Improving the Boot Time of the Android OS
Xia Yang, University of Electronic Science and Technology of China, China
Nan Sang, University of Electronic Science and Technology of China, China
Jim Alves-Foss, University of Idaho, USA

Abstract — Increasing application requirements in embedded systems demand additional software initialization and configuration during startup, which adversely affects system boot time. The demand for fast startup is motivated by embedded systems, not only for consumer electronics such as digital TV and mobile phones but also for devices in automotive, medical and other applications. This paper presents novel approaches to reduce the startup time of complex embedded systems, using Android OS as an example. The startup latency of Android is analyzed in order to find ways to reduce the boot time. Then to reduce startup time, we propose a better U-boot Fast-Boot based on Suspend-Resume techniques, improve the read speed of SD cards, and try to reduce the size of suspend images. Our measurements show that these methods can reduce the total startup time by 83%.

Index Terms — Android, boot time, fast-boot, U-boot, suspend-resume

Advances in computing technology, enhanced communication bandwidth, and innovative software have brought increased functionality and capabilities to consumer electronics. Television sets now have embedded microprocessors and operating systems that allow internet access, streaming media and a wide variety of multimedia applications. Tablet PCs and smart phones come loaded with a wide variety of apps.

Unfortunately, the addition of new features; support for multiple services and data formats; complex and sometimes proprietary multimedia libraries; and increased hardware complexity has driven an increase in the complexity of the software that controls these embedded systems. In the early days of TV, we had to wait for vacuum tubes to warm up, before we could use the TV, but advances in electronics brought us to the day where devices could be turned on almost instantly, and you could readily switch between channels and quickly view the screen. As we moved to digital television, and especially smart televisions, consumers lost that access speed. One of the delays seen by consumers is the delay in starting the device from an off state.

The demand for fast startup is mostly motivated by embedded systems, because it is one of the most important factors affecting the decisions of buyers. A long waiting time will alienate users, especially, the users of consumer devices such as digital TV. We address that delay in this paper, and suggest ways that system designers can improve device start time.

Size, power and price constraints of embedded systems make software design and implementation uniquely difficult for engineers. Traditionally, the demand for embedded software was met by only a few proprietary platforms. In 2008, an open source software platform primarily for smart mobile phones, Android, was launched by the Open Handset alliance led by Google. Android has found great success since it entered the mobile terminal market. It is a complete embedded software stack, comprising of a modified Linux-based kernel, middleware, application framework and applications. As an open platform with free and convenient tools offered to application developers, Android Operating System is now also being used in digital TVs and tablet pcs and other devices. Android’s complicated software stack architecture can yield great returns for hardware and supply rich software services. However, in order to support converging applications, Android comes packed with features, resulting in considerably large code size. These advantages also delay the boot-up time of android devices up to an average of 30 seconds, therefore it is a great platform for our research.

This paper reports the results of our investigations into improving the boot time of Android devices. Based on our measurements and analysis of the Android boot procedure, we found that the most of time during the boot process is spent on the Android service and application. Using these results, we developed a new approach for booting which improves the startup time. This approach is based on the Suspend-Resume technique, but it is very different from that. When a user powers off a device it first suspends a running system and makes a system image including copies of CPU registers, memory and device state that is then stored on disk or Flash memory, after that the system will shut down all power to the device, not just enter a sleep mode. When the user powers on the device, it reloads the stored image from external disk and resumes the system with the suspended image instead of cycling through a full boot procedure. This technique is very useful for non-battery-powered devices and frequently rebooted devices such as Digital TV. It also can save battery for battery-powered devices like mobile phones and...
tablet pcs, because after saving the system image, all power can be shut down; the approach even works with rapid power loss.

In this paper, we analyze the Android boot procedure and the shortcoming of the Suspend-Resume technique and then present three approaches to improve the startup time of Android, beyond the use of the Suspend-Resume technique. This work provides the following contributions:

U-boot fast-boot technique: Through our experiments, we calculated the time taken in each boot step for the Suspend-Resume technique on Android. We found that much time is spent on the initialization of the boot-loader and Linux kernel. This paper presents the U-boot fast-boot technique, which can skip the startup of Linux kernel and also reduce the U-boot initialization time, resulting in shorter startup time.

