# EIMOS: Enhancing Interactivity in Mobile Operating Systems

Sunwook Bae[1], Hokwon Song[1], Changwoo Min[1], Jeehong Kim[2], and Young Ik Eom[2]

[1] Samsung Electronics Co., Ltd., Suwon, Korea
[1,2] School of Information and Communication Engineering
Sungkyunkwan University, Suwon, Korea
{swbae98,hokwon,multics69,jjilong,yieom}@ece.skku.ac.kr

**Abstract.** Interactivity is one of the most important factors in the computing systems. There has been a lot of research to improve the interactivity in traditional desktop environments. However, few research studies have been done for interactivity enhancement in mobile systems like smart phones and tablet PCs. Therefore, different approaches are required to improve the interactivity of these systems. Even if multiple processes are running in a mobile system, there is only one topmost process which interacts with the user due to the resource constraints like small screen sizes and limited input methods. In this paper, we propose EIMOS, a system which identifies the topmost process and enhances the interactivity. Our system improves the CPU process scheduler and I/O prefetcher in the mobile operating system. We also implement EIMOS in the Android mobile platform and performed several experiments. The experimental results show that the performance is increased up to 16% compared to that of the existing platform.

**Keywords:** Interactivity, Topmost process, Mobile system, Operating system.

## 1 Introduction

The computing hardware technology has been rapidly developed, but the issue of interactivity still remains due to multitasking and software bloat. Many processes execute concurrently in the system; some interact with users in the foreground, and the others run just in the background. These processes need sufficient system resources like CPU, memory, and I/O to run smoothly. The traditional operating systems allocate those resources to the processes that are more important to the user first, e.g., interactive processes. In the mobile systems, multitasking is also supported, but the performance of them is lower than desktop systems due to the limited system resources. Therefore it is more important to improve the interactivity in the mobile devices.

There has been lots of research to improve the interactivity in the operating systems of desktop environments [3, 6, 8, 9, 10, 11, 12]. However, few research studies exist for that issue in the mobile systems. In desktop environments, it is possible for the user to work with multiple foreground processes on a single screen. For example, it is feasible to run a

B. Murgante et al. (Eds.): ICCSA 2012, Part III, LNCS 7335, pp. 238–247, 2012.

translator or a calculator while working on a word processing program. In mobile systems, the multitasking feature is also supported but they use different methods to interact with users due to small screen sizes and touch-based input methods. With those restrictions, only one topmost process is used to interact with the user in most cases despite of supporting the multitasking feature [1]. The interactivity experienced by mobile system users depends on the topmost process as in Fig. 1.

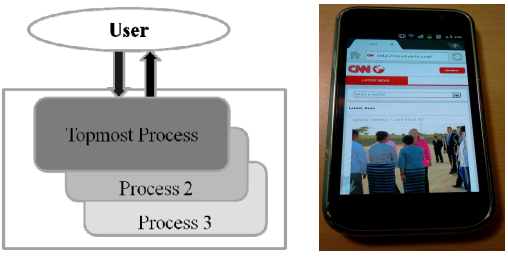


**Fig. 1.** Topmost process in mobile system

In this paper, we present EIMOS, a system to improve the interactivity by considering the mobile system characteristics. EIMOS identifies the topmost process and favors it to enhance interactivity of the system. It improves the process scheduler and prefetching mechanism of the I/O data. EIMOS also has a small runtime overhead and small amount of modifications to the existing operating systems. Specific contributions are as follows:

- Identify the interactive process with a small runtime overhead
- Adaptively apply the process scheduling and I/O prefetching for interactivity
- As far as we have known, this is the first research to improve the interactivity of the mobile operating systems.

The rest of the paper is organized as follows: we review related work in Section 2 and describe the key ideas of the paper and the implementation details in Section 3. Section 4 gives experimental evaluation, and we present conclusion and future work in Section 5.

## 2 Related Work

This section describes representative related work. We start by reviewing the Linux kernel scheduler. Moreover, we review typical research activities that are proposed to improve the interactivity on the desktop environments.

