Linux based Operating System Proposal for the Acquisition and Processing of Data in Embedded Devices

M.Eng. Roberto Alejandro Espí Muñoz
Digital Medical Technology ICID
202 and 17 Playa, La Habana, Cuba

ABSTRACT

The current work presents a proposal for a data acquisition base using the CID 300/9 device, an industrial motherboard developed at ICID Cuba. In the beginning a brief view on the necessity of using novelty techniques in the support of medical software is presented. The design guidelines in the optimization of the Linux based operating system are presented as well as the architecture for a Middleware variant for the target device running specific software projects that must execute in medical attention and monitorization environments. Ending the report, measurements and tests to the developed components in the boot up sequence as well as the Middleware layer are presented, thus validating the presented proposal.

Keywords:

Communication, embedded devices, Linux, medical equipment, micro-controllers

1. INTRODUCTION

In the medical equipments field there are a considerable number of projects and enterprises specialized in the development of integral solutions with a high degree of added value capable of providing coverage to specific fields in the treatment and diagnosis of pathologies as well as intensive care.

The Digital Medical Technology Enterprise ICID in Cuba provides a group of solutions that span throughout a wide spectrum of medical disciplines. Among them, one can find equipment specialized in cardiovascular, pulmonary and muscular conditions as well as intensive care, primary attention and critical attention to mention a few. The majority of these solutions must acquire data from medical sensors, send data or control signals to acting devices and make some type of processing to the handled information. The term used for this type of software is **Middleware** [1].

In a brief outline, Figure 1 shows the idea.

Middleware logic is currently integrated in devices developed by the ICID Medical Enterprise such as the following:

—**CardioDef** :: Stationary defibrillator
—**Doctus** :: Medical parameters monitor
—**CardioCID** :: Electrocardiography monitor

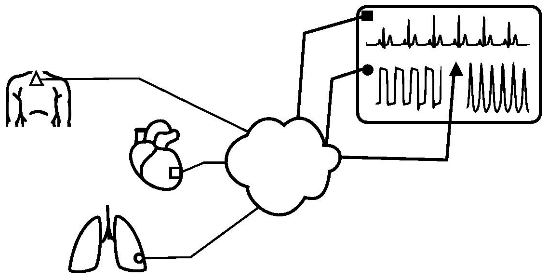

Fig. 1. Parameter values are measured from underlying sensors to a middle communication layer, they are processed and displayed to the user at a later time.

—**HiperMax** :: Non-invasive pressure meter
—**CardiHogar** :: Remote monitor of cardiac parameters

This design however is not exclusive to medical equipment, it comes, in part, from solutions found in Supervision Control and Data Acquisition (SCADA) software[2], which focus in providing automation and control in industrial processes.

In the year 2009 ICID developed a processing base (industrial motherboard) from a native design. It has provided the devices the company develops with a processing unit. In Figure 2 an image of said device is portrayed.

2. METHODS AND MATERIALS

The structure of the software layer in the CID 300/9 can be visualized as a chain of components by which the devices have to pass by until they arrive to the point where their upper layer application can be executed. In Figure 3 these components are displayed. Due to the inherent design of computing devices, every component in order for it to be executed must be copied from a storage medium to RAM where the processor will execute every one of it's instructions. Thus a key strategy in the design and construction of operating systems for embedded devices is the reduction of their size or weight in storage.

Bootloader

Among the existing bootloaders, **barebox[3]** is one of the more customizable and efficient there is. It's capable of booting Linux

Fig. 2. CID 300/9 industrial motherboard. **µC:** ARM-920T, **CPU:** 400 MHz, **Ports:** 4 USB, 1 Ethernet, 2 Serial**, RAM:** 128MB**, Audio:** 2 3.5mm input/output, **SD Card:** 1 port, **Video:** Touchscreen and LDVS port, **OS:** Linux based

Fig. 3. Boot chain of a Linux based operating system.

