Analysis and Research on Improving Real-time Performance of Linux Kernel

BI Chun-yue
School of Electronics and Computer/ Zhejiang Wanli University/Ningbo, China

ABSTRACT

With the widespread application of Linux operating system in embedded system, it is an important problem to effectively improve the real-time performance of Linux. In this paper, we analyze the real-time problems of Linux kernel. To solve these problems, we put forward a series of methods to improve the real-time performance of Linux, such as dual-kernel methods, interrupt mechanism, clock granularity and real-time scheduling algorithms. After testing, the real-time performance of Linux has been greatly improved, which can satisfy the requirement of real-time embedded system.

Keywords: Real-time Performance, Linux Kernel, Dual kernel, Clock granularity, Interrupt Mechanism, Scheduling algorithm.

I. INTRODUCTION

With the rapid development of information technology, real-time embedded systems account for more and more proportion in daily life, a good many real-time system products can be seen everywhere around us. So real-time operating system—RTOS—has become the focus of research, compared with common OS, RTOS has a good characteristic, i.e. real-time performance. Real-time operating system (RTOSs) provide basic supports for scheduling, resource management, synchronization, communication, precise timing, and I/O. RTOSs have evolved from single-use specialized system to a wide variety of more general-purpose operating system such as real-time variants of Linux. We have also seen an evolution from RTOSs which are completely predicable and support safety-critical applications to those which support soft real-time applications. Such support includes the concept of quality of service for open real-time systems, often applied to multimedia application as well as large, complex distributed real-time systems. At present, the RTOS market includes many popular Oss such as VxWorks, QNX, LynxOS, Nucleus, uC/OS-II and so on.

In recent years, Linux operating system is widely applied in the embedded field such as industrial automation, consumer electronics, communications, intelligent appliances, and so on. For its advantages of open source, supports a variety of hardware platforms, a rich library functions, modular design, Linux has attracted strongly the majority of developers to study. At present, more and more products need to have the real-time performance, but Linux isn’t a RTOS, it is a time-sharing operating system itself. Relative to the real-time systems, Linux has several disadvantages such as non-preemptive kernel, rough time granularity, frequent disabled interrupt, priority inversion, passive scheduling, virtual memory, and so on. So many institutes begin to research on Linux kernel and improve its real-time performance, which has been an important research subject. Many researchers have developed new ideas that enhance traditional operating systems to be more efficient and predicable.

II. REAL-TIME PROBLEMS OF LINUX

A. Characteristics of Real-time Embedded System

Real-time performance is an important characteristic of RTOS, which not only depends on the logic correctness, but also depends on outputting result in time. According to different demands of real-time, RTOS can be divided into two types, hard real-time system and soft real-time system. Embedded real-time system has common characteristics and several key technologies as follows [1].

- multi-tasking functions and task switch time
B. Real-time Problems of Linux

Embedded Linux operating system, mostly obtained from the standard Linux, and has many advantages. But the general Linux is not a real-time operating system, lack of real-time performance in the application is a serious obstacle. Real-time constraints of Linux operating system have the following main points ([1], [2], [3]).

1) Non-preemption of Kernel

Preemption can be described the following process, when a priority appears higher than the current running process in the system, the system immediately interrupts the running process and switches to the high priority process immediately. Due to the kernel of Linux is non-preemptive, so the real-time performance is very poor. All these need to be improved to meet its real-time performance.

2) Process Synchronization and Mutual Exclusion

Linux takes semaphores method to ensure that the process synchronization and mutual exclusion. Due to the frequent locking and unlocking operation of the signal will affect the overall performance of the system, so Linux adopts strategy of rough clock granularity. Signal locking and unlocking operation will result in a longer interval, so systems are often unable to meet requires of many hard real-time applications.

3) Priority Inversion

Linux OS does not provide priority inheritance strategies, so the priority inversion phenomenon will make real-time process with high-priority be blocked by a low-priority process, and result in the uncertainty of execution time.