### 2.1 Process Scheduler in the Linux Systems

In the Linux kernel, the O(1) scheduler was introduced in the early version 2.6 by Ingo Molnar [2, 3]. The scheduler improved the interactivity of the interactive processes in two ways. The first was to give a dynamic priority [2] to the interactive

processes by analyzing the average sleep time of the processes. The second was to let the interactive processes remain in the active queue [2] even if the time slice expired. It therefore removed the delay by other processes. From the Linux kernel 2.6.23, Completely Fair Scheduler (CFS) [4, 5] scheduler has replaced by the old O(1) scheduler. The design goal of the CFS scheduler is to provide fair CPU resource allocation for executing processes. However, it also optimized the interactivity by adjusting the virtual runtime value of the processes that slept for a period of time [6].

Lo et al. [7] proposed Modified Interactive Oriented Scheduler (MIOS), a scheduler that improves the interactivity by eliminating unnecessary overhead from the Linux scheduler. MIOS achieved the improvement by removing the overhead caused by maintaining two queues, active and expired, in the O(1) scheduler.

Above three studies used the limited information, such as the average sleep time, to identify the interactive processes. Because of the lack of information, they cannot detect the right interactive processes even if they improved the performance and the scalability of the scheduler.

## 2.2 Research in the Desktop Environment

### 2.2.1 Identification of the Interactive Processes

Etsion et al. [8, 9] suggested that multimedia processes should be treated in a special way as well as interactive processes. They defined these as Human Centered (HuC) processes which are detected based on the display output production. They also showed that the performance of the HuC processes was not degraded even if heavy background processes were running. However, the approach is not suitable in the mobile system because there are still many interactive processes which are not related to multimedia jobs, such as messenger and calendar applications.

Zheng and Nieh [10] presented RSIO, an approach improving the response time of interactive latency-sensitive processes. RSIO identified the interactive processes dynamically by monitoring the I/O channels usage for user interactions and then boosted the priority of interactive processes when they handled latency-sensitive activities. The problem of this approach is that some I/O channels, which are suggested in RSIO such as tty and mouse devices, cannot be applied to the mobile system and it is not enough to improve the interactivity by adjusting priorities in the processor scheduler.

### 2.2.2 OS Support for Improving the Interactivity

The previous research was focused on identifying the interactive processes and prioritizing them in the process scheduler to improve the interactivity. Yan [11], however, presented a holistic approach addressing process scheduling, memory management, and I/O scheduling. He proposed a new process scheduling policy, LRU memory management system, and disk I/O scheduling policy. He showed the improvement of computer responsiveness by modifying on the existing Linux/X desktop system.

Yang proposed Redline [12], a system designed to support interactive and resource-intensive modern applications in commodity operating systems. It maximized the

responsiveness of interactive applications by orchestrating memory and disk I/O management with the CPU scheduler.

Above two studies focused on the desktop environment and modified the operating systems entirely. The suggested policies of memory management and disk I/O scheduling also have a large runtime overhead. Therefore, it is hard to apply them in the mobile operating systems.

# 3 Design and Implementation of Eimos

In this section, we introduce a design and implementation of EIMOS in detail. EIMOS is a system that improves the interactivity in the mobile systems. EIMOS consists of two steps. The first step is to identify the topmost process which is an important process for interactivity. The second step is to improve the interactivity by allocating the CPU/IO resources to the topmost process first.

## 3.1 Identification of the Topmost Process

The method to identify the topmost process depends on the mobile operating systems. It is also impossible to detect the process in the OS kernel layer alone because the information can be managed by the UI or windows framework. In case of the Android mobile platform, we can use a low memory killer module which was newly added to the Linux kernel to address the out-of-memory problem [13]. The main role of the low memory killer module is to kill the less important processes depending on the information of Table I. The `oomkilladj` variable in the `task_struct` structure which handles the process information is used to classify the processes and the variable is updated from the ActivityManager which is a component to handle the UI information in the Android mobile platform. According to the `oomkilladj` variable, the process groups are divided into seven. The low memory killer module finds the less important processes from the groups of high `oomkilladj` values, such as `EMPTY_APP` and `HIDDEN_APP`. EIMOS, however, identifies the topmost process by using the information of the `FOREGROUND_APP` group. This approach is simple and has less runtime overhead comparing with previous studies.

