Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programa de Maestría en Computación

Improving Operating Systems for Efficient Multimedia Handling in Embedded Systems

Thesis Proposal
Submitted in partial fulfillment of the requirements for the degree of

Magister Scientiæ en Computación

Author:
Diego Dompe

Advisor:
Ph.D. Francisco J. Torres-Rojas

May 2011

Resumen

Resumen en español

Abstract

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

ONE

INTRODUCTION AND BACKGROUND

> People in the embedded space don't do prototypes. They hack something until it works, then it's done.
>
> xav user on lwn.net

1.1. Introduction

1.2. Background

Embedded Systems are one of the fastest growing markets in the computer industry, with more than 10 billion embedded processors shipped in 2008 [6].

Embedded Software development is a discipline that is younger that software development for other environments like Personal Computers (PCs) or Mainframes. There are some particular issues that transform embedded software development into a complex eco-system:

- It usually requires some level of integration with the hardware that is non-common in other software development areas.

- Many of the hardware integration that is done is very particular and unique, which usually makes the code hard to re-use if isn't designed properly (or if the technology is new and isn't clear how to implement the software solution in a generic way).
- The market for embedded system is so fast-paced that developers are in pressure to deliver working devices as soon as possible. Few companies care about the quality or re-usability of their embedded code, except by hardware vendors or efforts from single open-source minded developers who want to contribute back. For example recent articles regarding status of the contributions to the Linux™ kernel for the ARM architecture the contributions status as a "mess" due many of the different interests and agendas behind it [16].
- Many embedded developers may lack formal training on software development, but instead have an Electronic Engineer (EE) background.

These factors usually means that embedded developers will not likely find or apply the same best-practices that the rest of the software industry use. Is not rare to find embedded projects where each revision of the same product will have a complete new software stack or re-implementation from the previous iteration.

The fact that the code is tailored for specific hardware to the point where it can't be easily re-used goes against Operating Systems (OSs) theory which proposes that the OS should abstract all the software from the hardware implementation [19, p. 29]. However many embedded devices (especially ones with low memory footprint) may use a custom-made OS or use no OS at all (also called an OS-less solutions).

On embedded systems capable enough to support it, Linux has gradually gained acceptance as an OS for embedded systems due his open-source nature and support from embedded processors manufacturers, especially on the System on a Chip (SoC) market. SoCs are the main trend for new product development in recent years due the benefits it brings to new chip development [20].

The advantages that Linux provides for new embedded product development are clear:

- It's a probed OS which has been successfully used in different embedded markets.
- No royalty fees or licenses cost associated directly with it (although some embedded vendors may add royalty fees if you use their custom distributions).
- Wide knowledge base available and many developers are familiarized with it.
- Usually hardware companies provide drivers and support for Linux in their latest devices (for example ethernet or wireless circuits, storage devices, etc.), simplifying the integration of hardware components for an embedded system. This is a critical element for minimizing the time-to-market of a product.

However Linux also presents some disadvantages for the embedded space:

- It's not designed to be a Real Time Operating System (RTOS), which make it not suitable for all type of embedded applications.
- There is a learning curve for RTOS developers, since Linux enforces some concepts that may be foreign to them:
  - Memory space protection: most RTOS do not provide memory space protection.
  - Separation between the kernel and user space applications: This is a side-effect of the memory space protection.
- There are several algorithms used in non-RTOSs like Linux that may affect the performance compared to RTOSs or OS-less solutions:
  - Context Switches: there is a penalty associated on several hardware architectures with the memory protection.
  - I/O Scheduling: the kernel page cache may defer I/O work. Linux offers full control to customized this behaviors for embedded systems, but many developers may be unaware of them.

- Interrupts prioritization: mainline Linux kernel doesn't offer a way to control priorities for interrupt handlers. There are patches to transform interrupt handlers into kernel threads, but they aren't available for all kernel versions and may require specific analysis and tuning to achieve desired results [21].

Despite the short-comings that Linux present versus a RTOS, just the fact of have widely available drivers and software stacks for peripherals on the embedded space usually outweighs these problems. Many SoCs may require complex software stacks to interact with integrated peripherals, such is the case of heterogenous SoCs using a General Purpose Processor (GPP) and a Digital Signal Processor (DSP) or other sort of hardware unit for accelerating specific operations (such as video coding, cryptographic algorithms, etc). Implementing or porting the functionality of these software stacks required to interact with the desired peripheral to an OS-less or other non-supported RTOS is usually non-effective in terms of cost or time.

