Analysis of Embedded Linux using Kernel Analysis System

Kiduk Kwon†, Midori Sugaya‡, Tatsuo Nakajima†
†Department of Computer Science and Engineering, Waseda University
3-4-1 Okubo Shinjuku-ku Tokyo, Japan
{pugara, tatsuo}@dcl.info.waseda.ac.jp
‡ Dependable Embedded OS Center, Japan Science and Technology Agency (JST), Tokyo, Japan
doly@dependable-os.net

Abstract

Recently, for embedded systems, the complexity of the software is rapidly increasing due to the advancement in the fields of multimedia and network. Due to these developments, it is difficult to find the cause of problems in system. This is especially true when the causes of problems are buried in the kernel layer, finding them more difficult to compare with the user layer. One reason of the difficulties comes from that there are no effective system which can analyze kernel problem available for developers.

In this paper, we propose a system named Kernel Analysis System (KAS) that can find errors and bugs in kernel. As a case study, the KAS find timer latency which is serious problem for the real-time processing system in kernel layer. It can effectively find the problem and support to find the cause of them. In the future, we try to generalize the system to find the other problems and causes to support the developers.

Keywords : Kernel Analysis System, Timer latency, High resolution timer, Embedded kernel, System analysis

1. Introduction

In the recent years, there is a growing need for improved reliability and quality for application due to the rapid increase of the applications related to multimedia in embedded systems [14]. However, errors or bugs are reoccurring in the kernel for embedded system. For example, a serious accident was occurred by the software bugs in electric motor break system in Chrysler automobiles in 2004. Similarly, in 2003, the bugs were occurred in the supplying system in electric power plant. It caused the electric power failure in wide area in U.S.A. Such kind of problems comes into the system with various reasons.

Among others, there is accuracy of timer which is a matter of importance for a time-sensitive application. Generally, there are many time critical applications in embedded systems that have a deadline until which they need to complete their jobs. For instance, multi-media applications are needed to be processed for their frames within some time constraints. The accuracy of timer is very important for the kind of applications, however, it is not guaranteed by the kernel. In kernel, there are a lot of elements that disturb the timer accuracy. In many cases, even developer cannot find when/where timer latency is occurring in kernel. For example, problem where the application does not detect a problem (that is, the error code paths have not been triggered), and problem without producing any error messages (hardware issues, memory corruptions, uninitialized memory, or a software bug causing a variable overflow) [16]. Especially, timer latency exerts a fatal effect on the embedded system such as automobile embedded system or medical embedded system. If we can pin-point when/where timer latency are occurring in kernel, we can thus find solution of problem. However, it is very difficult to find the timer latency in kernel.

It has been pointed out that other reason is the lack of analytical tools in the embedded systems for finding the causes. Effective tools for analyzing the system problem could enable us to more quickly and accurately analyze and solve the problem.

As a one of tools, we propose Kernel Analysis System (KAS) that can effectively find the problem and support to find the cause of the problem in kernel. In this paper, we focus on the methodology and tool development that can help to find the problem of timer latency. The problems occur in the kernel can be analyzed more easily and effectively through the use of this proposed system.

The remainder of this paper describes as follows. In Section 2, we discuss problem definition of the timer latency. In Section 3, we describe Kernel Analysis System. In Section 4, we present the result of experiment using KAS. In Section 5, we introduce previous case studies. Finally, In Section 6, we present conclusions and future work.

978-0-7695-3678-1/09 $25.00 © 2009 IEEE
DOI 10.1109/ICESS.2009.74

2. Problem definition

2.1. Background

Linux kernel have many advantages such as royalty free licensing, open source code, and availability of a large pool of skilled developers. In addition, Linux has been ported successfully to a large number of processor architectures, which allows it to run on many different types of CPUs. Therefore, many companies often use the Linux kernel for embedded system that not only has system stability but also easy to develop. Recently, embedded Linux focused on real-time applications and time-sensitive applications, which are characterized by temporal constraints. Such applications may require periodic execution where, for example, the period is derived from the frame rate of an audio/video stream [1]. Timer should provide accurate and precise time for the system. However, sometimes it cannot afford of it.

