Mapping of the synchronization mechanisms of the Linux kernel to the response-time analysis model

Daniel Bristot de Oliveira
Federal University of Santa Catarina
Florianopolis, Brazil
daniel@bristot.eti.br

Romulo Silva de Oliveira
Federal University of Santa Catarina
Florianopolis, Brazil
romulo.deoliveira@ufsc.br

ABSTRACT

Response-time analysis is a method for determining the schedulability of real-time systems based on fixed priority. Although the Linux kernel with the PREEMPT_RT patch is a real-time system based on fixed priority, the tools are used only to measure its latency. The kernel is generally considered as a black box. This paper associates the variables used in the response-time analysis with the kernel functions that cause blocking and interference. The objective of the present study is to identify the points of interest within the Linux kernel for the response-time analysis.

Categories and Subject Descriptors

D.4.8 [**OPERATING SYSTEMS**]: Performance—*Measurements*; D.4.1 [**OPERATING SYSTEMS**]: Process Management—*Mutual exclusion*; D.4.7 [**OPERATING SYSTEMS**]: Organization and Design—*Real-time systems and embedded systems*

Keywords

Linux, Real-Time Systems, Response-Time Analysis, Mutual Exclusion, Measurements.

1. INTRODUCTION

The patch PREEMPT_RT is the de facto standard for real-time Linux [4][9]. Nevertheless, there are frequent discussions about the differences between the real-time Linux and the theory of real-time systems [2][1]. Among the points of contention there are the method of analysis and the metrics used to evaluate the system.

Considering the Linux kernel, the primary metric for real-time analysis is latency, which is the main metric for PREEMPT_RT. To get the value of the primary metric, the developers of the Linux kernel use the tool *Cyclictest* [8]. *Cyclictest* is a benchmark which handles the Linux kernel as a black box.

SAC'14 March 24-28, 2014, Gyeongju, Korea.

http://dx.doi.org/10.1145/2554850.2555113

The strategy used in the Linux kernel is effective for many applications. However, both the metric and the classic analysis of the system are simplistic when compared with the theory of real-time systems.

The purpose of this article is to map the functions of the Linux kernel, which can block or delay user tasks, to the variables used in the response-time analysis. This mapping is necessary as the basis for a future improvement of the real-time analysis of the Linux kernel.

2. RESPONSE-TIME ANALYSIS

For system based on fixed priority, it is possible to verify the schedulability using the method of response-time analysis [6]. The maximum response time is the time a task takes to complete its execution while considering the worst-case blocking and interference this task may suffer. The maximum response time of a task τ_i is represented by variable R_i, which is obtained with Equation 1.

$$\begin{aligned} R_i &= W_i + J_i \\ W_i &= C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil . C_j \end{aligned} \quad (1)$$

Using this equation, it is possible to determine the worst-case response time taking into account the following variables: C as the worst-case execution time of the task; P as the period of the task; B as the worst-case blocking time of the task; J as the maximum release jitter of the task; and $W_{hp(i)}$ as the interference from higher priority tasks.

The goal of this paper is to identify how the operating system affects the release jitter J and blocking time B.

3. RELEASE JITTER

The maximum release jitter, represented by variable J, occurs when a higher priority task is delayed at the beginning of its execution, due to a lower priority task. In the Linux kernel with PREEMPT_RT, there are two types of tasks: interrupt handlers and threads.

Interrupt Handlers: Its activation happens with the occurrence of an interrupt. There is only one way a hardware interrupt is signaled without immediately stopping the execution of a task, is when the system has already disabled that interrupt. The Linux kernel includes functions to disable all maskable interrupts of a processor or disable an interrupt on all processors.

Preemption: The threads are activated by events that change their state in the scheduler, from sleeping to ready

to execute. When a higher priority thread is awakened by a lower priority thread, the scheduler is called and starts execution of the thread of higher priority, unless preemption is disabled.

4. BLOCKING

Blocking happens when a lower priority task retains a lock requested by a higher priority task. The Linux kernel has several mechanisms for mutual exclusion. There are two reasons for these several different mechanisms: The execution contexts (interruptions and threads) which have different constrains, and optimization cases (performance or determinism). The methods of generating mutual exclusion that cause blocking in the Linux kernel are described below.

