A Lightweight Linux Architecture for Resource-Limited Media Systems

Holger Patecki, Peter Altenbernd, Michael Ditze, Reinhard Bernhardi-Grisson
{holger.patecki, peter.altenbernd, michael.ditze, reinhard.bernhardi}@c-lab.de

Abstract

Integrating multimedia processing into resource-limited computer systems (either stand-alone or part of a network environment) is more challenging than integrating it into traditional systems. With regards to multimedia applications, conventional systems have essentially limitless memory, non-volatile storage and computing power. Embedded devices are usually limited by at least one of those resources, if not all. This paper describes how we approached multimedia processing on limited platforms by introducing a Linux-based approach that focuses specially on the interaction of an enhanced kernel scheduling behavior and a down-scaled operating system. The experimental results clearly show the benefit of our prototype for MPEG processing, as compared to a full-version Linux system. Thus our approach can be utilized for media streaming in real-time networks.

Keywords: Linux, Multimedia, Scheduling, Quality of Service, Real-time Systems

1 Introduction

Multimedia applications such as distributed Video-on-Demand and video conferencing, represent added-value services that increasingly cross converging network borders by exploiting high-speed access and core networks. They strive for deployment in heterogeneous application domains that feature all kinds of end-devices, from traditional desktop stations to fast-developing embedded devices like cellular phones, set-top boxes or PDAs. As these exhibit restricted resources in terms of processing power, memory and graphical capabilities, embedded real-time operating systems (RTOS) are required which support multimedia processing and simultaneously claim low system resources. Quite a few such RTOS hit the market, either commercially available like Windows XP embedded or Windows CE, or open-source labeled like Linux. What they have in common is their lack of multimedia support in resource-restricted environments.

This paper will introduce an affordable open-source software architecture based on the Linux OS. Unlike other approaches, it is scaled down to suit embedded device requirements and yet supports multimedia processing like video encoding, decoding and transcoding.

As shown in previous approaches [2], multimedia processing (especially MPEG decoding and transcoding) is significantly improved in the presence of dedicated priority-based scheduling algorithms. Similarly, MPEG processing can be interfered by at least two major causes: kernel latency times[1] and a couple of services provided by the operating system. However, some of these services are redundant on the dedicated system, merely consuming CPU time. Additionally, by limiting data memory one is forced to consider a small-scale user interface for reserving memory resources for the main objective of the system: media processing.

Thus we follow an approach that is capable of composing any media system based on a tool kit. With the help of this tool kit it is possible to include only the components that are essential in the context of a down-scaled Linux OS, enhanced by kernel patches, that are applied in order to reduce the latency times while accessing the local hard disk buffer. The experimental evaluation (see Section 4) shows that our resulting prototype clearly outperforms both a standard full-scale Linux system as well as an optimized and down-scaled Linux system. Our criteria for comparison were the transcoding and encoding performance of a system.

Our prototype represents one example of a general approach to creating media clients, for both stand-alone and network environments. The domains covered can be found at home, in TV sets, in video-on-demand systems [4], in VCR's, in cars (Infotainment) and in industrial automation (maintenance, surveillance). The resulting appliances will be affordable both in terms of development costs and hardware components. This paper mainly contributes to real-time networking by improving the process of handling end system encoding and transcoding.

The rest of the paper is organized as follows. In

[1]Latency time is the interval between a wakeup signaling, that an event has occurred and the Linux kernel scheduler gets the opportunity to schedule the thread, that is waiting to be processed.

the next section, we will show related work, including a brief introduction to modifications of the Linux scheduling algorithms. Section 3 introduces the architecture of our system; followed by Section 4, which presents a first use case by evaluating the encoding ability of our system compared to a standard full-scale Linux OS. We will show that our prototype clearly outperforms standard Linux architecture in terms of media processing. In Section 5, we give our conclusions and prospects to future work.

2 Related Work

2.1 The Linux Kernel Latency

The main obstacles for deploying resource-limited computer systems for multimedia processing are latency times of the Linux kernel and the absence of a priority-based scheduling algorithm. This section briefly introduces Linux kernel modifications that aim to improve the interrupt and scheduling behavior. Two kernel patches are presented. The integration of both of them results in reduced kernel latency times as shown in our use case below.

There are six sources of latency in the Linux kernel [8]: Calls to the disk buffer cache, memory page management, calls to the /proc file system, VGA and console management, large processes and the keyboard driver. Two former approaches deal with the kernel latency problem: the *Preemption Patches* and the *Low Latency Patches*.

The *Preemption Patches* [5] allow a lower priority process to be preempted, even in kernel space, resulting in improved system response. This is done by modifying the spinlock[2] macros and the interrupt code.

The basic idea of the *Preemption Patches* is to check opportunities to preempt and reschedule when releasing a spinlock or returning from an interrupt routine. The *Preemption Patches* prevent preemption while holding a spinlock.

