Measuring and Improving Application Launching Performance on Android Devices

Kyosuke Nagata, Yuta Nakamura, Shun Nomura
Electrical Engineering and Electronics
Kogakuin University Graduate School
Tokyo, Japan
{cm12021, cm13021, cm13024}@ns.kogakuin.ac.jp

Saneyasu Yamaguchi
Department of information and Communications Engineering
Kogakuin University
Tokyo, Japan
sane@cc.kogakuin.ac.jp

***Abstract*—For smartphone users, application launching time, which is period from time of application icon touch by a user to time at which the application starts interacting with the user, is one of the most important performances for comfortable uses. In this paper, we focus on Android devices and discuss methods for measuring application launching time and decreasing launching time. First, we introduce a method for measuring application launching time by inserting monitoring functions into Android operating system. Second, we present our measurement results of application launching times. Third, we focus on class loading process and discuss a ways for decreasing application launching time. It expands the list of preloaded classes by adding standard classes. Last, we present our evaluation results and demonstrate that the proposed method can improve launching time.**

Keywords—component; Android; Zygote; application launch

I. INTRODUCTION

Application launching time is one of the most important performances for smartphone users.

In this paper, we focus on application launching performance, with the goal of realizing their profound analysis and decreasing it. For these purposes, we will introduce our system that will enable detailed investigation of application launching procedures by modifications to the Android system, and will discuss a method for decreasing launching time with our analyzing system.

The remainder of this paper is organized as follows. Section 2 introduces Android with its unique application launching procedure. In section 3, application launching time is discussed, with explain of our analyzing system and analytical results. After the analyses, we propose a method for decreasing application launching time and evaluate the method. Section 5 introduces related works. Finally, we conclude our paper in Section 6.

II. ANDROID

A. Zygote

In order to decrease application launching overhead, new processes are forked using a special process called Zygote. Some important class files are read by most applications, so it is not effective that every application reads these class files every time individually. To prevent this, Zygote is equipped. Zygote is a system process with important class files loaded, and a new application process is forked by this process. Then, forked processes start with these files loaded.

This method is effective also for memory usage saving. Fork is performed with CoW (Copy-on-Write), thus a copy of these classes does not increase memory usage until write operation for these class files. In usual cases, no write operations are issued to all class files. Therefore, these class files are shared with all Application processes. This is more efficient way than a way with which each application process load class files and consumes memory individually.

B. Application Launch Procedure

An Android application is launched according to the following procedures. There are two procedures and one of them is applied based on the process state. The first procedure is forking a new process in the absence of the application process. The second procedure is resuming the existing application process by bringing the background process to foreground. We call the former "*forking*", and the latter "*resuming*" in this paper.

In the case of "*forking*", an application is launched according to the procedure in Fig. 1. (1) First, a user touches an icon of an application. Then, Home Applications sends Intent, which is a request to launch the application, to Activity Manager. (2) Second, Activity Manager sends a request of generating a process to Zygote. (3) Third, Zygote forks itself and generates a child process. (4) Fourth, a new process is initialized. (5) Fifth, *onCreate()*, *onStart()*, and *onResume()* in the lifecycle are performed.

In the case of "*resuming*", an application is launched as shown in Fig. 2. (1) First, a user touches an icon of an application. Then, Home Applications sends Intent to Activity Manager. (2) Second, Activity Manager sends a request of resuming a process to background application. (3) Third, Activity Manager calls *onRestart()*, *onStart()*, and *onResume()* in the lifecycle. In this case, any new process is not forked and initialized, and *onCreate()* is not called.