**Table 1.** `oomkilladj` values for the classes of processes

| Group | Oomkilladj |
|---|---|
| SYSTEM | -16 |
| **FOREGROUND_APP** | **0** |
| VISIBLE_APP | 1 |
| SECONDARY_SERVER | 2 |
| HIDDEN_APP | 7 |
| CONTENT_PROVIDER | 14 |
| EMPTY_APP | 15 |

### 3.2 Scheduler Support

There can be many methods to improve the interactivity of the topmost process. We first present to allocate the CPU resource to the topmost process by giving an additional bonus to the process in the OS scheduler. Fig. 2 shows the workflow of the scheduler support in EIMOS. First, we add a topmost flag to all `task_struct` structures and initialize it to a value of `false` on device booting time. Second, we identify the topmost process by monitoring the low memory killer module and set the topmost flag of the process to a value of `true`. Third, we adjust the priority of the topmost process in the OS scheduler. Finally, if the topmost process is changed, we modify the topmost flag and the priority of the previous process to an old value. The Linux O(1) scheduler and CFS scheduler are the priority-based scheduler and the processes with higher priorities get better response time. We give a dynamic priority bonus of 19 to the topmost process in EIMOS. This approach has a little impact on other processes and the interactivity of the topmost process does not significantly decrease in situation of running lots of background processes.

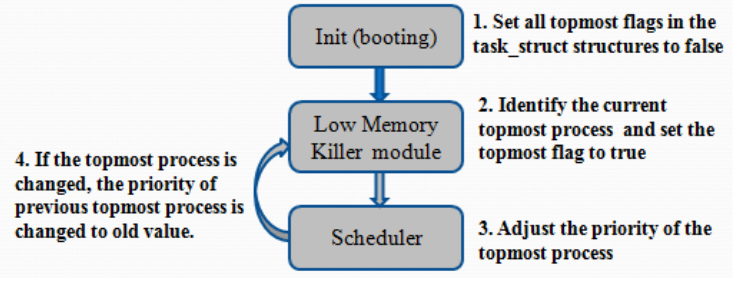


**Fig. 2.** Workflow of process scheduler support in EIMOS

### 3.3 I/O Prefetch Support

The I/O performance is always a bottleneck of the computer systems. Recently, there are many applications to handle big data like music videos and movies. In these cases, the I/O prefetch technique which loads the I/O data to memory in advance is very effective due to hiding the I/O latency. We therefore propose the interactivity aware adaptive prefetch scheme here. The previous prefetch scheme in the Linux kernel used the information based on the sequentiality, but we consider the interactivity additionally. We implement the read-ahead policy to read more adjacent pages of data when the topmost process requests the system to read the I/O data. Fig. 3 shows the workflow of the I/O prefetch support in EIMOS. In the Linux kernel, the maximum size of the read-ahead buffer is fixed but we also extend it eight times in case of the topmost process. This approach can improve the I/O performance of the topmost process with a little modification.

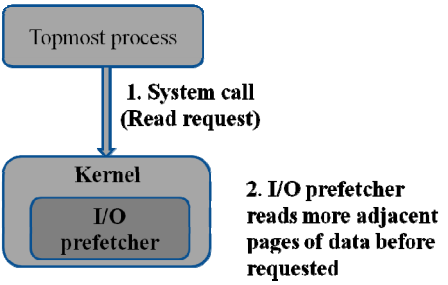


**Fig. 3.** Workflow of I/O prefetch support in EIMOS

## 4 Evaluation

The hardware environment for the experiments is shown in Table II. We implemented EIMOS by modifying the Linux kernel sources in the Android 2.2 froyo emulator. We developed two micro benchmarks and used one more realistic workload for evaluation: CPU and IO bound micro-benchmarks and a Linpack benchmark [14] which is a measure of a system's floating point computing power. Linpack has been used for years on all types of computers and it shows the performance of the topmost process. We can evaluate the improvement of the interactivity by using these benchmarks and by comparing between EIMOS and original Android emulator.

