Proceedings of IC-NIDC2012

BOOTING LINUX FASTER

Wonsik Lee, Youjip Won

Embedded Software Systems Laboratory,
Department of Electronics & computer Engineering, Hanyang University
matureelf@hanyang.ac.kr

Abstract

Booting is a process that initializes all the systems and it is an essential process of operating system to work. Because of this reason, system users have to wait until this process finishes and it is very inconvenient. Recently, non-volatile memories are rising up as an alternative method for DRAM. Their non-volatile characteristic makes several processes in existing boot mechanism unnecessary. If these processes are eliminated, the system users can spend shorter time to control the system. In this paper, we propose a technique which uses only one kernel image for every booting by using non-volatile memories.

Keywords: Fast booting, Non-volatile memory

1 Introduction

Linux's boot process is completed in the order of copying a binary kernel image which is saved in a secondary storage to a ramdisk and then loading it to a main memory, and finally executing the kernel image. As non-volatile memories (MRAM, FRAM and so on) are developed, new software technologies of operating system are needed.

Current boot mechanism is based on the assumption of a kernel boot image is exist in a block-based storage. If memory elements can save their values without power supply, booting process of operating system can be much more effective. In other words, non-volatile characteristic makes booting faster by eliminating the loading process of the kernel image to the main memory, and decompressing process of the kernel image.

Shortening of boot time is a one of important topics in information appliances such as smartphone, smart TV and so on. In this paper, we will focus on reducing overheads caused via 'loading process of a kernel image to a main memory' and 'decompressing process of a kernel image' in boot time[1][2]. Ironically, key points of this paper is not about using the non-volatile characteristic of new memories but eliminating the non-volatile characteristic of kernel objects in every boots.

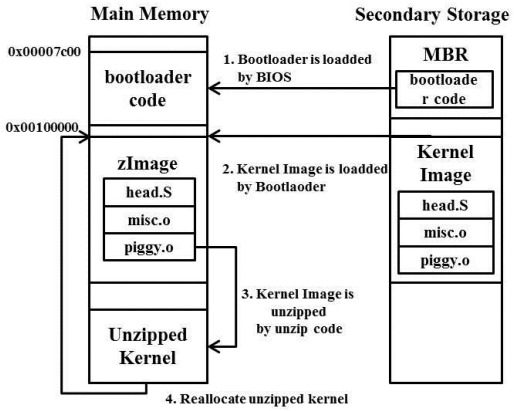

Figure 1. Booting mechanism

2 Background

2.1 Boot process of Linux

Booting is a process that makes CPUs, memories and other devices to initialize and operating system to run, so that the user can use the stabilized system.

Generally, a ROM BIOS stage is the process that CPUs and memories to initialize in computing environment. This process is a first step of the boot process and first codes that carry out when the power goes into computer; however, we will not deal with it since we are focusing on initialization of software that stays in a main memory.

There is a boot loader seeking code that approaches Master Boot Record (MBR) of each disks, exist at the end of the ROM BIOS. The MBR is formed with the boot loader code, a partition table and a magic number. The ROM BIOS finds a main partition refer to the partition table and finds existence of a boot loader according to the magic number in the MBR of this partition.

If a boot loader exists, that boot loader will be loaded to the main memory and since loaded, the

boot loader has a control of system. The boot loader loads a kernel image from a disk to a main memory.

Commonly used kernel image form is ‘bzImage’. ‘bzImage’ is arranged by this order head.S code which activates virtual address by MMU, ‘misc.o’ code which decompresses kernel and ‘piggy.o’ that is the compressed kernel.

Boot Loader activates a MMU by jumping to starting part of head.S after loading the kernel image to arranged starting address. After this code has been activated, CPU can’t reach to physical address of the main memory and only through the virtual address, they can access. Compression of the piggy.o will be decompressed by calling kernel decompressing code and when its compression is decompressed, a kernel binary image exists within the memory; however, that does not mean the kernel is carrying out immediately.

Since the compressed kernel image in the main memory is no longer used, it will be replaced by the decompressed kernel to prevent wasting of memory space. After that, the boot process enters to start_kernel() function also known as starting function of the kernel, with playing of initialization code of each kernel components, calls init() function to kernlel thread from rest_init() function and then goes into idle state with infinite loop.

Even though number 0 process enters idle state, the booting is still not finished. ‘init()’ function that was called to a kernel thread becomes number 1 process and also calls daemons and ‘initcall()’ functions. When all the process is completed, finally it finishes booting through a shell program and waiting for users to input.

2.1 Kernel object initializing

Initializing methods on memory can be changed depending on what kinds of objects are used by kernel. If the loading process and the decompressing process of the kernel image are removed, as the goal of this report, values that are saved in the kernel objects will be influenced by object initializing methods.

From the programming point of view, when same images are used for every booting, every object must have same value regardless of booting number. Therefore, the chance of influencing on the kernel objects should be eliminated in the boot process.

Local objects are initialized at the beginning of kernel’s run. After kernel compiling, assembly codes are made to set the local objects to be certain initialized value in the code section. Since the code section is unchangeable during kernel’s starting progress, it will always have same value unless there are any external effects. Which makes its effect on booting is small and it does not cause problem; however, global objects might cause some problems according to initializing method.

A memory area which kernel global objects exist is divided into two sections; a Block Stated Symbol (BSS: uninitialized data section) section and an initialized data section. Among them, the global objects in initialized data section are causing troubles.

