Profiling Tool for Booting Sequence

Jaeseop Kim
Soongsil University
Seoul, Korea
jskim.oslab@gmail.com

Gwangyong Kim
Chungbuk National University
Cheongju, Korea
brightdragom@gmail.com

Bongjae Kim
Chungbuk National University
Cheongju, Korea
bjkim@cbnu.ac.kr

Jaehoon An
Korea Electronics Technology Institute
Seongnam, Korea
corehun@keti.re.kr

Jiman Hong*
Soongsil University
Seoul, Korea
jiman@ssu.ac.kr

ABSTRACT

The Baseboard Management Controller is a specialized service processor that allows administrators to perform remotely inspecting hardware devices in racked servers. By using the Baseboard Management Controller, Administrators of servers can quickly inspect hardware in case of server failures. A fast booting time of the Baseboard Management Controller is essential for inspecting hardware quickly. To reduce the booting time, the booting procedure which is the process of starting a computer as initiated must be optimized by profiling the CPU, the memory usage, the function calls information, and so on. In this paper, we propose a profiling tool prototype for analyzing and optimizing the booting procedure of a Linux-based BMC system. The proposed tool provides profiling information of hardware on the main board dynamically, and a comparison functionality according to kernel changes.

KEYWORDS

BMC, Booting-time, Profiling, Linux

1 INTRODUCTION

The Baseboard Management Controller (BMC) allows administrators to remotely perform tasks that require a physical visit to the server, such as server power cycling, fan speed/component temperature monitoring, and hardware failure [1]. The administrator can quickly hardware inspect through the BMC in the event of a server failure [2]. The BMC's quick booting time is essential for quickly inspecting hardware. In order to reduce the booting time, a trace tool should be used to profile and optimize the time required for function calls, and CPU and memory usage during the booting process [3]. However, existing profiling tools do not provide integrated information on CPU, memory usage, and function call information for the booting procedure.

In this paper, we propose a profiling tool prototype that provides information required for booting procedure analysis. The proposed profiling tool focuses on the device driver and file system initialization process, which takes up most of the run time in the booting procedure. The proposed profiling tool provides function call information, required time, and CPU and memory utilization information. To provide CPU and memory utilization information, CPU and memory usage information are collected and post-processed for each device and file system initialization and provided in graph form. In addition, it provides summation and sorting functionalities for the number of function calls and execution time based on *ftrace* utility to find the delay section during the booting procedure. Finally, it provides functionality to track and compare changes for each kernel version changed during the booting procedure optimization process.

The remainder of the paper is organized as follows. Chapter 2 presents methods for collecting profiling information. Chapter 3 presents the prototype interface of the proposed profiling tool. Finally, concluding this paper in Chapter 4.

2 PROFILING

AST2600-EVB [4], BMC evaluation board, is used for collecting information from sensors on a server for inspection. The collected information is post-processed in order to be provided to the user through the profiling tool. Table 1 shows brief specifications and software version information of AST2600-EVB.

Table 1: AST2600-EVB specifications

Features	Descriptions
CPU	Dual-core ARM Cortex A7
SDRAM Memory	2GB DDR4 SDRAM with speed grade higher than DDR4-1600Mbps
Flash Memory	SPI flash memory
BMC	BMC controller with IPMI 2.0/1.5 compliant
Kernel Version	5.4.62

* Corresponding Author

2.1 Required Time Information for Initialization

Generally Booting time is affected by many factors but initializing device drivers and file system initialization tasks consume most of the booting time. Therefore, it is essential to optimize the initializing time, to reduce the booting time.

The *initcall* mechanism operation process is traced to provide CPU and memory utilization information in the initialization process of each device driver and file system. The *initcall* is a Linux kernel mechanism for initializing built-in device drivers and file systems in the correct order [5].

