Optimization Methodology and Case Studies for Linux Fast Booting

Gwangyong Kim
Department of Electrical, Electronics, information and Computer Engineering
Chungbuk National University
Chungju, Korea
brightdragom@chungbuk.ac.kr

Jinsung Cho
Department of Electrical, Electronics, information and Computer Engineering
Chungbuk National University
Chungju, Korea
cjs97111@chungbuk.ac.kr

Jaeseop Kim
Department of Computer Science and Engineering
Soongsil University
Seoul, Korea
jskim.oslab@gmail.com

Ajung Kim
Department of Computer Science and Engineering
Soongsil University
Seoul, Korea
ajkim.oslab@gmail.com

Bongjae Kim*
Department of Computer Engineering
Chungbuk National University
Chungju, Korea
bjkim@chungbuk.ac.kr

Jiman Hong
Department of Computer Science and Engineering
Soongsil University
Seoul, Korea
jiman@ssu.ac.kr

ABSTRACT

Recently, the amount of data has increased rapidly, and attention has been paid to how data is processed and managed. The data center market continues to grow because it can process and manage large amounts of data for various services. The demand for IT infrastructure construction is exploding. The number of data centers in 2021 was 1,851, with an average annual growth rate of about 15.9%. It is expected that the BMC (Baseboard management controller) for system service management, monitoring, maintenance, and control of the data center will grow with the growing data center market. Optimization of the boot process must be successful in supporting the stability and quick start of the BMC. This paper discusses the optimization methodology and implementation examples studied for developing Fast Booting technology for the Linux operating system, mainly utilized in BMC.

KEYWORDS

BMC, Fast Booting, Datacenter

1 INTRODUCTION

The development of the Internet and the fourth industrial revolution has progressed, and various amounts of data are rapidly increasing. As a result, the market size of server equipment or data centers that handle tasks and services on a variety of data is continuously growing. According to IDC Korea (International Data Corporation Korea), an IT market analysis and consulting firm, the server market will grow at an average annual growth rate of 9.1% over the next five years. BMC and IPMI (Intelligent Platform Management Interface) technologies are being developed as the server and data center markets continue to grow.

* Corresponding Author

BMC gathers information about the network and hardware devices' physical status, such as temperature, humidity, power supply, fan speed, OS operation, etc., by using various sensors. Through gathered sensor data, BMC warns administrators of system issues or suspicions. A rapid start-up speed should be guaranteed for administrators to monitor and take action quickly. Optimizing the processes used for booting, settings and CPU, memory use, etc., is vital to ensure that BMC boots up fast.

This paper describes the optimization methods in the boot process to ensure the fast boot speed of BMC. The structure of this paper is as follows. Chapter 2 introduces the optimization methodology for a fast start-up. Chapter 3 examines implementation examples of various optimization methodologies, and Chapter 4 concludes this paper with future works.

2 OPTIMIZATION METHODOLOGY FOR FAST BOOTING

2.1 Linux Boot Process

Figure 1: Linux boot process.

Generally, the Linux boot process consists of 4 steps, as shown in Fig. 1. First, a power supply step for operating hardware and a POST (Power On Self Test) step for self-checking through a ROM-BIOS program after the power supply is performed. If all devices' status is normal, control permission is passed to the loaded bootloader, and the BIOS exits. The bootloader imports the Linux

kernel images to the memory. Then the kernel loads and calls the process with process ID 0. The called swapper process works by running an init process that initializes the various drives that use the attached equipment.

2.2 Bootloader Time Optimization

Optimization at the Bootloader step during the boot process can be done either by optimizing early in the Bootloader step or by optimizing the kernel image loading and decompression process. The most time-consuming part of the bootloader process is unpacking the kernel image and kernel-related data. Therefore, research has been conducted to use compression and decompression techniques to reduce boot time. For example, we can use advanced decompression algorithms such as UCL to optimize the time spent on processes related to compression over gzip decompression.

One of the other methods is skipping the most time-consuming image, data loading, and compression steps. There is XIP (eXecute-In-Place) technology. XIP technology directly executes and processes kernel images and data in Flash memory without importing them into RAM and processing them, allowing optimization by skipping the process of copying to RAM and compression [1]. In addition, a snapshot-based technique can be applied. The snapshot-based technique stores kernel parameters and kernel images in snapshot images. After the bootloader initializes the device, copy the snapshot image to the original address and boot the system, optimizing boot time by skipping the kernel image loading process.