There are lots of timers such as periodic timer, HPET [17], and High Resolution Timer [6] that are implemented in Linux which are used for provide the time service for their applications. In these timers, the high resolution timer [6] is expected to provide highly accurate and precise time for time-sensitive applications. We analyze the timer as the case study. In this section, first we define the general model of the problem of timer, then, describe the specific problem of the High Resolution Timer.

2.2. General definition of the timer latency

At first, we need to define the general model of timer latency. The latency is defined as the differences of the accurate time, so we define the timer latency as follows.

Definition: Let t be a task belonging to an application that requires execution at time t, and let $t^{'}$ be the time at which t is actually happened. We define the timer latency as follows:

$$L^{timer} = t^{'} - t \tag{1}$$

2.3. Problem model in HRTimer

In order to consider about the cause of the problem, we need to discuss about the implementation of the timer. It is generally depends on the timer interrupt by the kernel. For example, Linux kernel has the function that handles the interrupts from hardware. It provides the facilities to support two types of interrupts. The one is the hardware interrupt that needs to response quickly, and the other is software interrupt which is less to have quick response. To improve the overhead and response time of interrupts, software interrupt functions are executed after expending hardware interrupts. Both of hardware and software interrupts can be cause of the timer latencies.

In this case study, we analyze the High Resolution Timer (General-purpose Linux resolution of timer could support up to one millisecond, however, the kernel that has hrtimer could support resolution of timer up to nanosecond). Generally, Linux uses periodic timer. Therefore if they want to the timer of high resolution then it needs to raise the clock frequency. Hrtimer structure allows the system to execute callback functions directly from the next event interrupt handler in x86 [4]. It makes possible without having to raise the overall system periodic interrupts. The softirq for running the hrtimer queues and executing the callbacks has been separated from the tick bound timer softirq to allow accurate delivery of high resolution timer signals. The execution of this softirq can still be delayed by other softirqs.

Although there is a difference of resolution supported depending on the system, currently it is possible to set up to $10\mu s$ in the kernel that supports hrtimer with less overhead. As compared with other timers, hrtimer also operates from interrupt occurrence.

We define the hrtimer model as follows:

- T^{lapic} - The period from occurring hrtimer hardirq to occurring hardirq to be expired.
- $T^{softirq}$ - The processing period to latency softirq after the occurrence hardirq.
- $T^{expired}$ - The period of high resolution timers (softirq handler) which is expired.

T^{time}, as indicated in formula 2, signifies the total time spent by timer during a period. HRT^{tick} signifies the time the developer set, and will have the same value in the case latency does not occur. However, as indicated in formula 2, the occurrence of latency in timer can be confirmed in the case $HRT^{latency} > 0$.

$$T^{time} = T^{lapic} + T^{softirq} + T^{expired} \tag{2}$$

$$HRT^{latency} = T^{time} - HRT^{tick} \tag{3}$$

Figure 1 shows the model of timer latency. The accuracy of timer in Linux depends on the accuracy of hardware and software interrupts. Timer interrupts are not occurring accurately when system is overloaded. It would cause timer latency in kernel. This paper conducts an experiment on finding the hrtimer latency based on proposed KAS. In this paper, we analyze the reason of the hrtimer latency by analyzing event log.

3. System Architecture

To analyze problems that occur in kernel, solution can be found by using the model of timer latency. In order to ana-

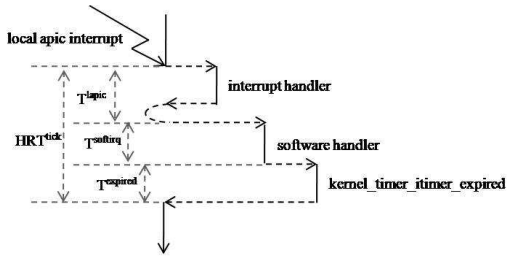

Figure 1. High resolution timer latency model

lyze the latency, not only timer related event but also all the information regarding to events (for example, system call, interrupt, thread, memory etc.) that occurred in kernel must be analyzed. Figure 2 shows the structure of the proposed system. The events occurred in kernel are logged using LTTng [5]. The logged data can analyze the problems by using KAS. It generates the results of analysis data according to each layer to analyze the source of a problem easily by analyzing a huge amount of event log.

