The Research of Fastboot of Embedded Linux based on State Keeping and Resuming

Peng Chen, Quansheng Zhang
Software Engineering Center Co., Ltd. Chinese Academy of Sciences Beijing,China
(pchen,xfdong)@sec.ac.cn

Shuzhi Wang, Xiaofeng Dong
Software Engineering Center Co., Ltd. Chinese Academy of Sciences Beijing,China
(szwang,qszhang)@sec.ac.cn

Abstract—**To improve the startup speed of the embedded devices has a very important significance for application and development of embedded Linux systems. The paper analysed the startup process of embedded Linux systems, based on the method of accelerate way of modern system, proposed a new way to accelerate the startup speed of embedded Linux system, and applied this method to the specific embedded board to verification, achieved significant result in the end.**

Keywords*—*Fastboot; Embedded Linux; State Resume; Image.

Project support: Important National Specific Projects(2009ZX01039-003-002)

Ⅰ Introduction

Embedded operation system is non-PC system, But it has the function of computer. It is the dedicated computer system based on application-centric, software and hardware can be cut , for functionality, reliability, cost, size, power consumption etc have strict demanding [1]. In recent years, Linux, with its excellent features ,widely used in embedded System. However, as the operation system that refer to PC design, the designer did not consider the startup speed in the beginning, leading to the boot time is generally more than tens of seconds with the typical way , the situation is unacceptable to users. In addition, with the increasing function in Linux kernel can also lead to boot time more longer[2]. Therefore, improving the boot time is the key problems for the embedded Linux operation system. At present, accelerating the boot speed of embedded Linux mainly in the following ways[3].

Firstly, in the Firmware and the Bootloader stage, at this stage, we can choose reset to bypass the Firmware and Bootloader, this is means to allowing the running kernel to loading and running another kernel. The typical implementations is Kexec, it has two components, kexectools which exist in user space and kernel patches. Another method is to add reboot = soft number in the kernel command line, this way can also skip the Firmware, but the disadvantage is it is not called from the user space.

Secondly, the kernel stage, at this stage, we can improve the startup speed to using some system optimization approach to removing the functions that auxiliary detect the system performance such as RealTime Clock or calibrate_delay. But the effect is not obviously in this way.

Thirdly, in the user space stage, Many kinds of Linux systems (such as consumer electronics) need to boot the graphical interface and other services, the system that non-optimized will default to boot the process that will never use or not use currently, this part will be spent much time. The easiest optimize way is to rewrite the service configuration file according to the actual needs. In addition, init script execution is serial, lots of scripts will lead to the boot process very time-consuming, So the parallel running can be considered to speed up the boot speed. However, these methods only for the special areas.

In addition, Pre-reading, pre_link , XIP and file system optimization technology in the embedded applications can also improve system boot time at present, but these technologies are dependent on the hardware platform, and some way mutually exclusive, so they will be considered according to the actual needs.

Ⅱ The normal boot process of embedded Linux

The boot process of Linux as follows: after open the power , the system power-on and self-test, as followed, the system will run BootLoader, BootLoader is a small program which ran before the operating system kernel running. It initializes hardware components, establish mapping diagram of the memory space in order to bringing the system's hardware and software to a suitable environment state, and next, It read the memory image according to path of the kernel image, and extracting it, and then the system will place the decompressed kernel in memory, and calling start_kernel () to start a series of initialization function and initializes various devices to complete the establishment of the Linux kernel environment. After the kernel is loaded, / sbin / init is running which reads / etc / inittab file, the file is used to initialization and set to run level, and then, Linux system will execute a script to finish a lot of work, including setting PATH, setting the network configuration, starting the swap partition, setting the / proc and so on. After do those ,The system will be loading the kernel module according to a file under the etc directory . and then run the script based on the levels of script program to complete the initialization and start the appropriate service. Finally, It execute / etc / rc.d / rc.local script, which is the place Linux users can set their own personality . And next ,perform the login process to complete the Linux boot. The total flow chat is shown in Figure 1.

