K
Bare + Network + USB	974K
Bare + Network + IRDA	1010K

is the C library GNU libc. However, there are alternative design options to this library that are more appropriate for embedded systems. uClibc is a C library specially developed for embedded systems, which we found adequate among the available options. It features almost all GNU libc functionality but it's small size makes it appropriate for system with low resources. This was accomplished by rewriting the code with size optimisations in mind and by modularising some functionalities which allows the configuration of the uClibc library adapting it to the requirements of the target system.

The use of uClibc allowed Bullet Linux to keep a small size, when comparing against the use of a standard GNU libc. Figure 3 shows the immediate advantages brought by uClibc. The compilation of GNU/Linux utilities and tools against uClibc brings, then, added benefits.

Library	GNU libc 2.4	uClibc 0.9.29
Size	1828K	448K

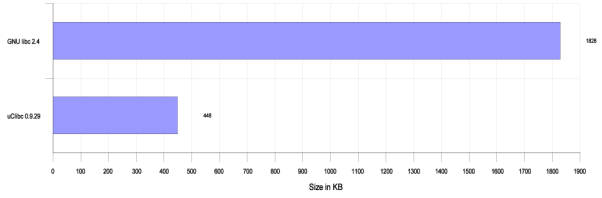

Fig. 3. Size comparison between a standard GNU libc and uClibc

C. GNU/Linux utilities and tools

To complete a Linux based operating system, we need some well known tools on Linux and UNIX systems that everyone expects to find. For this purpose we selected Busybox as a set of those tools bundled together in a single file. These tools were rewritten to be smaller than their original counterparts. This is accomplished by code optimisations and by the absence of some of the features offered although maintaining the most important functions.

Busybox is also highly modular and configurable, as shown in Figure 4.

Even using shared libraries, standard GNU tools can use a lot of space, which is a real problem when dealing with resources shortage. Busybox by itself allows a notable size gain and, when compiled against uClibc extends that margin even further. Figure 5 shows that size gain obtained from the usage of Busybox.

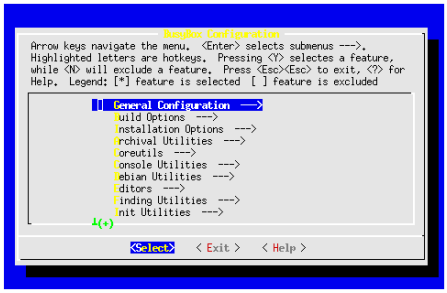

Fig. 4. Busybox configuration screen

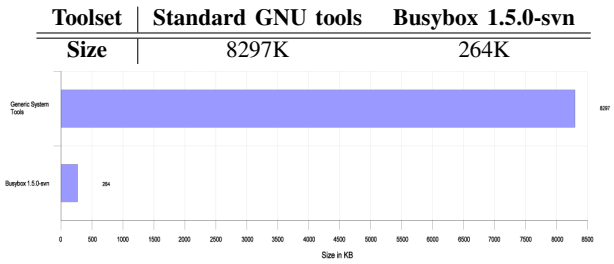

Fig. 5. Size comparison between a set of standard GNU Linux tools and busybox

D. Building process

By putting together a specially configured Linux kernel 2.6 for our prototype systems, a smaller system library and a restricted set of system tools, it became possible to build an entire GNU/Linux operating system that can fit on an 1,44MB floppy disk.

The kernel is compiled from unpatched sources with a specific configuration for the existing devices and interfaces of the prototype systems (i386 based, ethernet network, no hard disk drive). Buildroot is used to facilitate the configuration and build process of the uClibc system library and the Busybox toolset; it configures builds and prepares the cross compiler environment for the later build of the system library and toolset (an example of the configurability of Buildroot is shown in the Figure 6). This cross compiling environment is necessary because the target architecture for Bullet Linux may be different from the architecture of the build system. The system library and toolset are also tailored to be built as small as possible, while maintaining all the important functionalities.