KAS can be separated into three major steps.
•Latency Detection: Detect timer latency and count the number of them by using a huge amount of log data.
•Data Separation: Separate the event log from the entire data using the position information that created in Latency Detection.
•Analysis: Generate the statistics data such as the frequency of event execution, the time spent for each event, and the overall accumulated time.

A problem solution can be more easily and effectively found by analyzing the cause of the problem using the results from the above three steps.

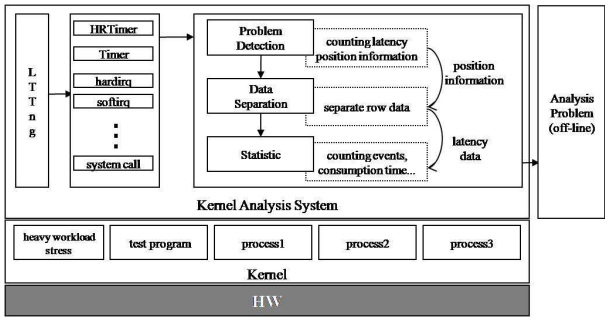

Figure 2. Proposal of architecture

4. Evaluation of HRTimer Latency

This section address the specification of experiments set up and evaluation of hrtimer latency. The system is with a 1.83GHz Intel Pentinum4 uniprocessor and 1GB RAM, on which running a Linux Kernel 2.6.24.

4.1. Setup for Measurement

This paper attempts to solve the timer latency by analyzing event log and to explorer the reason of timer latency. First of all, we apply LTTng patch to the Linux kernel in order to collect event logs. Next, it needs to circulate the hrtimer regularly. We use *setitimer()* system call to send SIGALRM signal to processor when timer is finished, the function of *setitimer()* occurs interrupts in the process itself at certain future time. We set HRT^{tick} as $100\mu s$ and set the cycle of repetition as 10,000 with heavy background load.

The $HRT^{latency}$ analysis is based on a loop that:
1. reads the hardirq for hrtimer T^{lapic}
2. reads softirq of hrtimer $T^{softirq}$
3. reads itimer_expired time $T^{expired}$
4. computes formula 2 and formula 3

4.2. Analysis of HRTimer Latency

The experiment of hrtimer latency is based on the proposed system architecture as shown in Figure 2. There are three layers to analyze the event log explained in Section 3.

Table 1. HRTimer latencies(in ns)

$HRT^{latency}$	latency count	$HRT^{latency}$	latency count
100,032	1	93	-
116,693	2	5,393	-
1,423	-	100,465	16
...	...	1,423	-
100,806	15	-548	-
950	-	103,898	17

Table 1 is a part of data from Latency Detection layer. T^{time} - HRT^{tick} indicates hrtimer latency. We defined equation $HRT^{latency} \geq 100\mu s$ as latency, and record the number of latency where latency count as Table 1. It records not only the latency time but also the position information where the latency is occurred. In addition, *irq entry/exit* time, *softirq entry/exit* time, entry time of hrtimer handler, and the expired time of hrtimer are recorded in Latency Detection. Data Separation layer is to separate the real event data in the occurrence of timer latency from the whole the

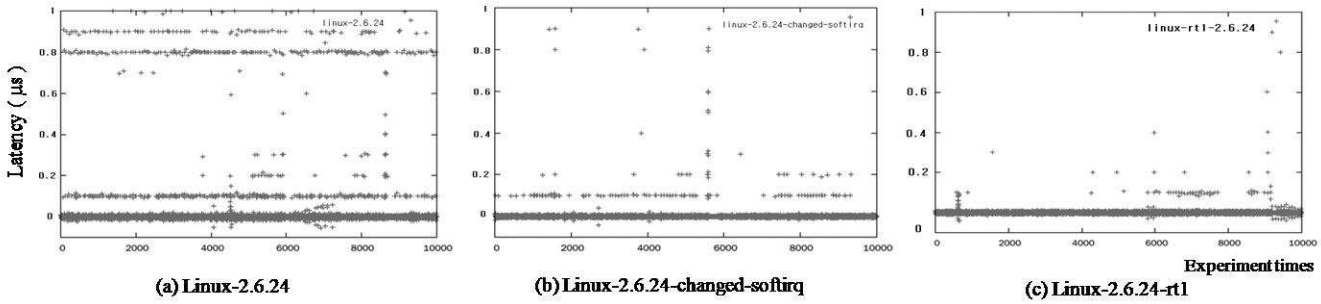

Figure 4. Results of experiments of hrtimer latency with heavy background load.