978-1-4244-8165-1/11/$26.00 ©2011 IEEE

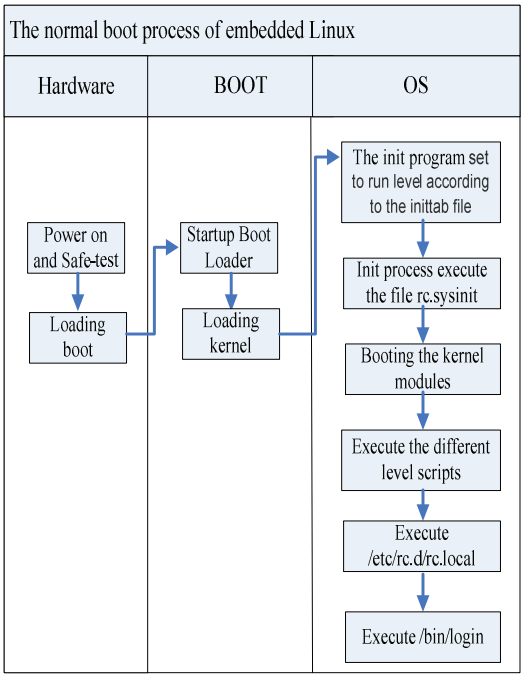

Figure1 The normal boot process of Embedded Linux

Ⅲ The fastboot strategy of embedded Linux operating system based on the state keeping&resuming

Nowadays, many embedded systems are used the hibernation&weak technology to improving the boot time. In the latest kernel Linux-2.6 support hibernation and standby mechanism, the realized way in PC system have suspend2, swsusp etc, those technology save the memory information both in the swap partition and in the swap file [4]. Specific execution of these have two ways, the kernel mode and software mode, but all of these take the sleep process divided into two stages, The first one is SUSPEND stage, the stage save the state of current OS to the nonvolatile storage device and turn off the system. the second one is RESUME stage, the system reboot and resume the saved state[5]. We will integrate these technologies that talked above, propose a new method, and applied to the embedded board, realized the fastboot of embedded Linux operating system.

The Figure 2 shows the realized process of state keeping&resuming that described in the paper.

1 Frozen system activity

The first stage of state keeping&resuming is stopping all other activities. All of the processes are divided into four groups: the processes that marked PF_NOFREEZE, the processes that not marked PF_NOFREEZE, the user processes that marked PF_SYNCTHREAD. and the remanent, the freezing order as follows:

- The processes Not marked PF_SYNCTHREAD or the user processes that marked PF_NOFREEZE;
- The user process that Marked PF_SYNCTHREAD;
- The kernel processes Not marked PF_NOFREEZE.

2 Memory "occupation"

In order to take the state keeping&resuming program running successfully, we need enough disk space to store the image, and need enough memory space to satisfy recovery algorithm and other constraints. For the purpose of obtain the parameters that system required, the keeping&resuming programs will "take up" memory. After freezing the process, if the memory did not meet the parameters that system specified, The keeping&resuming programs will frozen all other processes , and then began to "take up" the memory, until satisfied the requirements that system specified. Then freeze the processes again and re-examine the calculations.

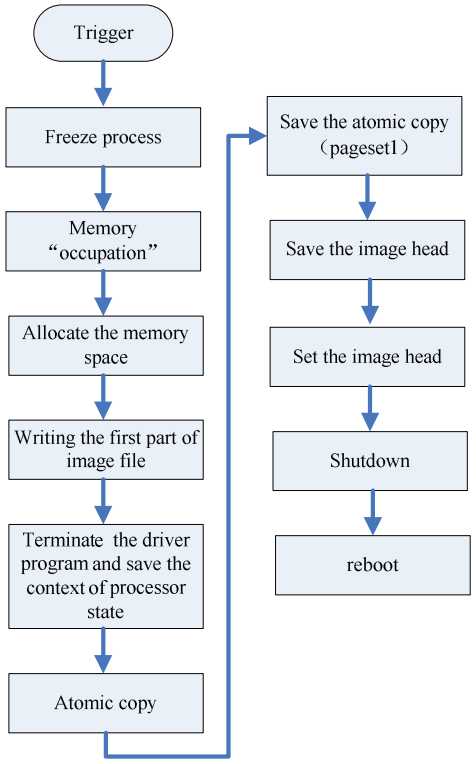

Figure2 The realized process of embedded Linux system based on state keeping&resuming

3 Allocation the storage space

In this stage ,the state keeping&resuming programs allocate the storage space for the image file. The keeping&resuming programs do not know how many pages or which pages need to save, so we have to apply enough storage space to saving the image file. If the application fails, we will "take up" more memory, and then try it again. If the application is successful, and then try to allocate additional storage space, the purpose of doing it just to prevent the compression ratio less than our expectations.

4 Writing the first part of the image

The state keeping&resuming programs divide the image file into two parts, Pageset1 and Pageset2. Pageset2 make up of active and inactive pages and essential cached pages. Pageset1 is composed by all other pages. The reason that we divided in this is that we want to copy the kernel and at the same time ensuring the consistency of image file and kernel. As pageset1 not save the pageset2 page, so we save pageset2 page firstly. And then, we do atomic copy using pageset2 pages and other free pages. When we save these two kinds pages, we use block I / O operations, and so it will not change the contents of the cache page.