Passing "initcall debug=1" to the kernel command line to collect device driver and filesystem initialization information from booting messages through the *initcall* mechanism. This allows the booting message to include the name of the initialization function, return value, and initialization execution time, as shown in Fig. 1. For using the *initcall* mechanism, "CONFIG_PRINTK_TIME" should be included in the kernel compilation option. This option is used to print the function execution time to the console.

```
$ dmesg | grep “initcall”
initcall inet6_init+0x0/0x32c returned 0 after 10307 usecs
initcall ip6_tables_init+0x0/0x9c returned 0 after 20 usecs
initcall ip6table_filter_init+0x0/0x64 returned 0 after 240 usecs
initcall ip6table_mangle_init+0x0/0x94 returned 0 after 148 usecs
initcall nf_defrag_init+0x0/0x5c returned 0 after 28 usecs
initcall nf_log_ipv6_init+0x0/0x64 returned 0 after 3 usecs
initcall ipv6header_mt6_init+0x0/0x24 returned 0 after 0 usecs
initcall reject_tg6_init+0x0/0x24 returned 0 after 1 usecs
initcall sit_init+0x0/0xd0 returned 0 after 7099 usecs
initcall packet_init+0x0/0x88 returned 0 after 4883 usecs
```

Figure 1: Booting messages with “initcall” keyword

2.2 CPU and Memory Utilization Information

The implementation to collect CPU and memory usage during device driver and file system initialization is as follows. The functionality for profiling CPU and memory usage is implemented in “linux/init/main.c” by using “kcpustat” and “sysinfo” kernel structures that store CPU and memory usage [6]. Tables 2 and 3 show the necessary items in the “kcpustat” and “sysinfo” structures for CPU and memory utilization calculations. The unit “USER_HZ” in Table 2 uses 100 in most architectures, but it can be changed arbitrarily according to the user. If the “USER_HZ” value is 100, it means 1/100 of a second.

Table 2: CPU usage data (Unit : USER_HZ)

Items	Descriptions
user	Time spent in user mode
nice	Time spent in user mode with low priority
system	Time spent in system mode
idle	Time spent in the idle task
io_wait	Time waiting for I/O to complete
irq	Time servicing interrupts
softirq	Time servicing softirqs

Table 3: Memory usage data (Unit: Kilobyte)

Items	Descriptions
MemTotal	Total Usable Memory
MemFree	Amount of free memory

On lines 1, 5, and 9 in Table 4, the message containing the “calling” keyword is the starting point of the device driver and file system initialization process. The data in Tables 2 and 3 are printed on lines 2, 3, 6, 7, 10, and 11. Lines 4, 8, and 12 containing the “initcall” keyword are the points where each initialization process is finished. In this manner, CPU and memory usage information are collected at each initialization process.

Table 4: Booting messages of CPU and memory usage information

line	message
1	calling sock_diag_init+0x0/0x50 @ 1
2	[OSLAB] user : 2, nice : 0, system : 444, idle : 409, iowait : 0, irq : 0, softirq : 0
3	[OSLAB] MemTotal : 924852, MemFree : 789712
4	initcall sock_diag_init+0x0/0x50 returned 0 after 152 usecs
5	calling nfnetlink_init+0x0/0x84 @ 1
6	[OSLAB] user : 2, nice : 0, system : 444, idle : 409, iowait : 0, irq : 0, softirq : 0
7	[OSLAB] MemTotal : 924852, MemFree : 789484
8	initcall nfnetlink_init+0x0/0x84 returned 0 after 40 usecs
9	calling nfnetlink_log_init+0x0/0xb4 @ 1
10	[OSLAB] user : 2, nice : 0, system : 444, idle : 409, iowait : 0, irq : 0, softirq : 0
11	[OSLAB] MemTotal : 924852, MemFree : 789484
12	initcall nfnetlink_log_init+0x0/0xb4 returned 0 after 38 usecs

2.3 Kernel Function Call Information

ftrace utility can be used to analyze delays occurring at specific parts of the booting procedure [7]. Users can check the function call flow and execution time for each function by using the filtering functionality for a specific function. Fig. 2 shows the part of the *ftrace* utility result and is an example of applying “ast8250_probe” to the trace function filter. As seen in Fig. 2, the default information provided by *ftrace* utility does not include the sum of the execution times and counts for each function. In addition, there is no support for sorting by the required time. To solve this problem, the proposed profiling tool includes a sorting

functionality based on the function execution time and the number of total calling times. The function name, execution time, and calling frequency from the "/sys/kernel/debug/tracing/trace" file is collected to provide sorting functionality.