**Table 2.** Experiment environment

| | |
|---|---|
| CPU | Intel(R) Core(TM) i5 CPU 2.40 GHz |
| Memory | 4 GB RAM |
| OS | Ubuntu 10.10 (Linux) |

We first made a simple micro-benchmark for measuring the CPU performance of EIMOS. Fig. 4 shows the source code of the benchmark. We ran this micro-benchmark 50 times and measured the average time of them as in Fig. 5. The first graph shows the result by running this micro-benchmark alone and the second graph is the result by running the micro-benchmark with 10 stresses of the same benchmark application running on the background. Without any background loads, the performance of EIMOS improves by 4%. As the load on the system increases, the performance of EIMOS improves by 16%.

```
For (i=0; i<1000000; i++)
          value += i;
          For (j=0; j<1000000; j++)
                    value += j;
```

**Fig. 4.** Source code of CPU bound micro-benchmark

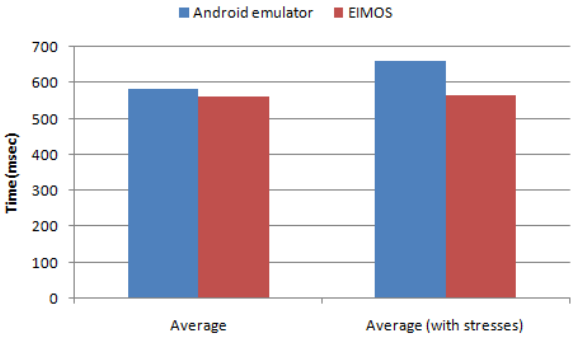


**Fig. 5.** Average time of CPU bound micro-benchmark

Fig. 6 shows the code of our micro-benchmark for measuring I/O performance. The benchmark reads 48 bytes data sequentially from the file of total 30MB size and finishes when it reaches the end of the file. We ran the micro-benchmark by changing the maximum size of the read-ahead buffer by 8 times and 16 times additionally. Fig. 7 shows the result of the average time of 10 trials. We can see that the I/O performance improved by 5% and 7% respectively. Although this I/O operation performs the sequential read requests, there was not much improvement from the experiments than we expected. We should consider the page cache of the Linux kernel for I/O operation. Fig.8 shows the result of taking time to read a same file of 30MB size repeatedly. We can see little improvement after first trial because the page cache contains the previous I/O data. Therefore, we have a future research plan to improve the I/O performance in EIMOS considering the page cache.

```
byte[] buffer = new byte[48];
FileInputStream fis
 = openFileInput("test.data");
   BufferedInputStream buf
 = new BufferedInputStream(fis);
while (true) {
       num = buf.read(buffer);
       if (num < 0)
              break;
}
```

**Fig. 6.** Source code of I/O bound micro-benhmark

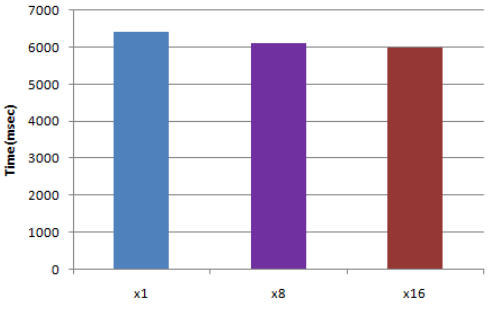


**Fig. 7.** Average time of I/O bound micro-benhmark

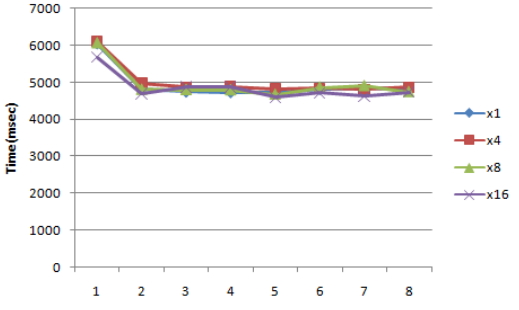


**Fig. 8.** Time to read same data repeatedly

Fig. 9 shows the result by running the Linpack 1.2.8 benchmark with and without background stresses. EIMOS also improves the performance maximum 18 times in the background stresses of the 10 micro-benchmark. In computing systems, if the background workloads increase, the performance of the topmost process should be degraded. However EIMOS shows the consistent performance in the heavy background stresses because EIMOS identifies the topmost process and boosts the priority of the process.