CONFIG_ARCH_S3C24xx=y	CONFIG_AUTO_COMPLETE=y	#CONFIG_CMD_CD
CONFIG_MACH_CID300_9=y	CONFIG_DYNAMIC_CRC_TABLE	#CONFIG_CMD_MOUNT
CONFIG_S3C_NAND_BOOT=y	#CONFIG_ERRNO_MESSAGES	#CONFIG_CMD_UMOUNT
CONFIG_AEABI=y	#CONFIG_TIMESTAMP	#CONFIG_CMD_BOOTZ
#CONFIG_CMD_ARM_CPUINFO	CONFIG_CMD_EDIT=y	CONFIG_CMD_TIMEOUT=y
#CONFIG_ARM_EXCEPTIONS	CONFIG_CMD_SLEEP=y	CONFIG_CMD_PARTITION=y
#CONFIG_CMD_MEMORY	CONFIG_CMD_LOADENV=y	#CONFIG_CMD_HELP
#CONFIG_LOCALVERSION_AUTO	CONFIG_CMD_EXPORT=y	#CONFIG_CMD_DEVINFO
#CONFIG_BANNER	CONFIG_CMD_PRINTENV=y	#CONFIG_SPI
CONFIG_TEXT_BASE=0x33F80000	CONFIG_CMD_READLINE=y	#CONFIG_MTD_WRITE
CONFIG_BROKEN=y	#CONFIG_CMD_LS	CONFIG_MTD_RAW_DEVICE=y
CONFIG_MALLOC_TLSF=y	#CONFIG_CMD_RM	#CONFIG_UBI
CONFIG_PROMPT="X:/> "	#CONFIG_CMD_CAT	#CONFIG_DISK
CONFIG_CBSIZE=256	#CONFIG_CMD_MKDIR	#CONFIG_USB
CONFIG_GLOB=y	#CONFIG_CMD_RMDIR	#CONFIG_VIDEO
CONFIG_HUSH_GETOPT=y	#CONFIG_CMD_CP	#CONFIG_MCI
CONFIG_CMDLINE_EDITING=y	#CONFIG_CMD_PWD	

Fig. 4. Proposed configuration options for barebox. Options starting with # are removed. Options ending with *y* are included.

based operating systems and allows running basic tasks for using the system such as working with files, modifying environment variables, running applications and interacting with devices.

Generally a bootloader must be configured to make a direct and quick jump to the next step in the boot chain: the Linux kernel. It's not of much use that when booting a device there's a delay waiting for user confirmation for a predictable, well established and recurring next step. A candidate configuration for the execution of this type is shown in Figure 4.

In the proposal one can observe how unnecessary functions at boot time are removed.

Linux

The Linux kernel is a project actively maintained by a vast community of users worldwide. It supports 24 known architectures and a greater number of micro-architectures[4]. It can be configured and optimized for different scenarios (embedded devices, desktop computers, servers, laptops). Linux can insert new functionality via

```
#!/bin/sh

# Start all init scripts in /etc/init.d
# executing them in numerical order. #

cd /opt/elasticnodes
./elasticnodes -qws &

modprobe ohci-hcd
modprobe usb_storage
modprobe usbserial
modprobe evdev

exit 0
```

Fig. 5. Start script of an example application based on Qt 4.8.1 framework. In turn services and applications are initialized. It starts with the visual application (elasticnodes), and then execution is forked "**&**". At the end the necessary modules are loaded in the kernel.

modules at runtime. In order for it to be configured a configuration menu is provided similar to that of the bootloader.

Some of the chosen configuration options are:

—**CONFIG_PRINTK** is not set : Disables printing messages on screen.
—**CONFIG_SLAB=y** : This way of handling objects in memory is the fastest after analyzing test results.
—**quiet** : Disables all messages printed by the kernel.
—**lpj=xxxxxx** : Sets a calculated value every time the kernel starts. Specifying it removes the necessity for the kernel to generate it.

Fs + services

Choosing the correct type of filesystem influences notably in boot time and execution of the operating system. Two candidates were identified:

—**External media** (SD Card, HDD, USB Flash)
The recommended choice for this type of media is a system without journaling such as **ext2**.
—**Internal media** (NAND, NOR)
The recommended filesystem for these cases is **UBIFS.**

In the service layer a strategy for reducing the number of services that are instantiated in order, is establishing a single script file with a set of instructions that are to be executed, and also running in parallel the loading of less prioritized devices. The example in Figure 5 shows this desing guideline.

Applications

The most notable optimization in the application layer of an operating system is the inclusion of it's dependencies inside the executable. Normally in an operating system a group of programs coexist which depend heavily between them sharing functionality and libraries. In Windows systems these are known as **.dll**. In Linux the same principle exists. When executing an application, if it is compiled with the shared libraries option, it's start up becomes slower as well as the execution of it's internal functions since it has to search for sources outside the binary. This scenario is known as dynamic linkage[5].

The counterpart of this scenario is known as static compilation. It allows the inclusion of all provided functions and dependencies within the same codebase removing the necessity of external calls.