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |    |                     |   |   |   |
 0)               |  ast8250_probe() {
 0)               |    devm_kmalloc() {
 0)               |      __kmalloc_track_caller() {
 0)   1.605 us    |        kmalloc_slab();
 0)   1.190 us    |        should_failslab();
 0)   8.249 us    |      }
 0)               |      devres_add() {
 0)   1.310 us    |        _raw_spin_lock_irqsave();
 0)   1.387 us    |        add_dr();
 0)   1.210 us    |        _raw_spin_unlock_irqrestore();
 0)   8.364 us    |      }
```

Figure 2: *ftrace* utility in the booting procedure

3 PROFILING TOOL

3.1 Kernel Version Management Window

As shown in Fig. 3, the performance can be compared and analyzed in the Kernel Version Control window by selecting one or two kernel versions. For the functionality of kernel versioning, the kernel version being developed is compressed each time the kernel is built. It is stored with the hash value of each file for later code comparison and performance analysis.

3.2 Code View Window

As shown in Fig. 3, if a specific kernel version is selected, in Code View Window, the structure of the directory and file of the kernel source code is displayed in a tree form. If the user chooses each source code file, the source code is displayed. In addition, if the user selects two kernel versions that the user wants to compare, only the parts where the source code differs are displayed selectively.

3.3 Performance View Window

As shown in Fig. 3, the CPU and memory utilization are displayed as a graph in the form of a line chart. The graph's X-axis means the progress time after power is applied (unit: seconds). The Y-axis is CPU and memory utilization (unit: Percentage). The users can check where the CPU and memory utilization is high.

3.4 Function View Window

As shown in Fig. 3, the function call information is also provided in the function view window. FUNCTIONS, TIME, and COUNT indicate the function name, the total execution time of each function in microseconds, and the total number of calls of each function respectively. Function View Window also allows the user to sort their searches by TIME and COUNT.

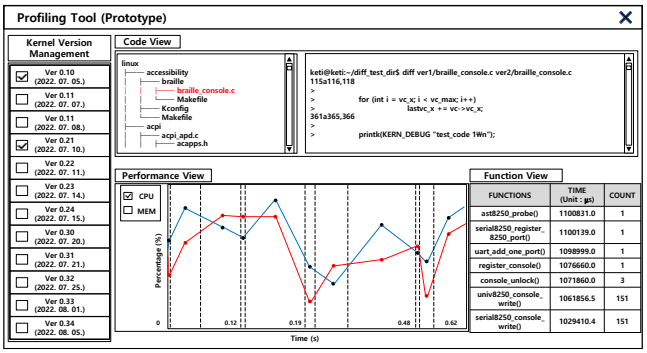

Figure 3: Prototype of Profiling Tool

4 CONCLUSIONS

This paper proposed a booting procedure profiling tool prototype to reduce the booting time of Linux-based BMC. The proposed profiling tool provides CPU and memory utilization in graph form, focusing on the device driver and file system initialization process. It provides the summary of information on the number of function calls and execution time based on *ftrace* utility to find the delay section in the booting procedure. Finally, the proposed tool can compare and analyze the performance of the two different Kernel versions.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2022-0-00202, Development of Intelligent BMC SW to reduce the power of server)

REFERENCES

[1] Jessie Frazelle. 2019. Opening up the Baseboard Management Controller: If the CPU is the brain of the board, the BMC is the brain stem. *Queue* 17, 5 (Sep-Oct. 2019), 5-12. https://doi.org/10.1145/3371595.3378404

[2] Ghazanfar Ali, Jon Hass, Alan Sill, Elham Hojati, Tommy Dang, and Yong Chen. 2022. Redfish-Nagios: A Scalable Out-of-Band Data Center Monitoring Framework Based on Redfish Telemetry Model. In *Proceedings of Fifth International Workshop on Systems and Network Telemetry and Analytics* (SNTA '22). Association for Computing Machinery, New York, NY, USA, 3–11. DOI: https://doi.org/10.1145/3526064.3534108

[3] Tomi Kyöstilä. 2018. *REDUCING THE BOOT TIME OF EMBEDDED LINUX SYSTEMS*. Master's thesis. University of Oulu, Oulu, Finland.

[4] ASPEED, 2019, https://www.aspeedtech.com/server_ast2600/

[5] Mylene Josserand. 2020. Demystifying Linux Kernel initcalls. https://elinux.org/images/e/e8/2020_ELCE_initcalls_myjosserand.pdf

[6] Linus Torvalds. 2022. Linux (v5.4-rc6). https://github.com/torvalds/linux

[7] Masami Hiramatsu. 2020. The Linux Kernel/Boot-time tracing. https://www.kernel.org/doc/html/v5.8/trace/boottime-trace.html