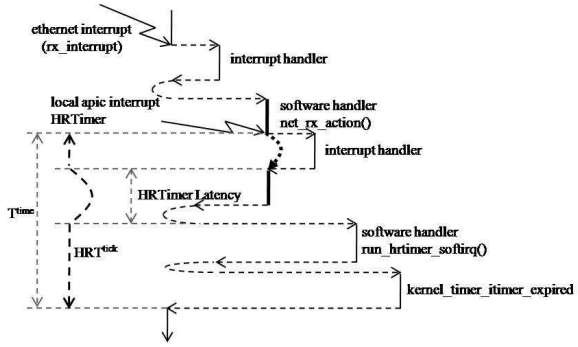

Figure 3. Hrtimer interrupt when latency is occurred

event log. At this time, position information generated from Latency Detection layer to separate event log is used. The last layer, Analysis, is to get statistical data from separated event log of the second layer. We record that all event names, event generation frequencies, and consumption time of each event that every single event executes during T^{time}.

Figure 3 shows hrtimer latency after latency analysis. Time of $HRT^{latency}$ is occurring at over $100\mu s$ of latency. However, we cannot analyze hrtimer latency accurately only with Table 1. Hence, it needs to analyze hrtimer latency with the data from the three layers.

There is not only one reason of latency in the kernel but also complex factors of latencies coming from the kernel; the reasons could be followed by the dependencies with hardirq, softirq, and other processes [10, 15]. It is difficult to solve the problem by one perfect solution. However, the system performance will be improve, when just one problem is solved and analyzed accurately from the cause of a lot of problems. After hrtimer latency experiment and analyzing event log by using KAS, the most problem is the priority of softirq. In Linux, the case of softirq had designated priority. In case of *__do_softirq()* after processing occurrence of hardirq event, it operates the specified softirq handler and it fits in priority. While operating softirq handler (for example, *NET_RX_SOFTIRQ* or *BLOCK_SOFTIRQ*), even if hardirq of hrtimer has occurred, hrtimer is on standby because it cannot operate *__do_softirq()* (*HRTIMER_SOFTIRQ*) due to its lowest priority in the Linux-2.6.24. Hence, softirq can be one of the reasons caused hrtimer latency in the Linux-2.6.24. Above all reasons, we have done hrtimer latency experiments after increasing the priority of softirq.

Figure 4 is the result of hrtimer latency experiment in the kernel. We repeat this experiment which runs for 10,000 activations in ten sets. Therefore, total number of measurement is 100,000 activations. In the experiment, we give network stress and I/O stress higher priority of softirq than hrtimer. The result for latency experiment of the hrtimer with Linux-2.6.24 is shown in Figure 4(a). The frequency of latency occurrence without measuring stress program is lower than measuring hrtimer latency in the Linux2.6.24 as 579 times out of 100,000. However, the result of latency occurred 5,546 times out of 100,000 when measuring hrtimer latency in the Linux-2.6.24 with heavy background load. Figure 4(b) is the result of measuring hrtimer latency whose priority of *HRTIMER_SOFTIRQ* is set by high priority. Its priority is higher than *NET_RX_SOFTIRQ* and *BLOCK_SOFTIRQ* in Linux-2.6.24 kernel. The way of measurement of this experiment is the same as Linux-2.6.24 experiment as well as the number of measurement. As Figure 4(b), rate of hrtimer latency with switched softirq in Linux-2.6.24-changed-softirq, gets lower than Linux-2.6.24 kernel. In the other words, frequency of occurrence of latency changes 5,546 to 1,620, which is considered. The rate of hrtimer latency gets lower but there is still hrtimer latency. Therefore, we can infer that there is another reason of latency and it is different from what we explained above. Figure 4(c) shows the result of analysis the cause that is the generated hrtimer latency in Linux-2.6.24-rt1 by using KAS. It is compare Linux-2.6.24-rt1 with Linux-2.6.24-changed-softirq. In the Linux-2.6.24-rt1, hrtimer la-