4) Interrupt Processing

In the course of system call, Linux often turns off interrupt for a long time to protect critical resources. This will increase the interrupt latency time, block high-priority interrupt will be processed immediately, result in real-time tasks can't be promptly scheduled for execution.

5) Clock Granularity is Rough

Clock management is the pulse of the operating system, time accuracy of Linux task scheduling can not satisfy some real-time applications which need rigorous time precision. In general circumstances, execution and suspension of tasks will be aroused by the clock directly or indirectly, and it is also an important basis for process scheduling. The clock frequency of Linux is 100Hz, the clock period is 10ms and the clock precision is too rough, all these can't meet the real-time requirements. In addition, soft-timer of Linux is completed by clock timer, when timers are relatively more, they will give rise to the conflict among the clock timers.

6) Virtual Memory

Management of virtual memory becomes a bottleneck of real-time application. There are many real-time tasks should be loaded in memory, it will take more time to go out and in memory if virtual memory technology is used. Obviously, it can't fit in with the requirement of real-time applications. According to the problems, we can modify Linux kernel to improve system's real-time performance and make it be applied to real-time environments.

III. MEASURES TO IMPROVE THE REAL-TIME PERFORMANCE OF LINUX KERNEL

There exist defects of Linux in real-time aspect, how to enhance the real-time performance of Linux to meet the needs of real-time systems is the focus of research recently. At present, many methods are adopted to implement
real-time performance of Linux. Generally there are two ways to improve the real-time performance of Linux: dual kernel method and direct kernel modification ([4], [5]). In this paper, the real-time performance of Linux is improved through several aspects such as dual-kernel method, interrupt mechanism, clock granularity and real-time scheduling Algorithm.

A. Dual-kernel Method

In order to make the Linux kernel become the real-time kernel that can be preempted completely, we adopt two kernels that they can work together on the same hardware platform, that is, a small real-time kernel is added between the hardware and general Linux kernel, which can manage interrupt and provides several necessary functions such as creation of low-level task, interrupt service program and so on, as shown in Figure 1.

![Fig. 1: Dual kernel structure for the real-time support of Linux](image)

Real-time core layer is responsible for hardware management and provides real-time task management, but general Linux core is seen as the lowest priority task (Idle task) of real-time core to schedule [6]. Real-time tasks run directly on real-time kernel, when there are no real-time tasks to run, general Linux core will be scheduled.

As for real-time kernel, it doesn’t always shut off hardware interrupt and receives all interrupt signals. When interrupt signals need to be processed through real-time process, real-time process will preempt Linux kernel. If interrupt signals need to be processed through general Linux, real-time kernel will capture interrupt through status bits of flag and interrupt flags, and decide how to deal with the interrupt. If the interrupt includes real-time handler, then the real-time handler will be called, or else, the signals will be passed to general Linux kernel. No matter what state the kernel Linux is in, which won’t prevent execution of real-time tasks, so we can achieve a real-time system.

B. Improvement of Soft Interrupt Mechanism

The threaded Interrupt is an important step to realize the real-time performance of Linux. After the interrupt is threaded, interrupt will serve as the kernel thread to run and be given a different real-time priority, real-time tasks can have higher priority than interrupt thread, so real-time tasks can serve as the highest priority execution unit to run, and has the real-time guarantee even in the severe load.

Now we will give the realization method of the threaded Interrupt. In kernel initialization stage, init_hardirq () creates a kernel thread for each IRQ. When the interrupts happen, the CPU will execute interruption response do_IRQ () to deal with the corresponding interrupt, interrupt response will judge whether each interrupt needs to be threaded or not, if need, then setting sign of thread and turn to the thread function --start_irq_thread (). In interrupt response stages, when the interrupt i occurs, system will call interrupt_i automatically, then execute do_IRQ, then enter the handle_IRQ_event and execute the interrupt handlers that be set in advance. Using do_IRQ function, we firstly judge whether the interrupt has already be threaded or not, if there is not, the interrupt will be executed in interrupt environment, which indicates this interrupt is real-time and the interrupt should be executed immediately,
otherwise this interrupt will execute the `start_irq_thread()` function in kernel thread environment, its priority will be lower than the priority of real-time process. If one of interrupts needs to be processed real-timely, it can use the sign of thread to declare itself not to be threaded. Interrupt handling workflow is described in Figure 2.