Different from this brute-force strategy, the *Low Latency Patches* [8] place conditional scheduling calls in strategic locations throughout the Linux kernel. These calls are able to interrupt long-term kernel threads. The idea is to detect regions that iterate over large data structures and find out how to introduce scheduler calls to re-schedule the system. By breaking spinlocks of long-term operations and re-acquiring them, kernel latency can be significantly reduced. As easy the *Low Latency Patches* concept is, as complicated is the implementation. One has to find out high-latency blocks of code and insert preemption points or modify the circumstances that lead to the latency problem, respectively.

[2]Spinlocks work through a shared variable. If a process needs the lock, it has to query this variable and if it is not available, the process "spins" until the lock is relinquished.

2.2 Graphical User Interface

This section reviews the effort that has been made to develop a graphics engine that is suitable for a resource-limited computer system. As mentioned above the amount of the standard Linux window system XFree86 is too large to be used in the target system.

MiniGUI [6] is a software project, that aims to provide a fast, stable, and lightweight Graphics User Interface. The design makes *MiniGUI* especially suitable for real-time embedded systems based on Linux. *MiniGUI* defines a set of light windowing APIs for applications. By using them, an application can create multiple windows and controls, and draw in these windows and controls without interfering the others. *MiniGUI* runs on top of the frame buffer device which is provided by the Linux operating system. The frame buffer device has a good performance at low resolutions but gets slower with increasing resolutions. This effect can be caused by high kernel latency times as described below.

QT/Embedded [7] is a toolkit for GUI and application development for embedded devices. It runs usually with Embedded Linux. *QT/Embedded* applications write directly to the frame buffer device, eliminating the need for an X Window System. *QT/Embedded* lacks a wide variety of multimedia applications. However it has good programming guidelines and is the first choice for developing applications for a resource-limited computer system.

2.3 Media Processing

The need for an appropriate MPEG encoder and transcoder arises by exploiting our platform for media processing. The most capable player in the Linux environment is probably MPlayer [3], supporting a wide range of video codecs. MPlayer's including encoder called MEncoder is an advantage over competing Open Source projects (e.g. Xine) targeting the same multimedia market. MEncoder converts uncompressed or coded video into a wide variety of video formats (including MPEG1, MPEG4, etc.), which may be necessary for archiving movie clips within a distributed computer network.

3 Our Approach

While the Linux OS is not necessarily suitable for a resource-limited computer system, its modularity and its adaptability makes it possible to scale down its complexity in order to achieve an appropriate architecture for an embedded system.

The target platform is limited by at least non-volatile and program memory. As a consequence there is the need to scale the operating system. Section 3.1 will show how we scaled Linux to be suitable for our end system.

3.1 Scaling Linux

Traditional desktop oriented Linux distributions run into hundreds of megabytes. To scale the Linux system we need for multimedia processing, we choose only mandatory components that make sense for our applications. We abandon services that are not needed by our specialized architecture yielding a smaller number of threads permanently running on our Linux box. Off-the-shelf Linux distributions brimming with security in a lot of applications, first of all the shells. We do not need the security functionality, especially PAM (Pluggable Authentication Module) within a system that focuses on performance and memory-saving aspects rather than bringing out an all-in-one device that is suitable for any kind of application. The same logic applies to networking, shells, servers and countless other applications, services and libraries. Developments formerly found in the context of embedded systems were utilized for a synthesis of our prototype. We employed *Busybox* [1], which is also known as the "Swiss Army Knife of Embedded Linux"; *Busybox* contains a lot of shell tools, that can be accessed via options in only a single application. So lots of programs normally running on a Linux box can be abandoned, reducing the footprint of our platform.

3.2 The Kernel Scheduler

As described in Section 2 there are two former approaches dealing with kernel latency and therfore with an enhanced scheduling behavior of the Linux kernel. Abandoning the requirement to solve the latency problem of the Linux kernel by switching to a real-time kernel, which would make it at least necessary to modify and re-compile the user-space applications; we decided to make use of the existing kernel enhancements. By evaluating measurements of the resulting kernel latency by ourselves and by utilizing the work done in [9] we applied both patches to the Linux kernel in order to achieve the lowest scheduling latency. The latency improves from peaks of 13ms while writing to the hard disk on a full-scale Linux OS to peaks of 4ms on our prototype.

3.3 The Graphical Alternative

Dealing with the limits led us to consider the substitution of the window system with an appropriate engine with smaller system requirements.

A graphical system like the matured traditional Linux graphics engine X11 is inadequate to run a limited computer platform. Without launching any application the X Server grabs 2.6 MBytes of program memory, while systems that address the Linux frame buffer device like *QT/Embedded* and *MiniGUI* primarily need the memory for their applications. The amount of additional memory for the window system can almost be neglected. By substituting the graphical interface, non-volatile and data or program memory can be reduced significantly.