BSS section is a section with existence of uninitialized global objects and each objects in this section doesn’t make any initializing code for a smaller kernel image when compiling. The section has only section start and end address at the section header and after the kernel image is loaded, initialization codes in ‘head.S’ make all memory section zero in a lump by using section start and end address.

The data section is section with existence of initialized global objects and it is initialized with kernel run. There are no problems at initialization time because this section is initialized same time as the logical object; however, because they have initial values, their initialization codes are built to reference addresses of data section when they were compiled. If the loading process of kernel is eliminated, the objects in the data section have different values in accordance with booting numbers; thus, it influences a normal booting.

3 Kernel image reusing technique

If the loading process of kernel image and decompressing process is removed, it can help to reduce the time for boot; therefore, a previously decompressed kernel image is needed and the decompressing codes of the kernel image have to be blocked, too.

In existing kernel booting mechanism, there is a process that deletes objects which was used in the boot time only. However, they are needed for the other bootings if only one kernel image is reused; thus, this process must be eliminated.

To complete booting stably regardless of the booting number, the kernel global objects should maintain same initial value in every booting. For this reason, individual global objects should be initialized properly in every booting.

3.1 Eliminating decompression process

The decompressing code of a kernel image is placed at 'misc.o’ file in existing kernel image. These codes are only used in boot time; thus, they are deleted after decompression of kernel image by kernel relocation codes.

To avoid the decompressing process of kernel and to reuse the kernel image, these codes should be bypassed and a previously decompressed kernel image is needed.

Fortunately, the building process of decompressed kernel image is existing in the kernel compiling process and the file name of decompressed kernel image is called 'Image'.

The existing boot mechanism have a process that deletes codes which was used in boot time only, but as long as it exist, it is impossible to boot repeatedly through one kernel image. 'free_initmem()' function deletes object which is used in boot time only; thus, this function should be bypassed.

Similarly, a decompressing process of a ramdisk and a deleting process of a ramdisk image must be removed. We can use the uncompressed ramdisk image without any changes, so if deleting process of the ramdisk image is removed, basic condition for reusing the ramdisk image is complete. 'free_initrd_mem()' function is similar to 'free_initmem()' except it deletes the ramdisk image after ramdisk image is decompressed; thus this function have to be bypassed, too.

3.2 Global object initialization

If a Linux kernel is operated on a non-volatile memory, all kernel objects have non-volatile characteristic; thus, they maintain previous value unless they are initialized explicitly. This characteristic occurs critical errors in the existing boot mechanism. A memory area which kernel objects were saved is divided into three areas: stack area, heap area and data area [1], among them, the most influencing area to boot is the data area.

The Stack area and the heap area are used by logical object. These areas are dynamically allocated area and their metadata are updated immediately; therefore, they are not influenced by non-volatile characteristic.

Among the data area, the BSS section won't be influenced by the characteristic; however, global objects which have initial values are making errors because other bootings are continued with their previous value after first booting.

The initial values of kernel global objects are different from each other, so it is impossible to initialize in a lump; thus, over works and times are needed to initialize the data section. In this paper, for shortening kernel code modifications, we use several kinds of techniques.

First, we made an initialization list via extracting all of kernel global object's name, address and size in the kernel symbol table.

Second, we listed the global objects that have changed value via comparing memory addressed of decompressed kernel image which was previously booted with the original image.

Third, we initialized the global objects in the list by using 'memset()' function because many global structures are initialized only few member variables when they were declared, it is too much works to initialize all member variables.

5 Conclusions

There is a chance to replace the DRAM to non-volatile memory as a main memory. Under this circumstance, the decompressing process and the loading process of kernel image are obviously overheads; therefore, by removing these processes, we can gain reduced boot time. There is no need to save the original image at storages like flash memory; therefore, it gives a good chance to making better use of the information appliance's small storage space. The problem of maintaining initial value which we found during the improvement of booting speed is another important issue, too. If non-volatile memories are being used, maintaining of global objects will be an important issue in not only booting but also in application level. We believe that the method which we proposed will be a good solution to these problems.

Acknowledgements

NRL: This work is sponsored by KOSEF through National Research Lab at Hanyang University (R0A-2007-000-20114-0).

References

[1] D.P.Bovet, M. Cesati, Understang Linux Kernel 3rd Edition, O'Reilly Press, 2006.

[2] Chung, K.H, Choi, M.S, Ahn, K.S, "A study on the packaging for fast boot-up time in the embedded linux", RTCSA 2007. 13th IEEE International Conference on Embedded and Real-Time computing Systems and Applications, 2007.

[3] D Fuji, T Yamakami, K Ishiguro, "A Fast-Boot Method for Embedded Mobile Linux: Toward a Single-Digit User Sensed Boot Time for Full-Featured Commercial Phones", IEEE Workshops of International Conference on Advanced Information Networking and Applications, 2011.

[4] K. Baik, S. Kim, S. Woo, J. Choi, "Boosting up Embedded Linux device: experience on Lunux-based Smartphone", In proceedings of the Linux Symposium, 2010.

[5] Uncompress Kernel, http://elinux.org/ uncom pressed_kernel

[6] Hiroki Kaminaga, "Improving Linux Startup Time Using Software Resume", In proceedings of the Linux Symposium, 2006.

[7] T. Benavides, J. Treon, J. Hulbert, W. Chang, "The enabling of an Execute-In –Place Architecture to Reduce the Embedded System Memory Footprint and Boot Time", Journal of computers, 2008.