Spinlock: Only one task is allowed to access a section protected by a spinlock. A task blocked on a spinlock is held at busy waiting. Despite the fact that busy waiting for the lock consumes CPU time in vain, this avoids a more complex control, which is beneficial when you have small critical sections [7]. In the kernel with PREEMPT_RT, spinlocks are converted to RT Mutexes.

Read-write spinlocks: In some cases, the critical sections are accessed multiple times for data reads, and sometimes for the update. To improve the throughput of the system, exclusive access to these data is needed only when writing the data. There may be concurrent accesses to read the data and an exclusive access while updating the data [7]. The kernel vanilla uses spinlocks to protect the write access.

Semaphores: Unlike spinlocks, semaphores do not use busy waiting. When a task tries to acquire a semaphore and this is unavailable, the semaphore puts the task to sleep [5]. Semaphores accept various tasks in its critical section. This is controlled by a counter. To implement mutual exclusion using a semaphore, the counter must be one. One side effect is that by making the task to sleep, it is possible for a high-priority task to suffer unlimited priority inversion. Semaphores also have a version for read-write. Read-write semaphores do not have counters, the rule is the same as read-write spinlocks: a writer requires mutual exclusion but several concurrent readers are possible.

Mutex: The mutex option was implemented as a simple mutual exclusion to put tasks on contention to sleep, mainly to replace semaphores initialized with a count of one.

RT Mutex: The RT mutexes extend the semantics of mutexes with the priority inheritance protocol.

RT Mutex and PREEMPT_RT: In the Linux kernel with the patch PREEMPT_RT, spinlocks and mutexes are converted to RT mutexes. Spinlocks are converted to rt_spinlocks, using the RT Mutex to implement mutual exclusion. This is possible because in the PREEMPT_RT many sections of the kernel which were originally in interrupt context, were converted to threads running in the kernel.

RCU: The RCU (read-copy-update) is a synchronization mechanism. However, as it uses some features of the architectures of current processors such as the atomicity of operations with pointers aligned in memory, the RCU allows a writer and multiple readers in a critical section, concurrently. Thus it achieves a better performance when compared with the read-write spinlocks [3].

5. KERNEL MECHANISMS AND THE RESPONSE-TIME ANALYSIS

Mechanism	Abstraction
IRQ Control	Release Jitter
Preemption Control	Release Jitter
Spinlock	Blocking
RW Spinlocks	Blocking
Semaphores	Blocking
RW Semaphores	Blocking
Mutex	Blocking
RT Mutex	Blocking
RCU	Blocking

Table 1: Mapping between mechanisms of the Linux kernel and abstractions of the response time analysis

This Section presents the mapping of the Linux kernel mechanisms, listed in Sections 3 and 4, to the abstractions used in the response-time analysis, described in Section 2. This mapping is described in Table 1.

It can be seen that several synchronization mechanisms are used within the kernel, each of them may generate blocking. The release jitter happens due to the control mechanisms of interruption and preemption. The next step towards an analysis of response time that includes the Linux kernel would require the determination of the maximum time delay resulting from the code sections where such mechanisms are used.

6. CONCLUSIONS

This paper mapped the abstractions of blocking and release jitter from the theory of response time analysis to the timing of the kernel functions.

We believe it is necessary to develop another form of analysis for demonstrating the performance of real-time tasks, taking into account the variables used in the response time analysis.

7. REFERENCES

[1] B. B. Brandenbug and J. H. Anderson. Joint opportinities for real-time linux and real-time system research. *Real Time Linux Workshops*, 2009.

[2] T. Gleixner. Realtime linux: academia v. reality. *http://lwn.net/Articles/471973/*, July 2010.

[3] D. Guniguntala, P. E. McKenney, J. Triplett, and J. Walpole. The read-copy-update mechanism for supporting real-time applications on shared-memory multiprocessor systems with linux. *IBM Systems Journal*, 2008.

[4] R. H. Inc. Red hat enterprise mrg. *http://www.redhat.com/products/mrg/*, September 2013.

[5] G. K.-H. Jonathan Corbet, Alessandro Rubini. *Linux Device Driver*. O'Reilly Media, 3 edition, 2005.

[6] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[7] R. Love. *Linux kernel development*. Addison-Wesley, 3 edition, 2010.

[8] T. G. Paul McKenney. Cyclictest. *https://rt.wiki.kernel.org/index.php/Cyclictest*, July 2013.

[9] PREEMPT_RT. Real-time linux wiki. *https://rt.wiki.kernel.org/*, July 2013.