Compensating for Interrupt Process Times in Real-Time Multimedia Systems

Luca Abeni and Giuseppe Lipari
ReTiS Lab, Scuola Superiore S. Anna,
Via dei Martiri 11, Pisa, Italy
{luca, lipari}@sssup.it

Abstract

This paper describes an on-going work focused on using Adaptive Reservations to cope with unpredictabilities due to the interrupt processing times in real-time kernels. Using this approach, the execution time stolen by interrupt processing is modeled as a variation in the application execution times, and an adaptive scheduling mechanism automatically compensates its effects.

1 Introduction

In the last years, there has been a growth of interest in running applications with real-time requirements (such as streaming applications, software mixers, and so on) on top of desktop Operating Systems (OSs). For this reason, a lot of effort has been spent in providing open source OSs such as Linux with real-time capabilities.

One of the biggest problems that have to be addressed is that interrupt processing time can dramatically influence the performance of a real-time version of Linux. In particular, interrupts are mainly managed in bottom halves that can steal execution time from the currently executing process. This is a source of unpredictabilities that can completely jeopardize a real-time guarantee enforced by the CPU scheduler.

Although this problem can be addressed by modifying the kernel, in this paper we propose a less intrusive solution, based on *adaptive reservations* [1]. The core abstractions upon which our solution is built are resource reservations, that are widely used in RT/multimedia systems, and have been implemented in a number of RT versions of Linux, such as, for example, Linux/RK [2]. Using the adaptive reservations approach, *the execution time stolen by interrupt processing is modeled as a variation in the application execution times*, and an adaptive scheduling mechanism can try to compensate the effects of this stochastic variation. The proposed approach has been implemented in Linux/RK (without requiring any additional modification to the original RK), and some preliminary experimental results are presented.

2 Interrupt Handling

A modern PC is connected to a lot of peripherals, which can be considered as hardware resources that the OS kernel has to manage. In most cases, those resources can produce hardware interrupts, and the kernel has to properly manage them.

In the Linux kernel, a hardware interrupt is served in two phases: first of all, a short **Interrupt Service Routine (ISR)** is responsible for acknowledging the hardware interrupt and for activating a proper handling mechanism (the bottom half). After that a *bottom half*, activated by the ISR, is responsible for correctly handling the data to be received or sent through the device. Note that the bottom half code requires some CPU time to execute, and this time is generally accounted to the next scheduled task, creating some random variations in the accounted execution times.

In order to better understand the problems deriving from this incorrect accounting, let's remember that bottom halves are executed by the dispatcher immediately before switching to user mode. As a side effect, three sources of unpredictability are introduced: **(1)** the handler code is executed at apparently random times and *is not scheduled*, introducing anomalies in the CPU scheduling; **(2)** as a result, the handler execution time is accounted to the interrupted process, and is seen by user application as *stolen time*; **(3)** bottom halves are not preemptable, violating one of the assumptions of real-time scheduling theory.

As a result of these scheduling anomalies, the real-time guarantees provided by the system may be broken.

3 Adaptive CPU Reservations

A possible solution to the problem highlighted in Section 2 is to reserve a specified amount of the CPU time to each task, and to adapt the amount of these reservations in order to compensate the effects of the bottom halves execution. The basic resource reservation abstraction needed to implement this strategy can be provided by Resource Kernels (RK), in which a process can be guaranteed to receive the resource for a time Q each period T. Note that the RK concept is quite general, and has been ported to Linux.
This idea is used by *Adaptive Reservations*, presented in [1]: the adaptive reservation abstraction was originally developed in order to cope with tasks characterized by unknown or highly variable execution times, but it can be successfully used to mitigate the effects of stolen time. Adaptive reservations are based on the real-time task model, in which each task is divided in jobs, and each job is characterized by an absolute deadline that can be used to monitor the application performance. In this way, an implicit monitoring of the effects of the bottom halves execution can performed by simply measuring the number of missed deadlines.
In order to understand the use of adaptive reservations in this work, only few basic concepts are recalled: **(1)** when a process is created, it is reserved Q time units each period T; **(2)** if the process misses some deadline, the reserved time Q is increased; **(3)** if the process does not miss deadlines for a sufficiently long time, its reserved time Q is decreased.
When the network load increases, the bottom halves begin to consume a significant amount of CPU time, stealing it to reserved processes. Hence, a reserved process will miss some deadline, and if the process is attached to an adaptive reservation its reserved time will be increased. In this way, the amount of time reserved to a process increases when the network traffic increases, compensating the effects of the bottom halves execution.

4 Experimental Results

In order to show the impact of the execution time stolen by bottom halves execution, some experiments have been performed using Linux/RK, a Resource Kernel based on Linux.
The influence of the bottom halves on the CPU scheduling can be easily seen by causing a lot of bottom halves executions and measuring the impact on a reserved process, as shown by the following experiments. First of all, a periodic process composed by jobs having execution times $4ms$ has been run with period $T = 20ms$, attached to a $(4ms, 20ms)$ CPU reservation. As expected, if bottom halves do not execute for too much time, the process does not miss any deadline. In fact, the difference between the termination of two consecutive jobs (referred as *job inter finishing time* in this paper) is about constant, and equal to $20ms$ (the process & reservation period), as shown by the first plot in Figure 1.

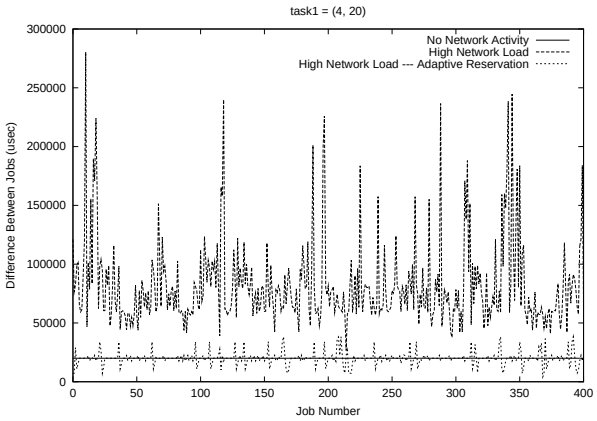

FIGURE 1: *Effects of bottom halves execution on job inter finishing times*

After that, a heavy network traffic has been sent to the test machine, in order to increase the CPU time consumed by bottom halves. As a result, the job inter finishing times become unpredictable, and the process starts to miss deadlines, as shown by the second plot in Figure 1.
In order to test how adaptive reservations can solve this problem, the experiment has been repeated attaching an adaptive reservation with period $20ms$ to the user process. Note that adaptive reservations can be implemented in user space, without requiring modifications to the kernel (the only requirement is that the kernel provides temporal protection in the CPU scheduler). As shown by the third plot in Figure 1, the adaptive reservation was able to control the job inter finishing times below $40ms$ in a short time, compensating the effects of bottom halves execution, and avoiding deadline misses.

References

[1] L. Abeni and G. Buttazzo, *Adaptive Bandwidth Reservation for Multimedia Computing*, Proceedings of the IEEE Real Time Computing Systems and Applications, Hong Kong, December 1999.

[2] S. Oikawa and R. Rajkumar. *Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior*, Proceedings of the IEEE Real-Time Technology and Applications Symposium, Vancouver, June 1999.