![Interrupt Handling Workflow](image)

**Fig. 2: Interrupt handling workflow**

### C. Improvement of Fine-granularity Timer

The clock granularity refers to the minimum time interval provided by OS, which is an important factor that influences the response speed of the system. In the time-sharing system, the clock interrupts are triggered by timers, each clock interrupt of Linux triggers three functions to work collaboratively, and to complete the selection and switch of process. There are three functions as follows ([7], [8]).

- Schedule (): process schedule function.
- Do_timer (): the main components of clock interrupt service.
- Ret_from_sys_call (): system call return function.

The traditional Linux timer only offer 10ms scheduling granularity, timing granularity is very rough, if we simply improve the clock frequency, the system load will be increased, thus to reduce system performance. As for many real-time systems, they usually need microsecond-level timer, so the scheduling granularity of general Linux can’t meet the requirements of response speed.

We adopt the solution to problem is to provide a fine-granularity timer on hardware. Here two types of timer mode are introduced in system, i.e. periodic mode and one-shot mode.

**Periodic mode** [8]: In order to realize microsecond-level granularity, we can simply program on the clock chip to make it run at a higher frequency. Clock interrupt is generated periodically by hardware and it is determined by a kernel parameter Hz. In Linux kernel, the value of Hz is 100, which means the cycle time of clock interrupt is 10ms. We can obtain 10 microsecond granularity through increasing Hz to 100000. However, increasing the value of Hz will increase greatly the consumption of system resources. Regardless of whether there are events that need to schedule, CPU always has been frequently interrupted. This periodic mode of timer can be applied in some strong cyclical situations.

**One-shot mode** [8]: What many real-time applications really need is not in every one microsecond clock interrupt occur, but in any one microsecond clock interrupts are allowed to occur, so the clock chip is not set to a fixed frequency any longer, but is set to the time for the next event to be occurred. This mode for setting clock chip is called one-shot mode.

After introducing fine-granularity timer in Linux kernel, the control functions for timer mode and granularity have been added in Linux system. According to the needs of different real-time applications to response time of tasks, we
can choose suitable timer mode and clock granularity.

D. Analysis and Improvement of Scheduling Algorithm of Real-time Task

1) Analysis of Scheduling Algorithm of Real-time System

Different applications will adopt different real-time scheduling strategies. Therefore, scheduling types of RTOS is directly related to its scope of applications. The following are usual real-time scheduling algorithms [9].

- TD: Time-driven scheduling algorithm.
- SD: Share-driven scheduling algorithm.

Each kind of scheduling algorithm has its own advantages and disadvantages, priority-driven scheduling algorithm and time-driven scheduling algorithm focus on hard real-time tasks, while Share-driven scheduling algorithm is more suitable for soft real-time tasks.

These three kinds of scheduling algorithm are decided according to the priority of process without taking the urgency and the time limit into account. In the high real-time response cases, these algorithms will fail to meet demands, many problems will appear in actual real-time system design, so the real-time scheduling algorithm need to be further improved and optimized.

2) Improvement of scheduling algorithm of real-time system

The From the above three scheduling algorithms, we know that, in the design of real-time systems, only considering process priority will go against the real-time improvement, we should consider comprehensively the process's priority and the urgency, so that it can greatly improve the real-time performance and ensure real-time process will be completed during the period of deadline. The following are improvement measures of scheduling algorithm.

In the improvement of scheduling algorithm, we will consider the process's priority and the urgency simultaneously.

\[ vi = wi + \frac{pi}{d - Ti} \cdot k \]

In the above formula, \( vi \) represents the priority number of process, \( wi \) represents the important degree of process and \( \frac{pi}{d - Ti} \) represents the urgency of process[9].