However many developers that are un-familiar with Linux and/or general purpose OSs design, approach the task of implementing the functionality for their embedded products bypassing Linux functionality. For example the author of this work have found several times kernel drivers for Linux designed just to specifically allow user-space access to the peripherals by any user space application in commercial Original Equipment Manufacturing (OEM)/Original Design Manufacturing (ODM) offerings for embedded development (TODO publish code and put a reference). There are several reasons for developers using this kind of approach:

- Some times the software stack was designed in OS-less environment and ported over to Linux. Developers just try to find a way to get the Linux hardware management out of the way of the software stack.
- Developers are simple un-familiar with Linux design.
- Linux may lack APIs or functionality to handle new scenarios posted by the hardware design. For example ability to realize zero-memory-copy operations.

- Developers try to use Linux's APIs but found that performance was un-acceptable for unknown reasons. In the embedded community is widely believed that general proposes OSs Linux may be unsuitable for certain embedded applications since is bloated for desktop systems.

Another common believe in the embedded community is that any software design highly specialized will provide better performance than a implementing it over a generic solution. One example of this is multimedia software architectures on top of Linux, where the amount of data being processed requires to use data paths with zero memory copy operations across the whole software stack.

1.3. Problem statement

Many embedded developers tend to create software designs and architectures that aren't re-usable or maintainable for different reasons; one of them is the empiric premise that generic software architectures penalize performance. Furthermore there are little incentives from the product development cycle for embedded products to invest into optimizing the OSs. All these factors lead to creation of software platforms that are hard to reuse or leverage into future project versions or other projects, potentially increasing the costs of development.

This work focuses on the particular scenario of multimedia software systems for embedded systems and uses Linux on the ARM architecture.

1.4. Hypothesis

It it's possible to optimize general propose Operating Systems to provide good performance with negligible penalty compared to custom solutions where not Operating System is used for multimedia software stacks in embedded systems.

1.5. Previous work

1.5.1. Literature review

Classic works:

exokernel: [10]

SPIN: [2]

YIMA: [18]

It wasn't possible to find existing literature on the specific subject of this work, however we found related literature that may apply.

Liedtke [13] proved with his work that using the right algorithms and analysis it is possible to implement software stacks that manage hardware access with minimal performance penalties. He faced common belief that μ-kernels suffer inherit design performance penalties and demonstrated that rather than a design issue it was an implementation issue the root cause of the performance problems. Liedtke also showed that architecture-specific optimizations were required without loosing generality in the software architecture design.

If we characterize the main differences between Linux and RTOSs or OS-less solutions, we can identify previous work that addresses the performance and optimizations for these areas in embedded processors.

[15] Operating system support for multimedia systems

[17] Operating System Support for Multimedia: The Programming Model Matters

Memory Protection and Context Switching

[4]) implemented FASS

[8] instrumented performance penalties associated with context-switching in ARM processor architecture.

[3] Protected Hard Real Time

I/O Handling

[5]: Zero copy tcp in solaris

Netwokring: [12]

Buffers and algorithms: [11]

Efective IO for RTP: [14]

[12] A Memory Copy Reduction Scheme for Networked Multimedia Service in Linux Kernel

Process Scheduling

TODO

1.5.2. Software Stack

Performance of Gstreamer: [7] and obviously me: [9]

1.6. Long-Range Consequences

This work will contribute to the analysis of the factors involved on measuring and improving performance the Linux kernel for his use on embedded systems dealing with multimedia software stacks.

CHAPTER

TWO

OBJECTIVES AND CONTRIBUTIONS

Don't find fault, find a remedy.

Henry Ford

2.1. General Objective

- Measure and analyze the possible performance bottlenecks in the design of a general-propose Operating System (specifically in Linux) in an embedded system for handling of multimedia data streams to propose changes in the algorithms and data structures to improve the performance for such data scenarios.

2.2. Specific Objectives

- To instrument and measure performance of the Linux kernel in an specific set of embedded systems specialized for multimedia data processing, using an specific set of multimedia processing scenarios.
- To design and execute experiments to identify the sources of performance bottlenecks in the selected scenarios.

- Document and characterize the results from the experiments for performance measurement.
- Propose algorithms or changes required to improve performance.

2.3. Contributions to the subject

This work will provide documentation regarding the details of the specific tradeoffs for the implementation approaches of multimedia software on the target platforms, and could serve as basics for future work regarding improving performance (for example exploring improvements for power consumption reduction on the specific scenarios).

2.4. Scope and Limitations

- Work will be done specifically using the multimedia framework GStreamer on Linux embedded systems running on SoCs from Texas Instruments on two different architectures:
  - OMAP3x architecture: this SoC has a DSP co-processor to offload codec processing.
  - DM36x architecture: this SoC has non-programable hardware accelerators to offload codec processing.
- Implementations of the proposed algorithms and improvements will be made, but no guarantee that such changes will be merged back into mainstream open source projects is done. Still the work will be done interacting with the respective project maintainers with the idea of creating changes that can be merge on mainstream.