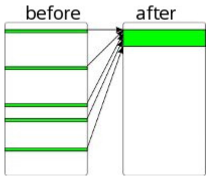

Fig. 6. Binary reordering. To the left the unordered location of start up functions are portrayed in green. To the right the same functions are displayed according to the final location.

```
typedef boost::mpl::vector<
        row<byte0, has_sync1, byte1, &self::procByte1>,
        row<byte1, is_any, byte2, &self::procByte2>,
        row<byte2, is_any, byte3, &self::procByte3>,
        row<byte3, has_sync2, byte4, &self::procByte4>,
        row<byte4, has_sync2, byte5, &self::procByte5>,
        row<byte5, has_sync2, byte0, &self::procByte6>
> transition_table;
```

Fig. 8. Equivalent code for the data stream. C++ code for working with the data stream is shown. Each row specifies a certain byte in the stream which calls a unique function upon validation.

The following are two examples on building binaries with static linkage:

—Applications built using autotools:
`./configure --enable-static --disable-shared`

—Applications built using the Qt framework:
`QMAKE_LFLAGS = -Xlinker -Bstatic $$QMAKE_LFLAGS`

Application load time can also be improved using a technique called **reordering[6]** that allows changing the internal structure of a binary in a way that in the first bytes the necessary functions for displaying the visual application are located. Graphically reordering could be shown as Figure 6.

Middleware

As a solution to the communication layer for the CID 300/9 a Middleware is proposed taking into account design aspects such as:

—Obtaining data from multiple sources

—Data handling using variable data streams and protocols

—Dispatch to multiple interested object

—Variable batch handling

—Efficiency in transmission

Device

The Middleware concept closest to the connected sensors and actuators is the **device**. A device is capable of executing 4 basic functions: **open**, **read**, **write** and **close**.

StateMachine

Is an abstract interface to various types of state machines. Each specific type carries with it an intrinsic protocol analyzer for each type of received data stream. Internally it uses a concept known as finite state filters[7]. An example showing how a 6 byte EKG data stream can be analyzed by the filter is provided in Figure 8.

Table 1. Size of the components

component	without optimizations	**optimized**
barebox	191.6 KB	85.2 KB
Linux	2.03 MB	964.8 KB
fs	39 MB	16.8 MB

Table 2. Boot time for ICID projects

project	**base system**	**time in seconds**
T50	CID 300/9	13.4
Doctus 9	CID 300/9	28
CardioDef 2	CID 300/9	9
Doctus VII	MS Windows Embedded	33
Proposal	CID 300/9	4.4

Dispatcher

This component is in charge of notifying interested objects upon arrival of a new value from devices. It uses the same event for dispatching to each object the new value as opposed to other approaches of using separate events.

3. RESULTS AND DISCUSSIONS

With the objective of validating the proposal a group of tests to measure the operating parameters of the base system were designed. In Table 1 the size of the optimized components are displayed.
Measurements were conducted regarding the boot times of the Linux kernel of various projects using the CID 300/9 motherboard and were later compared with the times offered by this proposal. In Figure 9 the boot times for the internal subsystems of the kernel are displayed.
The proposal offers reduced times in the **0.26s** range. Boot times for each project can be summarized in the following way: **T50** (7.2s), **Doctus 9** (9.3s), **Cardiodef 2** (2.59s).
Measurements to the total boot time to various ICID projects were also conducted, most of which use the CID 300/9 motherboard as their processing base. Results are showed in Table 2.
The design and implementation for the middleware communication protocol was also tested. A test case scenario was designed consisting in a server acquiring data at a constant rate of 6 bytes for every 5 milliseconds (data stream generation time) for over 20 hours. The test was conducted on two different architectures, one in PC and the other one in the CID 300/9 device. The results are displayed in Table 3.

Table 3. Middleware performance results.

parameters	**PC-x86**	**CID 300/9-ARM**
Execution time	23h 40min	28h 30min
Data stream size	6 bytes	6 bytes
Recolected data	100641809 bytes	103734325 bytes
Correct data	100640782 bytes	103716583 bytes
Discarded data	1027 bytes	17742 bytes
Accepted samples	16773634	17286097
Discarded samples	171	2957
Algorithm efficiency %	99.99 %	99.99%

4. CONCLUSIONS

The identified optimization techniques applied to the bootloader, kernel, filesystem and applications greatly reduce the boot time of