The larger value of urgency indicates the task is more urgent, the optimized scheduling algorithm still regards the value of the process as the foundation, at the same time, which also pays attention to the urgency degree of completing process. For the same priority of process, we use FIFO scheduling strategy. As for different value of process, its importance is also different, so CPU should execute the higher one.

Now we consider another question, for some processes, the value of them is very similar, but the urgency degree is very different, in accordance with the improved scheduling algorithm, we can judge the process's priority, and thus improve the real-time performance of system.

Now we will give a specific example to illustrate the scheduling algorithm.

Example: There are three processes A, B and C, which are submitted at the same time. A: the value is 1008, the estimated execution time is 3ms and the relative deadline is 5ms; B: the value is 1002, the estimated execution time is 1ms and the relative deadline is 5ms; C: the value is 1001, the estimated execution time is 1ms and the relative deadline is 2ms. Suppose the coefficient factor is 10, please give the execution sequence of the three processes.

Analyzing process: Obviously, we can see that the value of A process is the highest, through calculation, the priority is also the highest, so the A process will be run firstly. Process B and C, their values are very close, the value of B is more than C, but the urgency degree of C is more than B. Through calculation, the priority of B is 1004 and C is 1006, so we choose process C to run firstly, so we can avoid the unfair case that process C cannot be promptly
executed because of larger urgency degree. Table 1 describes the priority contrast of process A, B and C.

Table 1: The Priority Contrast for Process A, B and C

<table>
<thead>
<tr>
<th>Process</th>
<th>Value</th>
<th>Execute time</th>
<th>Urgency</th>
<th>Vi primary</th>
<th>Vi optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1008</td>
<td>3ms</td>
<td>5ms</td>
<td>1008</td>
<td>1014</td>
</tr>
<tr>
<td>B</td>
<td>1002</td>
<td>1ms</td>
<td>5ms</td>
<td>1000</td>
<td>1004</td>
</tr>
<tr>
<td>C</td>
<td>1001</td>
<td>1ms</td>
<td>2ms</td>
<td>1001</td>
<td>1006</td>
</tr>
</tbody>
</table>

The algorithm is realized by double linked list. In the normal load conditions of CPU, the optimized scheduling algorithm reflects more optimal real-time performance.

As for the improvement of real-time scheduling algorithm, we consider comprehensively the importance and urgency degree of process to decide the scheduling algorithm of priority. Algorithm links the two irrelevant concepts--deadline and value of process-- by formula together, which uses to calculate the priority number of process in a ready wait queue, thus to change the phenomenon that the priority is decided only with the value of the process in the current Linux real-time scheduling algorithm, and to ensure the deadlines of all process will be satisfied.

CONCLUSION

Linux operating system has become the hot spots in embedded field. At present, there exist various real-time approaches for Linux according to the characteristics of Linux kernel and the defects of real-time performance.

In this paper, we modify Linux to improve the real-time performance of Linux from the following aspects such as dual-kernel method, interrupt mechanism, clock granularity and real-time scheduling algorithm, so Linux will play a greater role in the occasions with high real-time requirements. With the developments of embedded field and improvement of real-time requirements, real-time algorithms and strategies of Linux will become more perfect.

ACKNOWLEDGMENT

I want to thank our research group for their helpful suggestions and sincere helps. I would also like to thank the lab's teachers for their providing me with a good experimental environment. All these have made me achieve good results. Here I will sincerely express my gratitude to them.

REFERENCES

[3]. DANIEL P.BOVET & MARCO CESATI. Understanding the Linux kernel, O’Reilly, 2001
[5]. Barabanov M. A linux-based real-time operating system(master thesis), New Mexico: NewMexico Institute of Mining and Technology, 1997
[7]. Y. Zhou, M.S. Li, Research and implementing of real-time support of Linux kernel, Journal of computer research and development, Vol.39, No. 4, April 2002