CHAPTER

THREE

METHODOLOGY AND SCHEDULE

> There is no greater harm than that of time wasted.
>
> Michelangelo

3.1. Methodology

Describe in technical language your research perspective and your past, present, or possible future points of view. List three research methodologies you could use, and describe why each might be appropriate and feasible. Select the most viable method.

3.2. Assumptions

Describe untested and un-testable positions, basic values, world views, or beliefs that are assumed in your study. Your examination should extend to your methodological assumptions, such as the attitude you have toward different analytic approaches and datat-gathering methods. Make the reader aware of your own biases.

3.3. Procedure

Describe in detail all the steps you will carry out to choose subjects, construct variables, develop hypotheses, gather and present data, such that another researcher could replicate your work. Remember the presentation of data never speaks for itself, it must be interpreted.

3.4. List of Deliveries

1. Patches to existing open source projects that implement the changes performed to the code against a specific version of the projects.

3.5. Schedule and Work Breakdown

Table 3.1: Work Breakdown

Task	Duration
• 1) Write Thesis Document Skeleton	1w
• **2) Review Literature and write down existing work**	**4w**
• 2.1) Review existing research and document the previous work	3w
• 2.2) Incorporate feedback from reviewers	1w
• **3) Collect initial performance measurements**	**8w**
• 3.1) Review existing tools for performance measurement	1w
• 3.2) Define testing methodology and tools	2w
• 3.3) Setup working environment and tools	2w
• 3.4) Collect measurement information about initial status of the platform for a set of...	2w
• 3.5) Incorporate feedback from reviewers	1w
• **4) Review performance optimizations at multimedia platform level**	**3w**
• 4.1) Review current performance at no-op case for test scenarios	1w
• 4.2) Review possible performance optimizations for no-op cases	2w
• **5) Review performance optimizations for I/O operations**	**15w**
• 5.1) Review performance for video interfaces	3w
• 5.2) Review performance for audio interfaces	2w
• 5.3) Review performance for co-processor interfaces	4w
• 5.4) Review performance for disk I/O	2w
• 5.5) Review performance for network I/O	4w
• **6) Finalize Thesis Document**	**4w**
• 6.1) Finalize document	2w
• 6.2) Incorporate feedback from reviewers	2w

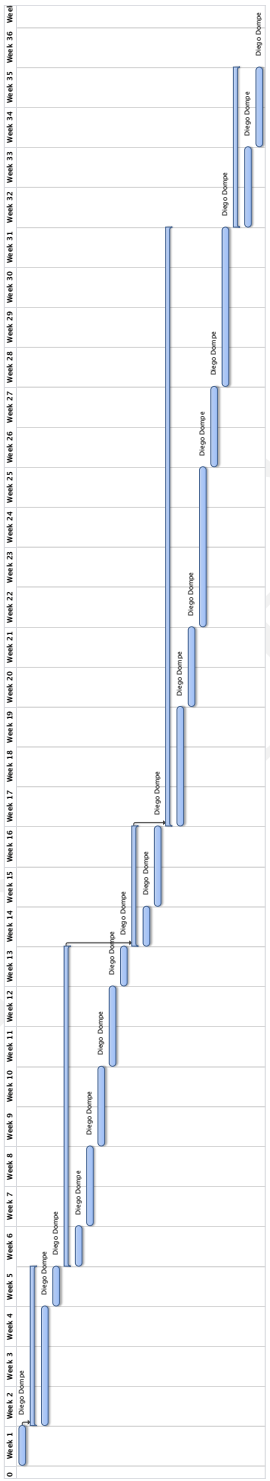

Figure 3.1: Gantt Chart

BIBLIOGRAPHY

[1] Difference Between ODM and OEM. URL http://www.differencebetween.net/miscellaneous/difference-between-odm-and-oem/.

[2] Brian N. Bershad, Stefan Savage, Przemys Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, and Susan Eggers. Extensibility, safety and performance in the SPIN operating system. pages 267–284, 1995.

[3] Bernard Blackham, Yao Shi, and Gernot Heiser. Protected Hard Real-time: The Next Frontier. In *Proceedings of the 2nd Asia-Pacific Workshop on Systems*, Shanghai, China, Jul 2011.

[4] Gilles Chanteperdrix and Richard Cochran. The ARM Fast Context Switch Extension. In *Proceedings from the Real Time Linux Workshop*, 2009.