Enhance the read speed of SD card: In order to reduce the time of reading an image, we enhanced the read speed of the SD card by modifying the I/O operations of SD card in U-boot when it reads the image from SD card.

Shrink image size: Since our system resumes with a suspended image, the size of image impacts the time it takes to boot; the larger the image, the slower the boot time. We’ve found a way to shrink the image to very small size.

I. BACKGROUND AND RELATED WORK

This section describes the analysis of Android boot procedure and related work for improving the startup time of embedded operating systems such as Linux, Android, etc.

A. The analysis of Android boot procedure

Booting of an Android OS starts at power-on and ends when the Android “home screen” is displayed. We have analyzed the boot sequence of Android, which includes four steps:

Step 1: Power on and Boot ROM: When we power the system the Boot ROM code starts execution, which loads the Bootloader from ROM into RAM and starts execution.

Step 2: Boot Loader: The Bootloader is the first program to run for initialization hardware such as board, processor and memory and etc. We use U-boot as the Boot Loader in this paper.

Step 3: Linux Kernel: Android kernel starts very similar to normal Linux system boot.

Step 4: Android userspace initialization (AUI): Gaurav calls this last step Android userspace initialization, which begins with execution of Android’s init process, and ends when user commands to the system can be accepted after display of “home screen”. This step starts the remaining processes such as Init process, Zygote, Dalvik VM and the system service and the “home screen” process.

We obtained the boot time of each of these steps (Figure 1). Most of the boot time is spent in Step 4. This is because this step includes making the Zygote process, preloading the Dalvik virtual machine, running Android service, scanning of application and system packages, etc. We call the time spent after the Boot Loader and loading of the Linux kernel “AUI’s time”. The sum total of the startup time is normally greater than 30 seconds, which is too long for most users.

We propose to change the boot procedure and focus our improvement effort on the optimization of the Suspend-Resume technique, which can reach the home screen in 5 seconds.

B. Related work

Some previous research projects have looked at improving the startup time of embedded systems. Bird proposed several techniques to enhance the boot time of a commodity OS. He analyzed each step of the boot sequence and suggested various methods such as kernel execution-in-place (XIP), probe delay elimination, and RTC read synchronization. Wool also proposed methods to optimize the boot time for an embedded OS. His approach not only looks at time reduction needed to initialize the core kernel and drivers, but also addresses application-level considerations and other areas of the kernel such as linear flash memory access and time optimized module insertion. Park et al. applied well known boot time reduction techniques to a commercial product. Chung et al.
analyzed the detailed boot time consumption for each boot step and also compared the performance of several file systems for fast boot for a digital camera.

Kaminaga proposed a snapshot boot mechanism to enhance the startup time of a commodity OS by using the Suspend-Resume technique. This method takes a snapshot of a running system image and resumes the system using the suspended image instead of going through a full normal boot. The snapshot image is created only once, stored on disk or flash memory, and the same image is used repeatedly, every time the system is powered on. Although this is similar to our technique, Kaminaga’s snapshot is only useful for read-only file systems since the snapshot image is permanent; which makes this difficult to deploy in a commercial product. Heeseung has improved the startup time of digital TV, also using a Suspend-Resume technique. These proposed approaches can reduce the startup time of digital TV by about 50%.

Gaurav presented a “system level” optimization method for embedded software to achieve faster boot time for Android. The total boot time of the optimized setup is ~10.1 seconds. The other closest idea is Pathpartner’s Android fast boot implementation which takes ~18 seconds for boot. Although these two methods have optimized the boot time of the Android OS effectively, the boot time is still more than 10 seconds.

C. Analyze the shortcoming of Suspend - Resume technique

The Suspend-Resume technique has two parts: suspend and resume. The suspend procedure consists of five steps: 1) Freeze all processes, 2) Suspend and turn off devices, 3) Save the processor state, 4) Store the Image to the disk, and 5) Power down. The resume procedure also consists of five steps: 1) Boot Loader, 2) Startup the kernel, 3) Read the Image from disk, 4) Restore process state, and 5) Resume device. There are three shortcomings in Suspend-Resume technique, which we address in the following three sections.