2.3 Optimization Progressed during Kernel Loading

Optimization is mainly carried out at the kernel level via minimizing extra work or lengthy kernel loading times. There is an easy way to disable and remove printk() calls and logs made during bootup. Reconfiguring several kernel-dependent device drivers is an additional approach. The Linux kernel was created with modules that were designed to meet a variety of needs. However, several pointless modules with functions that are not necessary may also be present, or a module may contain useless functions. Accordingly, based on the hardware's specification in which the kernel runs, optimization can be done by deleting or decreasing superfluous modules and functions [2,7]. For instance, the TCP/IP module can be disregarded for an embedded device that doesn't need networking. Additionally, parameters can be adjusted to prevent needless probing work [3].

2.4 Optimizing Mounting the Root File System

Linux booting might be faster because a smaller root filesystem will make mounting happen more quickly. However, performance may degrade if optimization is carried out by altering the settings of the original file system to speed up booting. This issue can be resolved by employing a different type of file system. The file system mount time is decreased by putting each application in its own partition and only mounting the partitions that the user is likely to use immediately after launch [4].

3 CASE STUDIES FOR LINUX FAST BOOTING

Table 1 summarizes the case studies for Linux fast booting described in the paper. As shown in Table 1, it supports fast booting within 5 seconds, even in embedded environments. In the case of the [5] and [6], it takes less time for booting only because the measured time is the time until the application starts as shown in Table 1.

Table 1: Summary of case studies for Linux fast booting

Processor	Spec	Booting Time (Unit: s)	Description	Ref.
OMAP 5912	192MHz@ ARM926EJ dual core	3.19	Time required to start the application	[5]
S5PC 100	667MHz@A RM Cortex-A8	4.54	Time required to start the application	[6]
CLM 9722	800MHz@A RM Cortex A8	2.66	Time required to boot only	[7]

3.1 A Fast Booting Technique using Improved Snapshot Boot in Embedded Linux [5]

In [5], the boot time of personal portable embedded devices running the embedded Linux operating system is reduced using the improved Snapshot boot technique. Both the kernel image and the data are saved in the snapshot when performing resumption. Additionally, the boot time is reduced by preserving only the pages required for booting in the snapshot instead of the complete page.

3.2 Fast Booting Solution with Embedded Linux-based on The Smart Devices [6]

In [6], the bootloader and script execution steps have been tuned for fast booting. The bootloader step no longer uses compressed images and removes all redundant routines and waiting times. The script execution portion of the boot process is optimized utilizing the minimum init process and the binarization-based script replacement technique after mounting the root file system.

3.3 Optimizing Boot Stage of Linux for Low-power ARM Embedded Devices [7]

In [7], it employs several methods to reduce boot time by using an ARM development board. The development environment's keyboard, LCD controller, and message output of initialization processes were dropped to improve bootloader initialization. The

boot time was reduced during the kernel initialization step by altering certain settings using kernel settings, specifying the initial values of internal variables using kernel boot parameters, and changing the initialization order of each module using the Kbuild system.

4 CONCLUSIONS AND FUTURE WORKS

This paper explained the Linux fast booting optimization methodology and some case studies. For fast booting of Linux, methods such as boot loader optimization, kernel optimization, and file system optimization can be used. Fast booting is a basic technology needed in BMCs, and it is anticipated. In future works, we have a plan to research reducing the boot time based on an actual BMC reference board that development will continue on it in the future.

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] ParkSungho, ShineKwangmu, KimYoungju. "A Fast Booting Scheme of Linux for Embedded System." "Journal of the Korea Institute of Information and Communication Engineering" (The Korea Institute of Information and Commucation Engineering), 2006: 2173-2180.

[2] ShinKwangmu, ParkSeongho, ChungKidong. "Fast Booting Implementation of the Linux in the Embedded System." "Proceedings of the Korean Information Science Society Conference", 2005: 853-855.

[3] Bird, TimR. " Methods to improve bootup time in Linux. " " LInux Symposium." 2004. 79-88.

[4] DEY, Swarnava; DASGUPTA, Ranjan. Fast boot user experience using adaptive storage partitioning. In: 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns. IEEE, 2009. p. 113-118.

[5] ParkSejin, SongJaehwan, ParkChanik. "A Fast Booting Technique using Improved Snapshot Boot in Embedded Linux." "Journal of KIISE:Computing Practices and Letters", 2008: 594-598.

[6] LeeGwanglo, BaeByeongmin, ParkHojun. "Fast booting solution with embedded linux-based on the smart devices." "Proceedings of the Korean Institute of Information and Commucation Sciences Conference." Republic of Korea: The Korea Institute of Information and Commucation Engineering, 2012. 387-390.

[7] KIM, Jongseok; YANG, Jinyoung; KIM, Daeyoung. Optimizing Boot Stage of Linux for Low-power ARM Embedded Devices. In: Proceedings of the Korean Institute of Information and Commucation Sciences Conference. The Korea Institute of Information and Commucation Engineering, 2013. p. 137-140.