5 Terminate the driver program and save the context of processor state

After finished pageset2 , the keeping&resuming programs will call the power management function and inform the driver to save the state, saving the processor state as preparation, at this time we can do atomic copy to the memory.

6 Atomic copy

In this step, in addition to maintaining the resume codes, all other things have stopped. When we are going to atomic copy, the processor is at the state of idle or in a frozen, the driver stopped and stored the configuration information in memory. In our low-end architecture specific code, we save the CPU state. Therefore, we can do atomic copy before the drivers and other procedures resuming.

7 Save the atomic copy (pageset1)

The state keeping&resuming programs in the stage is written the remaining atomic copy. Because we had to copy these pages to other places before do it, we continue to use the standard block I / O operation to write, and we also do not worry about our image will be destroyed.

8 Save the image header

"Image head," including other parameters that we need. we tell the allocator allocated some continuous space, so we will be able to re-read the image file when the system restart.

9 Set the image head

Finally, we will set the image head in the "resume = location". Distributor will change the signature of the image head, to illustrate the existence of the image, if necessary, we will also take it to point the beginning data.

10 Shutdown

11 Reboot

When reboot, we firstly copy pageset1 to the original location of memory, and then resume the context of processor. At this time, the original kernel is running. Next, the system reload the pageset2 pages release the memory depositing the keeping&resuming programs, resume the pageset header, reboot the processor.

Ⅳ Experimental Analysis

In order to ensure the practicality and reliability of the method that we talked above. we applied it to digital TV of Haier, and analysed the results in detail.

Experimental environment

Hardware environment: MSTAR 6198 TV board

1) Frequency: 700MHZ
2) Memory: 512MB
3) Flafh: 256MB
4) Serial ports:jtag.

Software Environment

1 TV board adopt Linux-2.6.28.9 Embedded operating system.
2 File System: UBIfs for root file system

Experimental procedure

The figure 3 shows the general experimental follow chat of the paper:

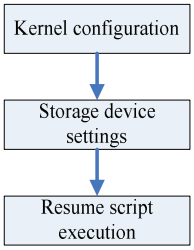

Figure 3 The total follow chat of the method based on state keeping&resuming

1 Kernel configuration

In order to make the system realize the memory state saving, we need patch the patch which can realized the state keeping&resuming according to the Linux kernel version. After do this ,we setup yes for the options ”Support for paging of anonymous memory(swap) “. As is shown in Figure 4.

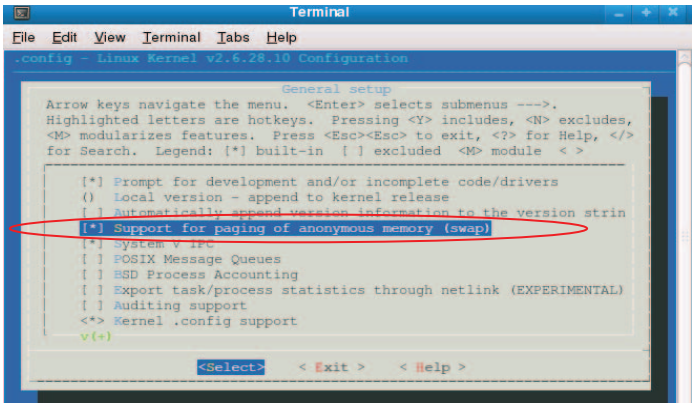

Figure 4 Adding the state keeping function

And then we allocate the new partition（dev/mtdblock2）

in the flash for storing the memory information. As is shown in Figure 5.

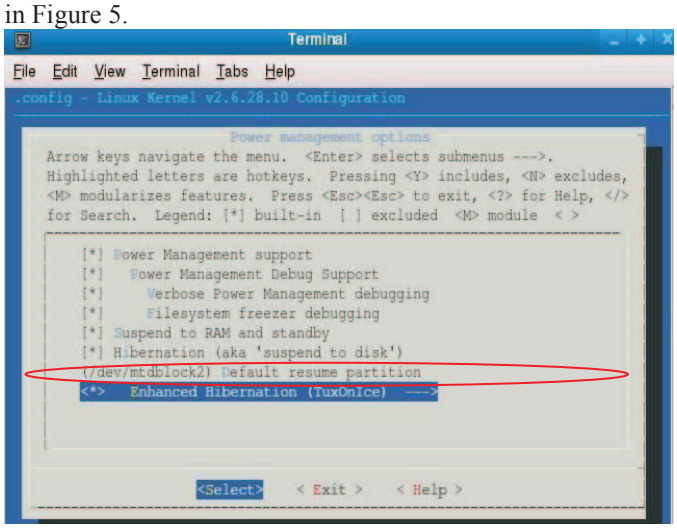

Figure 5 The setting of the location of state information

Because we require the system reboot every time form the previously stored information ,we will set No for the option of "Wait for initrd/ramfs to run, by default ". As is shown in figure6.