1. The kernel is responsible for reading the image from disk in the Suspend-Resume technique, and the kernel must still be started after Boot Loader startup; which takes the approximate 2~4s (Figure 1).
1.2. Another bottleneck is due to the fact that reading the system image from disk will require time, depending on read speed of external memory, which is about 10Mb/s for a typical SD card. The method used to read the stored image is important.

1.3. The last problem is that Suspend-Resume has a large image size. If the system image is very large, the resume time of system will be very long.

II. U-BOOT FAST-BOOT APPROACH

This section describes U-boot Fast-Boot technique, a new method to improve Android startup time, based on Suspend-Resume. We use U-boot as a Boot Loader tool, which reads the system image instead of executing the kernel when the system resumes.

Similar to the Suspend-Resume technique, the U-boot Fast-Boot technique also includes suspend and resume processes. The difference between the Suspend-Resume and the U-boot Fast-Boot techniques is that in Fast-Boot the image is read directly after board initialization rather than after kernel initialization. Therefore, this method saves the time normally needed to startup the kernel and hence speeds up the startup of Android OS.

A. Design the data format for image

In order to resume the system state, the current state of the system must be saved. This involves not only saving the memory registers and CPU state, but also all resident memory pages. To improve the save process, we examine the current kernel memory map, looking for inactive memory pages that are not currently in use by the system and therefore do not need to be saved.

In our initial attempt to resume the image in the U-boot approach the Memory Management Unit did not yet work, because the kernel wasn't initialized. Therefore we could not utilize the memory map to assist us in restoring the memory pages. To solve this problem, we redesigned the data format of the saved image (Figure 2).

Memory pages must be saved to disk according to this structure. The addr_bak data structure is used to map the physical memory pages from their current location, to their location on the storage device. This allows us to have a mapping between the saved images and the current state of the system. Some of these saved pages will include the current system's memory map for the MMU, and therefore after being reloaded, the system will have the correct memory map.

B. Write/Read image

Using the new data structure for system images we can write/read the image to/from the disk. First, we save all memory pages to the disk, page by page, based on the page_num field and the src_address of each page. As we store the source addressed to disk, we record the physical address of the pages on the disk to the dst_address field. Finally, we set the field fastboot_sig value to “success” if the process is successful; otherwise, set the value to “failed”, and store the fastboot data structure to disk.

```c
static struct fastboot_struct {
    struct addr_bak * addr_table; /* The physical address of pages that need to be saved*/
    u32 page_num;                  /*The number of pages to be saved*/
    char  fastboot_sig [10];      /* The fast boot result signature */
    u32 pc_addr;                   /* Current value of PC register */
} /* Image data structure of U-boot fast-boot*/;

typedef struct addr_bak {
    unsigned long src_address;  /* Source physical address of pages */
    unsigned long dst_address; /* Dest. physical address of pages */
} /* Structure for saved the address*/;
```

Figure 2: The image data structure
The process of reading the image from disk is the reverse of the process of writing to the disk. First, we retrieve the fastboot data structure and get the initial address of the image on disk. Then, we start reading the image, page by page, from the disk to the original memory address as specified in the data structure.

C. The startup step of the U-boot Fast-Boot technique

The startup step of the U-boot Fast-Boot approach is different from, and simpler than, the standard Suspend-Resume method and the normal startup of the kernel (Figure 3). In this method, U-boot is responsible for copying the image to external memory, not the kernel, which can reduce the startup time by skipping the startup of the Linux kernel. Moreover, this method also reduces the boot time of U-boot itself because the initialization procedure is simpler than that of the Suspend-Resume method.

If the image size is very large, the time to read the image will be long, and the startup time of the system will also be long. Therefore, reducing the image size and enhancing the average read speed of external memory are two other methods for further improving the startup time.

### III. ENHANCE THE READ SPEED OF SD CARD

We chose an SD card for external memory, due to its popularity in embedded systems as additional external storage. This section describes how we reduced the average transfer rate of the image.

Normally, U-boot reads the image page by page from the SD card, at a data transfer rate of about 15M/s on SD card Class 4. Each page is stored to the appropriate page in RAM. The idea we employ is rather simple, we modify the I/O operations in U-boot and increase the amount of data transferred per transfer request, removing the overhead and additional associated latency. We do this by reading more than one memory page at a time when U-boot reads image from SD card. This modification is internal to U-boot and will not impact the performance of applications that use the SD card after fastboot. If consecutive RAM pages are stored consecutively on the SD card, we can transfer both pages to RAM without having to change the target destination address and restart a transfer. This enables us to take advantage of some of the hardware data pre-fetching and reduces the handshaking overhead with the SD card.