978-1-4799-2795-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CANDAR.2013.116

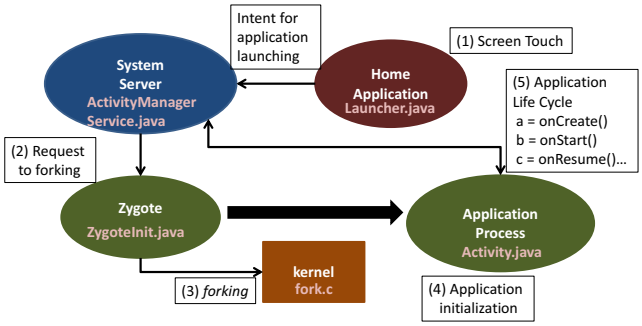

Fig. 1. Android Application Launching Procedure(*forking*)

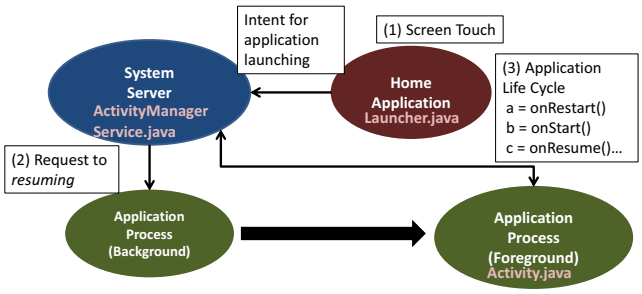

Fig. 2. Android Application Launching Procedure(*resuming*)

III. Application Launch Analyses

A. Application Launch Analyzing System

In this section, we explain our android application launch analyzing system. We introduced the monitoring functions into the implementations in which monitored events occur. The monitoring system records event type, time of the event, process ID, and other event information.

In the android kernel, the monitoring function is achieved by porting the implementation of the kernel monitoring system for Linux [1]. The monitoring system allocates memory for recording in advance, and then it stores event information in the allocated memory when events occur. The process required in run-time is only to store the data in memory. Thus, the runtime overhead is very small.

For monitoring events in userland, we inserted the monitoring code into the source code of Android application framework, Android runtime, and libraries. The code records event information into the Android logging system[2]. The stored information can be viewed by Android `logcat` command. The monitoring function is implemented in both Java and C language. The former is inserted into the application framework written in Java. The latter is applied to the Android runtime written in C language. In addition to these application launch monitoring functions, we have implemented a function for monitoring Dalvik VM's class load. This function will be mentioned in section IV.

B. Launching Time Evaluation

In this subsection, results of analysis using our system are introduced. We constructed analyzing systems using smartphones Nexus S (Android 2.3, Cortex A8 Processor 1GHz, 512MB RAM). Applications were launched on these system and the launching procedures are monitored.

In case of *forking*, we killed the existing processes of the target applications before launching it, and touched the screen to launch them. In case of *resuming*, an application had been forked and made a background process by tapping the home button beforehand. Then, screen was tapped to resume the application. In this paper, we define "start time of launching" as "time of screen touch", and defined "end time of launching" as "time of calling *onResume()*".

Fig. 3 depicts the total time to launch a browser application. The horizontal axis means the times of launching. From the first launch to the forth launch, the application was launched by *forking*. From the fifth to the eighth, it was launched by *resuming*.

Fig. 3 shows total time to launch an application. It implies that total time with "*resuming*" is much less than that of "*forking*". It also represents that the total time of the first "*forking*" and the first "*resuming*" are less than that of following "*forking*" and "*resuming*".

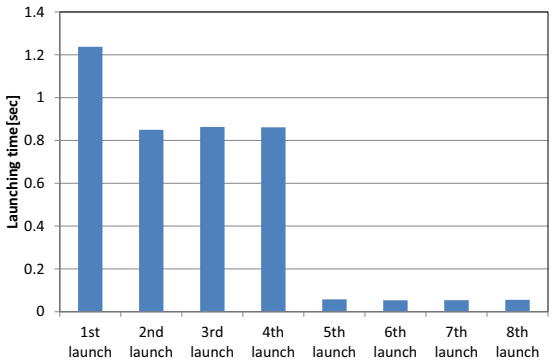

Fig. 3. Launching time (Nexus S)

IV. Decreasing of Application Launching Time

As mentioned above Zygote stays with some important classes loaded. It is expected that application launching time can be decreased by increasing the number of preloaded classes. In this section, we discuss the relation between the number of preloaded classes and the application launching time.

For exploring this relation, we measured application launching time with various numbers of preloaded classes. We launched *Browser*, which is Android standard browser. The experimental results are shown in Fig. 4. The labels in the figure are as shown in Table I. The same device in section III is used. Android version is 4.1.1.