Table 2. Statistical data from a cycle of latency

Event name	Execution time	Consumption time	Event name	Execution time	Consumption time
kernel_arch_syscall_entry	114	269,678	mm_page_free	9	18,643
kernel_arch_syscall_exit	114	129,687	fs_writev	2	1,616
kernel_sched_try_wakeup	8	16,628	fs_write	1	1,010
kernel_timer_itimer_expired	1	688	fs_read	4	3,317
kernel_softirq_raise	2	2,976	fs_ioctl	4	3,126
kernel_softirq_entry	3	4,182	fs_polled	24	19,678
kernel_softirq_exit	3	2,257	fs_select	66	54,945
kernel_timer_set	4	4,013	kernel_sched_schedule	6	9,645
kernel_timer_update_time	1	1,955	input_event	7	30,820
kernel_send_signal	2	2,939	net_socket_call	85	53,884
kernel_irq_entry	3	15,591	net_socket_sendmsg	85	68,806
kernel_irq_exit	3	9,675	net_dev_receive	6	4,761
mm_page_alloc	351	553,913	net_dev_xmit	6	15,093
Total latency time(in ns) : $HRT^{latency} = T^{time} - HRT^{tick}$ = 1,203,542					

tency occurred 608 times out of 100,000. The result is lower than in Linux-2.6.24 and in Linux-2.6.24-changed-softirq.

Table 2 is generated by Analysis layer when the latency is generated. It is shown that generated the event name, frequency of execution, and consumption time of each event. The total latency is 1,203,542ns and the event that causes latency generation is *kernel_arch_system_entry/kernel_arch_system_exit* and *mm_page_alloc* event. Especially, the *mm_page_alloc* event occurred latency of 553,913ns. Based on these results, data that generated by Data Separation layer is analyzed. As an experiment, excessive network stress and I/O stress is given to the system for experiment of latency. Consequently, we can find the timer latency when the timer latency is occurred, because *mm_page_alloc* and *kernel_arch_system_entry/kernel_arch_system_exit* increased rapidly in excess by frequent generation of *NET_RX_SOFTIRQ* and *BLOCK_SOFTIRQ*. Rate of latency in each kernel is shown in Figure 5.

As a result, we can analyze hrtimer latency more easily and efficiently by using proposed KAS.

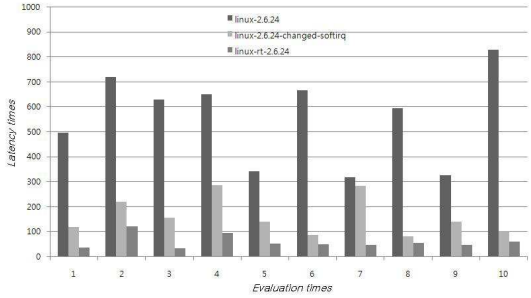

Figure 5. Times of latency in general-purpose Linux kernel, Linux-2.6.24-changed-softirq and Linux-2.6.24-rt1 kernel

5. Related Work

Embedded systems are getting more complicated and the analysis and solution of performance problems is available to debug the ability of engineer. However, there is a restriction of individuals experience and its intuition to solve the system problems. Therefore, we need new system that can more easily analyze and measure the performance of kernel for developers who program system software or application software.

It is well known that LTTng [9] is a new version of LTT. By using LTTng, we can copy and record the event occurring inside of the kernel such as thread, fork, interrupt, signal, and memory information, etc from the kernel space to user space quickly. In addition to using LTTV (Linux Trace Tool Viewer), we can record and review the event log visually, and the overhead is reduced from 1.54 to 2.28 [3].

There is LKST (Linux Kernel State Tracer) [11] among event trace tools occurring in the Linux kernel. LKST performs similarly performance to LTTng and it can record event occurring inside of the kernel such as process context switch, sending signal, exception, memory allocation, sending packets, etc. LKST can analyze the problems in the kernel mode [7] and users can expand the performance dynamically.