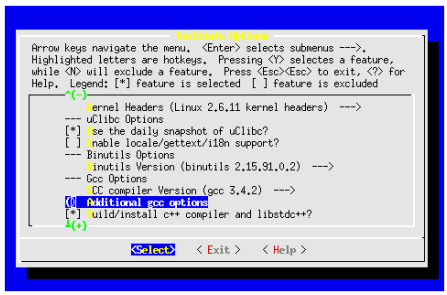

Fig. 6. Buildroot configuration screen

Bullet Linux is extremely customisable, due to its main components' flexibility and modularity. The kernel is configurable and very modular, buildroot permits the customisation of the build environment and partial configuration of the system library and toolset. Further refining of the system library and toolset is also possible by doing so directly from their respective source tree.

III. Overall size analysis

As referred before, one of the main purposes of Bullet Linux is to be a small operating system. A targeted and well defined functionality with little or no overhead or extras. Bullet Linux was built with Kernel version 2.6.19.2, uClibc 0.9.29 and Busybox 1.5.0-svn (2007-01-24) compiled against uClibc 0.9.29.

All the main components of Bullet Linux exist on a desktop distribution, but a desktop distribution is far from being comparable to Bullet Linux. The size gain of Bullet Linux can be analysed by comparing each of its components individually with the equivalent in a desktop distribution. Typically, a desktop distribution is built with standard or lightly patched Kernel compiled with a modular approach; a modular Linux Kernel is composed of an image plus a set of files that correspond to different modules. Modules are loaded into memory only if considered necessary by the system or the user. A typical modular kernel has an image size slightly above 1.5M and a set of modules with about 15M. The system library is a fully featured GNU libc 2.X (libc 6) along with many other smaller less generic libraries. The set of tools in a desktop distribution are compiled against it's glibc and occupies the biggest slice of storage space: it can reach several gigabytes.

Globally, Figure 7 summarises the analysis of size in two distinct situations: a generic Linux distribution and Bullet Linux.

Linux Kernel	Generic	1690K
	Bullet	596K
System Library	glibc 2.4	1828K
	uClibc 0.9.29	448K
System Tools	Generic	8297K
	Busybox 1.5.0-svn	264K

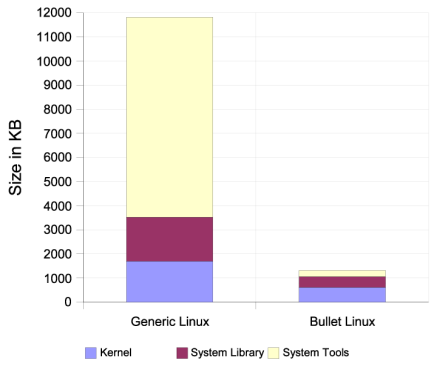

Fig. 7. Overall size comparison

IV. Guaranteeing real-time operation

A. Bullet Linux with RTAI

RTAI (Real-Time Application Interface) is an open source project for providing a Linux-based operating system with real-time capabilities. RTAI employs a modified version of ADEOS, which proposes a nanokernel hardware abstraction layer (HAL) approach. In this approach, the HAL provides the mechanism to pass interrupts to a real-time operating system (not implementing real-time capabilities on its own) [2]. It also presents a module-oriented structure, with its functionality spread among a few kernel modules.

Experimental build: To start the construction of a real-time Linux for embedded systems, Bullet Linux was built with a slightly different set of tools (namely, their versions) and requirements: Linux kernel 2.6.19.2 (submitted to the HAL patch distributed with RTAI), uClibc 0.9.28, and Busybox 1.2.2.1 (compiled against uClibc 0.9.28, and with a few additional required built-in applets).

Besides that, the only notable difference towards the Bullet Linux build described in Section II was the inclusion of the Bourne-again shell (bash) as an independent component; this was done exclusively in order to run some RTAI example scripts that are not fully compatible with the default shell provided by Busybox (ash). The version of RTAI used was 3.5, and was compiled against the same uClibc 0.9.28 toolchain as Busybox. The combined solution was verified as working with a specific test included in the RTAI workbench. This test verifies task switching latency by means of measuring the difference between the expected and actual switch times of a task as performed by the RTAI scheduler.