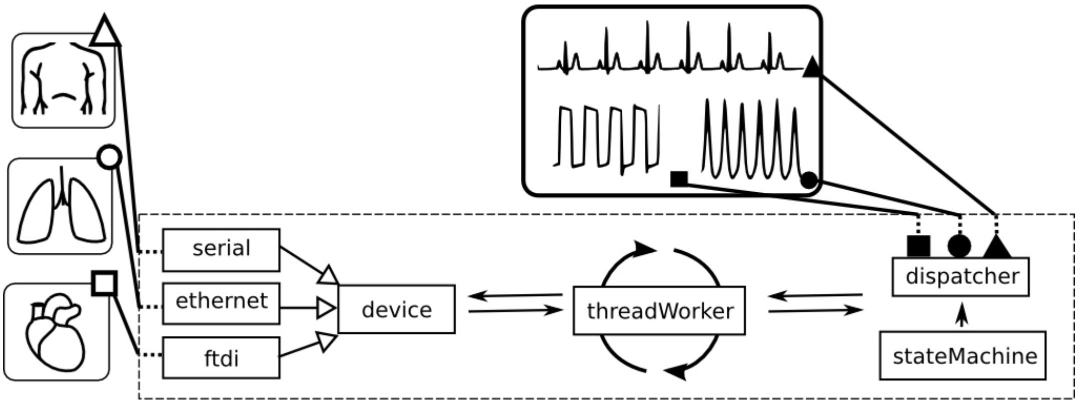

Fig. 7. Proposed Middleware architecture. A **device** represents a physical sensor to which the Middleware is connected. A **threadWorker** in a given period of time obtains a sample from a **device** and processes it with a **stateMachine**. The disassembled value is notified to all interested graphical objects via a **dispatcher**.

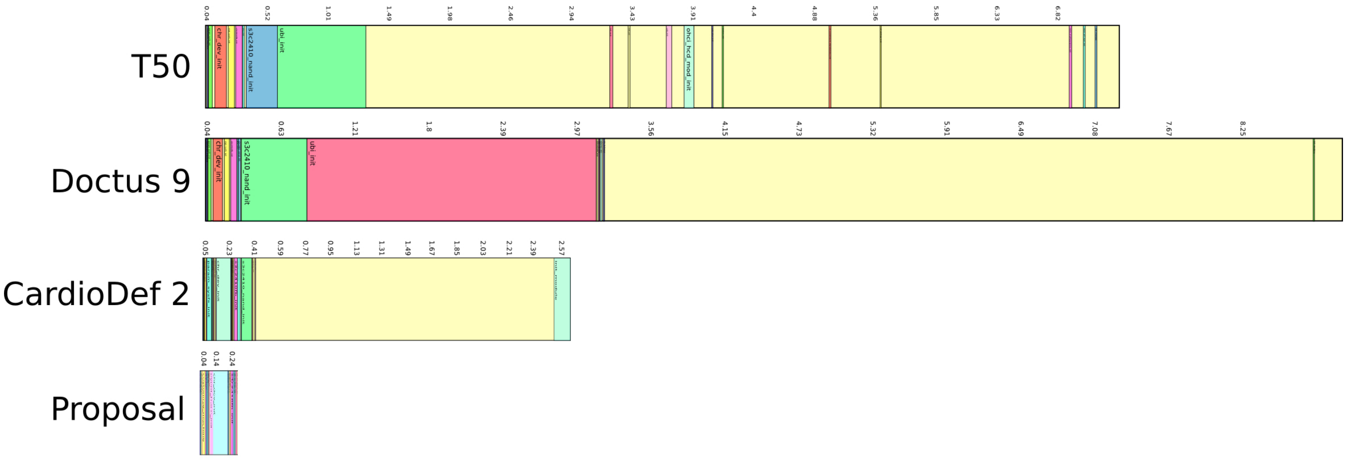

Fig. 9. Measurements to the boot times of the Linux kernel.

the operating system in general and increase the execution speed of each application in particular. The work with each component showed a notable reduction in the size of the executable thanks to the removal of driver support and unnecessary functionalities in every phase of the boot chain. Linux based operating systems are reliable candidates for the development of critical execution embedded applications.

5. REFERENCES

[1] "ObjectWeb - what's middleware," 2013. [Online]. Available: http://middleware.objectweb.org

[2] "SCADA system SIMATIC WinCC," 2012. [Online]. Available: http://www.automation.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/pages/default.aspx

[3] "start [barebox]." [Online]. Available: http://wiki.barebox.org/doku.php

[4] "What is linux: An overview of the linux operating system | linux.com." [Online]. Available: https://www.linux.com/learn/new-user-guides/376-linux-is-everywhere-an-overview-of-the-linux-operating-system

[5] "Shared libraries." [Online]. Available: http://www.iecc.com/linker/linker09.html

[6] M. Kim and O. Kokachev, "Instant startup for application using reduced relocation time and rearranged functions," Apr. 2008.

[7] "Boost.Iostreams." [Online]. Available: http://www.boost.org/