Another measuring and analyzing tool is Mevalet [8].

Mevalet is developed by system software department of NEC. Mevalet program hooks point inside kernel of embedded systems which is the object of measuring in order to log event. Mevalet is also applicable to any computer languages. In addition, Mevalet can analyze every application by measuring the performance from several tens of ns to several tens of s without changing application, and the overhead by several percentages.

Kernel Function Tracer (KFT) [13] and Kernel Function Instrumentation (KFI) [2] are also to solve the same problems. Kernel Function Instrumentation has higher overhead than LTTng, but it can log event in more detail.

As mentioned above, an existing research can be roughly divided with profiling tools and event-based performance monitoring tools. Profiling tools has gprof, prof, tprof, time, top, nmon, and oprofile, etc., and event-based performance monitoring tools has LTT, SystemTAP, and LKST, etc. Such tools monitor and profile the system, but it does not analyze the reason, when problems are occurred.

Above all, there are various tools that can log event occurring in the kernel, however, the purpose of this paper is a little different from event tracing. The purpose of this paper is to analyze the kernel problems easily and effectively by using event log, and to propose system architecture which can explore the reason of problem.

6. Conclusions and Future work

In this paper, we propose the Kernel Analysis System to analyze and solve the problem in kernel more easily and efficiently. As a result, we realize that one reason of hrtimer latency is the priority of softirq. Consequently, we can analyze the reason of problem in the kernel easily and efficiently by using the proposed system architecture. However, there is not only one reason of problem but also complex reasons of problem shown in the kernel. Therefore, there are also reasons of latency not only from priority of softirq but also from hardirq or interrupt locking etc.

We move to focus on improving the proposed system architecture to analyze various reasons of problem accurately and to show the visual data more easily.

References

[1] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, A measurement-based analysis of the real-time performance of the Linux kernel. In Real-Time Technology and Applications Symposium (RTAS02), Sept. 2002.

[2] Tim Bird, Learning the kernel and finding performance problems with kfi. In CELF International Technical Conference, 2005.

[3] Mathieu Desnoyers and Michel R. Dagenais, The lttng tracer: A low impact performance and behavior monitor for gnu/linux. In OLS (Ottawa Linux Symposium) 2006, July.

[4] Thomas Gleixner, Douglas Niehaus, Hrtimers and Beyond: Transforming the Linux Time Subsystems. Ottawa Linux Symposium 2006.

[5] LTTng project, http://ltt.polymtl.ca/

[6] G.Anzinger, High resolution timers project. http://high-res-timers.sourceforge.net/.

[7] Yaghmour, K. and Dagenais, M. R., Measuring and characterizing system behavior using kernel-level event logging. In Proceedings of the Annual Technical Conference on USENIX Annual Technical Conference, 1326, 2000.

[8] SystemDirector mevalet, http://www.nec.co.jp/cced/mevalet/

[9] Mathieu Desnoyers and Michel Dagenais, Low disturbance embedded system tracing with Linux Trace Toolkit Next Generation. In ELC (Embedded Linux Conference) 2006.

[10] Martin Bligh, Mathieu Desnoyers and Rebecca Schultz, Linux Kernel Debugging on Google-sized clusters. Proceedings of the Linux Symposium June, 2007, Ottawa, Ontario in Canada .

[11] Linux Kernel State Tracer, http://lkst.sourceforge.net/

[12] Debugging with DataDisplay Debugger, Users Guide and Reference Manual First Edition, for DDD Version 3.3.9. 15 January, 2004.An Introduction to the Real-time OS & Nucleus PLUS Training Guide. Accelerated Technology Inc.

[13] Kernel Function Trace, http://elinux.org/Kernel_Function_Trace.

[14] Nucleus, An Introduction to the Real-time OS & Nucleus PLUS Training Guide. Accelerated Technology.

[15] Yu-Chung and K.-J Lin, Enhancing the Real-Time Capability of the Linux Kernel. In Proceedings of the IEEE Real Time Computing Systems and Applications, Hiroshima, Japan, October 1998.

[16] Mark Wilding and Dan Behman, Self-Linux Mastering -The Art of Problem Determination.

[17] High Precision Event Timers Specification, Inter Corporation