B. Bullet Linux in a partitioned architecture

The AIR architecture, based on the ARINC 653 standard (defined for aeronautical applications which is being adapted to aerospace applications as well), specifies an architecture presenting spatial and temporal segregation, and allows for the use of different operating systems in different partitions [3]. Porting applications from Linux to one of the RTOS one might be using (RTEMS, eCos, VxWorks, etc.) can be a complicated task, and definitely not an error-free one [4]. To address this portability issue, we are evaluating the approach of having one partition of such a system run GNU/Linux. That partition interfaces with the AIR PMK (Partition Management Kernel) through a subset of the AIR APEX (Application Executive) Interface. This integration is shown in Figure 8.

In this scenario, existent applications for GNU/Linux can be used or created without the further effort of having to port them to a particular RTOS. In the scope of this integration, though, we are currently evaluating if the eventual employment of a non-real-time GNU/Linux solution will not contaminate the real-time properties of the whole system.

C. Control of event handling

In most embedded systems, like Bullet Linux, external events trigger an interruption; interrupts are immediately processed, postponing any other processor activity (except perhaps, other interrupts). Uncontrolled interrupt processing is especially dangerous in applications with stringent time and fault constraints. In real-time systems, the standard approach is to accommodate the event generation worst case scenario into the scheduling conditions. Because of the simplifications made, this is in general too pessimistic and the system ends up with an over-dimensioned system [5]. Following a previous work [6], Bullet Linux should integrated a specialised component to supervise and limit the rate of interrupt generation.

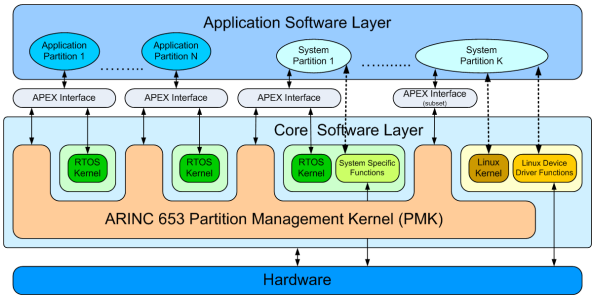

Fig. 8. Overview of the intended incorporation in the AIR system architecture

V. Conclusions

In this work, we reach to an understanding that Linux kernel can be used as the basis for a fully functional operating system for embedded systems with scarce storage resources, with the (also assessed) possibility of adding real-time capabilities through existent real-time executives (RTAI, Xenomai). This solution is also being envisioned to be integrated in a segregated (in time and space) system, so as to be able to run non-critical applications existent for GNU/Linux without having to port them to the RTOS used in the other partitions.

Acknowledgment

This work was partially supported by EU and FCT through Project POSC/EIA/56041/2004 (DARIO) and through the FCT Multiannual Funding Program. Project DARIO Web site — http://pandora.ist.utl.pt/projects/dario.

References

[1] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, 2nd ed. O'Reilly, December 2002.

[2] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application," *Nuclear Science, IEEE Transactions on*, vol. 55, no. 1, pp. 435–439, Feb. 2008.

[3] J. Rufino, S. Filipe, M. Coutinho, S. Santos, and J. Windsor, "ARINC 653 interface in RTEMS," in *Data Systems in Aerospace (DASIA 2007)*, May 2007.

[4] L. M. Kinnan, "Application migration from Linux prototype to deployable IMA platform using ARINC 653 and Open GL," *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, Oct. 2007.

[5] K. Sandstrom, C. Eriksson, and G. Fohler, "Handling interrupts with static scheduling in an automotive vehiclecontrol system," in *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*. Hiroshima, Japan: IEEE, Oct. 1998, pp. 158–165.

[6] M. Coutinho, J. Rufino, and C. Almeida, "Control of event handling timeliness in RTEMS," in *Proceedings of the 17th IASTED International Conference on Paralel and Distributed Computing Systems - PDCS 2005*. Phoenix, Arizona, USA: IASTED, November 2005.