[5] Jerry Chu and Sunsoft Inc. Zero-Copy TCP in Solaris. In *In Proceedings of the USENIX 1996 Annual Technical Conference*, pages 253–264, 1996.

[6] Peter Clarke. Embedded processor market to resist financial crisis, say VDC. *EE Times*, 2009. URL http://www.eetimes.com/electronics-news/4194731/Embedded-processor-market-to-resist-financial-crisis-say-VDC.

[7] Felipe Contreras. GStreamer, embedded, and low latency are a bad combination. URL http://felipec.wordpress.com/2010/10/07/gstreamer-embedded-and-low-latency-are-a-bad-combination/.

[8] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 workshop on Experimental computer science*, ExpCS '07, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-751-3. doi: http://doi.acm.org/10.1145/1281700.1281703. URL http://doi.acm.org/10.1145/1281700.1281703.

[9] Diego Dompe and Francisco J. Torres-Rojas. Tuning GStreamer on Embedded Systems for Soft Real-Time. In *Proceedings of the Embedded Technology Conference 2011*, volume 1, pages 44–50, 2011.

[10] Dawson R. Engler, M. Frans Kaashoek, and James O'toole. Exokernel: An operating system architecture for application-level resource management. pages 251–266, 1995.

[11] Xin Jane, Chen Milind, M. Buddhikot, Dakang Wu, and Guru M. Parulkar. Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS. In *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'98*, pages 326–337, 1998.

[12] JeongWon Kim, YoungUhg Lho, YoungJu Kim, KwangBaek Kim, and SeungWon Lee. A Memory Copy Reduction Scheme for Networked Multimedia Service in Linux Kernel. In *Proceedings of the First EurAsian Conference on Information and Communication Technology*, EurAsia-ICT '02, pages 188–195, London, UK, 2002. Springer-Verlag. ISBN 3-540-00028-3. URL http://portal.acm.org/citation.cfm?id=646603.697068.

[13] Jochen Liedtke. On μ-kernel construction. In *15th ACM symposium on Operating Systems Principles*, 1995.

[14] Nam-Sup Park and Chong-Sun Hwang. Effective I/O Scheme Based on RTP for Multimedia Communication Systems. *Journal of Computer Science and Technology*, 21:989–996, 2006. ISSN 1000-9000. URL http://dx.doi.org/10.1007/s11390-006-0989-5. 10.1007/s11390-006-0989-5.

[15] T. Plagemann, V. Goebel, P. Halvorsen, and O. Anshus. Operating system support for multimedia systems. *Computer Communications*, 23(3):267 – 289, 2000. ISSN 0140-3664. doi: DOI:10.1016/S0140-3664(99)00180-2. URL http://www.sciencedirect.com/science/article/pii/S0140366499001802.

[16] Brian Proffitt. Linux ARM support: A hot mess, an ugly clean-up. *ITworld*, 2011. URL http://www.itworld.com/mobile-wireless/175829/arm-and-linux-major-construction-ahead.

[17] John Regehr, Michael B. Jones, and John A. Stankovic. Operating System Support for Multimedia: The Programming Model Matters. Technical report, 2000.

[18] Cyrus Shahabi, Roger Zimmermann, Kun Fu, and Shu-Yuen Didi Yao. Yima: A Second- Generation Continuous Media Server. *IEEE Computer*, pages 56–64, 2002.

[19] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts Essentials*. Wiley Publishing, 2010. ISBN 04701287209.

[20] Deepak Somaya and Greg Linden. System-on-a-Chip Integration in the Semiconductor Industry: Industry Structure and Firm Strategies. *SSRN eLibrary*, 2000. doi: 10.2139/ssrn.259878.

[21] Yang Song and Peter Chubb. Interrupts Considered Harmful. In *11th Linux.conf.au*, Wellington, New Zealand, Jan 2010. annotated slides from presentation.

ACRONYMS

DSP Digital Signal Processor is a special type of micro processor.

EE Electronic Engineer

GPP General Purpose Processor is a processor suited for standard computational operations, in contrast of a DSP for example.

OEM Original Equipment Manufacturing refers to a company or a firm that is responsible for designing and building a product according to its own specifications, and then selling the product to another company or firm, which is responsible for its distribution. The one company produces products on behalf of another company, after which the purchasing company markets the product under its own brand name[1].

ODM Original Design Manufacturing is a company or firm that is responsible for designing and building a product as per another company's specifications[1].

OS Operating System

PC Personal Computer

RTOS Real Time Operating System is an operating system that is designed to meet with Real Time deadlines.

SoC System on a Chip is a technology that consist on integrating several functional blocks of re-usable electronics logic (even from different vendors) into single die.