Fig. 4 indicates that the launching time of "without_preload" is much longer than those with preload. Therefore, we can say that preloading of Zygote is an effective method for decreasing application launching time. The figure

also shows that increasing the number of preloaded classes is effective for further improvement.

Table I. PRELOAD SETTINGS

without_preload	no classes were preloaded
normal_preload	the default preload classes in Android 4.1.1(2303 classes) were preloaded
+51_preload	the default classes and 51 classes under `dalvik` were preloaded
+120_ preload	classes of "+51_preload" and 69 classes under `java.net` were preloaded
+281_preload	classes of "+120_preload" and 98 classes under `java.security` were preloaded

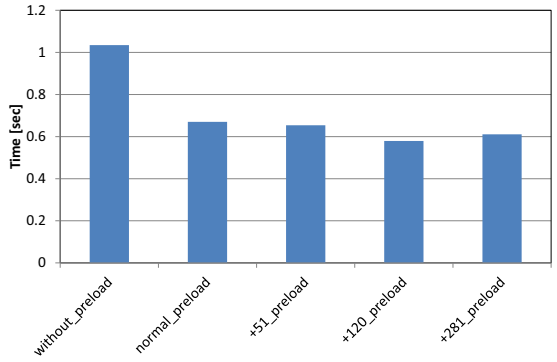

Fig. 4. Increasing the number of preloaded classes (manual selection)

V. RELATED WORK

Following studies are for Android performance. Singh *et al.* focused on operating system boot times [3]. They described system level optimization of embedded software to achieve faster boot times. Their evaluation demonstrated that their optimized Android stack booted 65 percent faster than the existing common approach. This work is useful for fast operating system booting, but the work does not provide a discussion for decreasing application launching times with application frameworks and Zygote. In [4][5][6], performance of Android native application, written in C language, and that of applications running on Dalvik VM are discussed. Then, they concluded that native application have large advantage in performance. In [7], performance of Android virtual machines and general Java virtual machines are compared and energy consumption in discussed. In [8], performance and power consumption of Android devices are comprehensively discussed and a method for saving power with guaranteeing the required performance. However, these works do not focus on application launching, so their contributions differ from ours.

VI. CONCLUSION

In this paper, we introduced our Android application launch analyzing system, which is constructed on open source Android operating system, and presented analytical results. After the analyses, we proposed a method for improving application launching performance and our evaluations have demonstrated that our method has been effective for performance improvement.

We plan to discuss a policy of processes termination to avoid forking a new process instead of resuming an existing process.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Numbers 24300034, 25280022.

REFERENCES

[1] Saneyasu Yamaguchi, Masato Oguchi, Masaru Kitsuregawa, "Trace System of iSCSI Storage Access," IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005), pp. 392-398, (2005)

[2] logcat|Android Developers http://developer.android.com/guide/developing/ tools/logcat.html

[3] Singh, G., Bipin, K., Dhawan, R., "Optimizing the boot time of Android on embedded system," Consumer Electronics (ISCE), pp 503 - 508, 2011

[4] Leonid Batyuk, Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Ahmet Camtepe and Sahin Albayrak, "Developing and Benchmarking Native Linux Applications on Android," Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2009, Volume 7, 381-392

[5] Sangchul Lee, Jae Wook Jeon, "Evaluating performance of Android platform using native C for embedded systems," International Conference on Control Automation and Systems (ICCAS), pp. 1160–1163, 2010

[6] Cheng-Min Lin, Jyh-Horng Lin, Chyi-Ren Dow, Chang-Ming Wen, "Benchmark Dalvik and Native Code for Android System," Innovations in Bio-inspired Computing and Applications (IBICA), pp. 320–323, 2011

[7] Kolin Paul, Tapas Kumar Kundu "Android on Mobile Devices: An Energy Perspective," 10th IEEE International Conference on Computer and Information Technology, 2010.

[8] Nagata, K., Yamaguchi, S., Ogawa, H., "A Power Saving Method with Consideration of Performance in Android Terminals," Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012