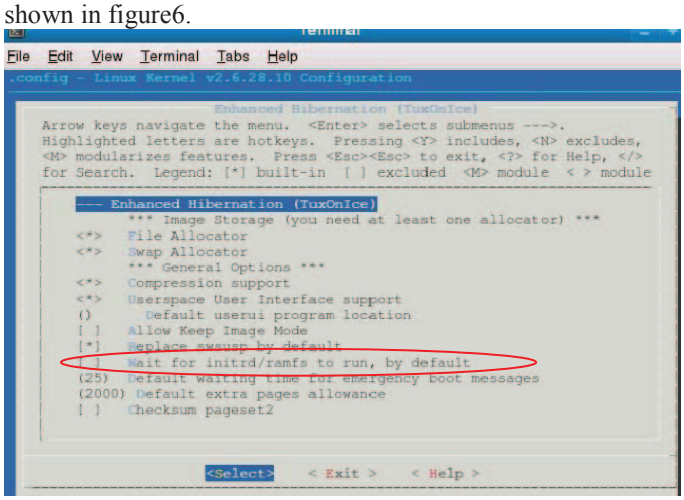

Figure 6 Setting the start configuration

2 Storage Device

The storage device that the method used is a partition which was allocated from the Nandflash of the board.

3 Execute fastboot script

The script is used to execute the state keeping program, Which include creating storage partitions and executing the state keeping&resuming procedures

Experimental results

Though the experiment, we get the boot time of system. before modified and after modified .As is shown in table 1,figure 7.

Table1 The contrast of system boot time before and after modified

	first/s	second/s	third/s	fourth/s	fifth/s	Average vaule/s
normal	14.57	14.16	14.55	14.55	14.53	14.47
modified	12.06	12.00	12.03	11.99	11.79	12.00

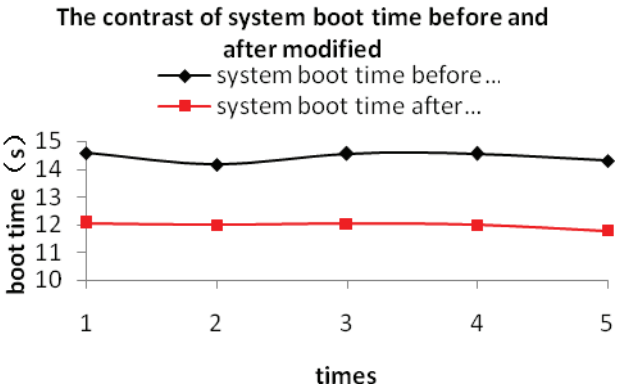

Figure 7 The contrast of system boot time before and after modified

As is shown above the table, we can see that the time of system boot after modified is reduced about 20% than before modified.

V Conclusion and Outlook

The paper analyzed the boot process embedded Linux, proposed a new method of embedded system based on state keeping&resuming combined with the resolvent of fastboot based on hibernation. though the experimental verification, the effect is obvious for accelerating the boot speed of Embedded Linux, It has a very important signification for the research of the embedded Linux startup process optimization and shorten the startup time of embedded devices.

References

[1] Chi Xu Embedded linux scientific and technological tide，2008(2):27-28.

[2] Jun Xu, Ya-qing Tu Analysis and Application of Accelerating the Boot Process for An Embedded Linux System, Journal of Logistical Engineering University, 2005(3):54-58.

[3] Hong-xiang Duan, Di-huan Sun, Wei-ning Liu, Wei Song Fast booting method based on kernel booting improvement in embedded Linux, Computer Engineering and Design, 2009,30(1):16-18.

[4] Kunhoon Baik,Saena Kim, Suchang Woo,Jinhee Choi.Boosting up Embedded Linux device:experience on Linux-based Smartphone[J].Proceedings of the Linux Symposium,2010.6.

[5] Xue-qiao Li, Ben-fu Xu, Xiao-ai Jia. Embedded Linux Rapid Boot-up Based on XIP and Hibernation, Electronics R&D. 2010(47):11-12.