The graphical systems tested are qualified for the substitution of the standard XFree86 server. But there is a lot of work to be done when using an alternative system mentioned above. Applications designed for X11 do not automatically run on one of the lightweight window system. They have to be adjusted for the API's that are different from the X11 API. The ambitions of the developers of *MiniGUI* to integrate the multimedia context into a small-scale user interface, e.g. the integration of a video player, influenced us to choose it for our prototype.

3.4 Computational Model

In this paper we consider only uniprocessor systems. The results we measured are transferable to distributed systems within real-time networks as well. Unlike other approaches we abandoned the possibility to alter the priority of processes that are important for our end system. The goal was to achieve a better performance of media processing on the standard priority level. Thus a much easier computational model has been reached.

The overall time for the concurrent encoding of movie clips (see Section 4) can be described with the equation:

$$T = \sum_{i=0}^{n} (D_i + S_i)$$

where T is the overall time for encoding, D_i the time to encode the respective video clip at 100% CPU time, S_i is the time for the scheduler to switch from one task to another (e.g. switching from encoding one video clip to another) and n is the number of video clips to be encoded concurrently. Additional CPU time for system services can be neglected without loss of generality. This is due to the reduction of services on our specialized system described in Section 3.1.

4 Evaluation

In an preliminary experiment we discovered the encoding ability of our platform. We encoded raw movie clips in MPEG4 (DivX5) format. Three files are encoded concurrently; the encoding time is compared on different platforms: the original (off-the-shelf) Linux system, a down-scaled version and our optimized and down-scaled prototype. Starting by encoding an uncompressed video clip into MPEG4, 30 seconds later our test continues with encoding of a second video clip (which for sake of comparison is the same) and yet another 30 seconds later with a third video clip, which is also the same as the first and second. The time needed to encode the respective video clips are measured on each system. The results are shown in Figure 1, the times for converting a single video clip is calculated as it were encoded at 100% CPU time. By comparing the resulting encoding times we can clearly show that our system significantly outperforms both other platforms.

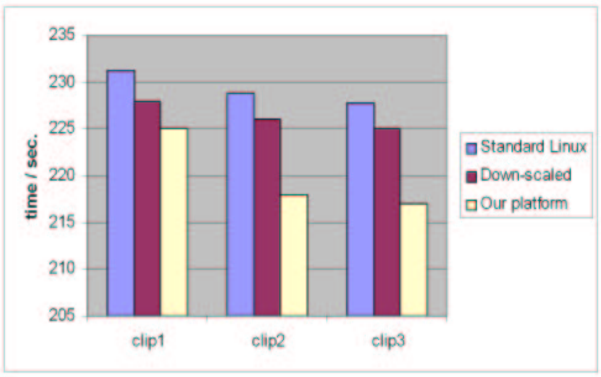

Figure 1. MPEG4 Encoding times

The measurement results show the improvements on MPEG4 encoding of our prototype. Measured enhancements currently are at 5% performance boost on the end system compared to a full-version Linux. We expect a much better result with am improved scheduler that will be dedicated to media processing.

5 Conclusions and Future Work

The work outlined in this paper describes our effort to enhance multimedia processing on resource-limited Linux platforms.

Utilizing our prototype significantly reduces the MPEG encoding times as shown in Section 4, while the lightweight user interfaces introduced save data memory and ensure a faster communication between the Linux kernel and the application running on top of it. This paper presents only a first glimpse in our view of a general platform to be provided for media processing on resource-limited platforms and is therefore still in progress. Future developments contain an optimized scheduling behavior for media encoding, especially with an adjusted quantum[3] (on condition dynamic) of the Linux scheduler. Our future efforts will also increasingly focus on streaming concepts concerning resource-limited platforms within real-time networks.

[3]A quantum or time slice is the period of time the scheduler assigns to a process, before the process is shuffled back to the end of the queue.

References

[1] BusyBox. *http://www.busybox.lineo.com.*

[2] M. Ditze and P. Altenbernd. *A Method for Real-Time Scheduling and Admission Control of MPEG-2 Streams.* The Seventh Australasian Conference on Parallel and Real-Time Systems (PART2000), 2000.

[3] K. Hindistan. *Video Playback and Encoding with MPlayer and MEncode.* O'Reilly Network, 2003.

[4] C. Loeser and P. Altenbernd. *Architecture of an intelligent Qualtiy-of-Service aware Peer-to-Peer Multimedia Network.* 7th World Multiconference on Systemics, Cybernetics and Informatics, 2003.

[5] K. Morgan. *Preemptible Linux: A Reality Check.* MontaVista Software, 2001.

[6] N.N. *MiniGUI Technology White Paper.* Feynmann Software Co., Ltd., 2002.

[7] N.N. *QT/Embedded White Paper.* Trolltech AS, 2002.

[8] D. Philips. *Low Latency in the Linux Kernel.* O'Reilly Network, 2000.

[9] C. Williams. *Linux Scheduler Latency.* Red Hat Inc., 2002.