Through experimentation we found that this method can improve the read speed from 15M/s to about 21M/s, reducing the read time of the image by almost 30% and consequently improves the startup time of the kernel.

---

**Figure 3: The startup steps of U-boot fast-boot technique**

- Power On
- Startup basis devices by U-boot
- Copy the image to src address by U-boot
- Jump to kernel-resume-point
- Devices resumed by kernel
- Thaw the process by kernel
IV. SHRINKING THE IMAGE SIZE

Even with increased data transfer rates, it is still important to reduce the size of the suspend image as much as possible. We use the following system management command to shrink the image to the minimal size possible:

```
echo 0 > /sys/power/0
```

This method exchanges inactive pages of memory to the swap partition of the disk using the internal LRU algorithm. As mentioned earlier, inactive pages are not saved as part of the system image. We then use following commands to suspend system and make the suspend image:

```
# mkswap /dev/block/mmcblk0p1
# swapon /dev/block/mmcblk0p1
# echo disk > /sys/power/state
```

This method is very effective. Through experimentation we found that it will reduce the image size by about 50%-70%. It takes some time to ensure that all pages are saved to the swap space as well as then saved to the SD disk. However, consumers will usually shut down their device and walk away, not waiting for the shutdown to be complete. Therefore we have shifted some of the system restore time to shut down time, by storing the inactive pages in the swap space.

However, this now means that when the system is restored, any applications that need their inactive pages will need to reload them from swap space. This will slow down the application’s first use by the consumer after system restart. However, the load time for the inactive pages for a single application will be substantially less than that for the full system; therefore the additional wait time should be less noticeable by the consumer, and therefore more acceptable. In addition, with some additional work, we could use a background process to restore the active pages from swap while the consumer is using the system; preloading the saved applications and speeding up subsequent use. We will research other approaches to reduce the image size and speed up the resume process.

V. EVALUATION

This section presents the startup time evaluation of the three methods described in this paper. For baseline comparison, we also evaluate normal booting and the Suspend-Resume technique on Android OS. We used the S5PV210 embedded development board as the testing hardware platform, which has an embedded CPU ARM Cortex-A8 1GHz and 1G memory. We also use an SD card as the disk to store the suspend image, with a read speed of about 15MB/s. The approach specified here is not restricted to a specific Android version; however we used Android 2.2 in our experiment.

A. Startup time test of Android normal booting

Figure 4(a) shows the normal startup time of Android, which includes the starting time of U-boot, Linux kernel and AUI. We found that startup time of AUI takes about 74% of the all startup time.

B. The startup time of Suspend-Resume technique for Android

Firstly, we suspend the running Android system and get a system image, which includes the CPU registers and memory state, approximately 84MB. To reboot the system, the boot process includes U-boot, kernel startup and reloading the system state. We conducted several tests on the system, calculating the average startup time for each of these components (Figure 4(b)). The average startup time of a Suspend-Resume system was 9.8 sec, with 1.3 sec for the kernel, and 5.5 sec to resume the system state (56.1% of the total startup time).
For comparison to the U-boot Fast-Boot technique, we used the same image as the Suspend-Resume method. The improved SD card resume technique gave us an improved read speed of 21M/s.

The time spent on resume is less than the time of Suspend-Resume method (Figure 4(c)). We also found that the time spent on booting U-boot is much less than on the Suspend-Resume method, because the U-boot initialization in our method is much simpler than in Suspend-Resume method. Our approach takes only 47% of the time of the Suspend-Resume technique and 17% of the time of a normal boot.

Since the SD reading technique is independent of our approach, we also looked at the performance if we do not enhance the read speed of SD card. If we assume the resume time is the same as measured in the Suspend-Resume technique, 5.5 sec, then the total startup time will be 5.8s, which is still 41% faster than the Suspend-Resume method. The measured time to resume system state is about 93.4% of total startup time.

C. The startup time after shorten image size for Android

Forcing the system to store inactive pages in swap space before we suspend the state, we reduced the size of the save image to 30MB from 84MB and further reduced the boot time (Figure 4(d)). Using this technique we are able to reduce boot time to 2.3 sec, or 9% of the standard system boot time.

D. Discussion

Our evaluation illustrates that our approaches provide for a greatly reduced startup time on Android. Saving inactive pages to swap space provides a great savings and could be combined with a traditional Suspend-Resume technique to improve boot time in many systems. Our U-boot Fast-Boot technique improves upon the traditional Suspend-Resume technique that it is based on, and could also be utilized by several systems. Table 1 summarizes the comparison between our method and others.

From our evaluation, it is apparent that our approaches are tightly associated with image size and read speed of external memory. The U-boot Fast Boot is advantageous in higher portability, as it can be used for digital TV, mobile phones and other embedded systems.
Table 1: The comparisons between our method and others

<table>
<thead>
<tr>
<th></th>
<th>Normal startup</th>
<th>Suspend-Resume</th>
<th>U-boot Fast-Boot &amp; improved read speed</th>
<th>Shorten image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot time</td>
<td>27s</td>
<td>9.8s</td>
<td>4.6s</td>
<td>2.3s</td>
</tr>
<tr>
<td>Image size</td>
<td>84 MB</td>
<td>84 MB</td>
<td>30 MB</td>
<td></td>
</tr>
</tbody>
</table>

VI. CONCLUSION AND FUTURE WORK

In the last few years, the popularity of the Android OS has increased. The Android OS is advantageous in that it provides convenient tools for application developers, allowing great diversity in applications for that use the system. The Android operating system is now being applied to digital TVs, tablet PCs, phones and other devices. However, the startup time of this diverse system is affecting consumer patience and acceptability of the systems.

In this paper, we proposed three approaches to improve the startup time of Android OS. The U-boot fast-boot technique is based on the Suspend-Resume method, the biggest difference being that our method can further reduce the startup time by skipping the startup process of Linux kernel. Moreover, this method also reduced the boot time of U-boot itself because the initialization procedure is simpler than that of the Suspend-Resume method. This method is 41% faster than Suspend-Resume. In order to reduce the time spent on reading the saved image, we enhanced the throughput of reading the image by modifying the SD card I/O operation. We have improved the read speed from 15MB/s to 21MB/s, resulting in a boot time that is 53% faster than Suspend-Resume, which can reduce normal startup time by 83%. In addition, we also shrink the image size by having the system save inactive pages in swap space, reducing about 35.7% of the image size and resulting in a boot time reduced by about 91% of normal. However, this last approach may slow down initial use of resident applications, since they still need to load the inactive page from the swap partition. But, the load time for the inactive pages for one application should be less noticeable by consumers, and therefore this method can be accepted and applied.

We also analyzed which points are bottlenecks to improve the startup time more. If the suspend operation is implemented every time there is a shutdown or power-off event, the shutdown will be very long, because the write speed of an SD card is much less than read speed. We believe we can achieve more improvement of the startup time for Android. We believe that the proposed mechanism and our analysis will be practical to enhance the startup time of a large embedded system such as DTV, mobile handset, and so forth.

REFERENCES


8
Xia Yang is an associate professor in the School of Computer Science & Engineering at the University of Electronic Science and Technology of China. Her research interests include embedded system, trusting embedded operating system, formal methods and software engineering. Yang received a PhD in Computer Science from the University of Electronic Science and Technology of China. Contact her at xyang@uestc.edu.cn.

Nan Sang is a professor in the School of Computer Science & Engineering at the University of Electronic Science and Technology of China. His research interests include embedded system, trusting embedded operating system. Sang received a M.S degree in Computer Science from the University of Electronic Science and Technology of China. Contact him at sn@uestc.edu.cn.

Jim Alves-Foss is a professor in the Department of Computer Science at University of Idaho, where he is the director of the University of Idaho's Center for Secure and Dependable Systems. His main research interests are in the design and analysis of secure distributed systems, with a focus on formal methods and software engineering. Alves-Foss received a PhD in Computer Science from the University of California, Davis (UCD). He is a senior member of the IEEE and ACM, Contact him at jimaf@uidaho.edu.