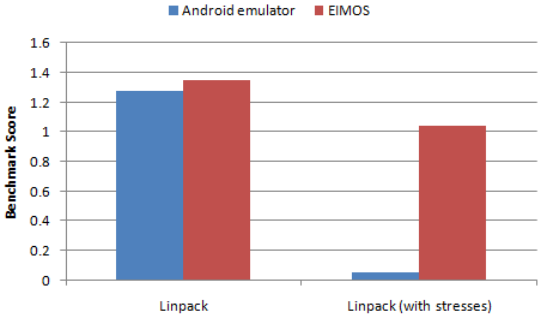


**Fig. 9.** Linpack 1.2.8 benchmark score

## 5 Conclusions and Future Work

This paper presents EIMOS, a new approach to enhance the interactivity on the mobile operating systems. EIMOS identifies the topmost process which has the biggest impact on user responsiveness and supports the process by adjusting the CPU scheduling and using the I/O prefetching technique. We implemented EIMOS in the Android mobile platform and the experiment results showed that the CPU performance of the topmost process was improved by 16% and the I/O performance by 7% in the heavy background stresses.

In future work, we will consider not only memory management, but also I/O scheduling to improve the interactivity of the topmost process. We are also interested in other resource management algorithms which are best suited for the mobile environments.

**Acknowledgments.** This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0025971).

## References

1. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in Smartphone Usage. In: Mobile Systems, Applications and Services (MobiSys) (June 2010)
2. Bovet, D.P., Cesati, M.: Understanding the Linux Kernel, 3rd edn. O'Reilly (2006)
3. Molnar, I., Kolivas, C.: Interactivity in Linux 2.6 Scheduler (2003), http://www.kerneltrap.org/node/780
4. Molnar, I.: Linux CFS Scheduler (2007), http://kerneltrap.org/node/11737
5. Molnar, I.: A description of CFS design, http://people.redhat.com/mingo/cfs-scheduler/sched-design-CFS.txt
6. Wong, C.S., Tan, I.K.T., Kumari, R.D., Lam, J.W., Fun, W.: Fairness and Interactive Performance of O(1) and CFS Linux Kernel Schedulers. In: Information Technology, ITSim 2008 (2008)
7. Lo, L., Lee, L.T., Chang, H.Y.: A Modified Interactive Oriented Scheduler for GUI-based Embedded Systems. In: Computer and Information Technology (July 2008)
8. Etsion, Y., Tsafrir, D., Feitelson, D.G.: Desktop scheduling: How Can We Know What the User Wants? In: Proc. of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video. ACM Press (2004)
9. Etsion, Y., Tsafrir, D., Feitelson, D.G.: Process Prioritization Using Output Production: Scheduling for Multimedia. ACM Transactions on Multimedia Computing, Communications and Applications (November 2006)
10. Zheng, H., Nieh, J.: RSIO: Automatic User Interaction Detection and Scheduling. In: Proc. of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (June 2010)

11. Yan, L., Zhong, L., Jha, N.K.: Towards a Responsive, Yet Power-Efficient, Operating System: A Holistic Approach. In: Proc. of the 13th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (September 2005)
12. Yang, T., Liu, T., Berger, E.D., Kaplan, S.F., Moss, J.E.B.: Redline: First Class Support for Interactivity in Commodity Operating Systems. In: Proc. of the 8th Symposium on Operating Systems Design and Implementation (December 2008)
13. Linux/drivers/staging/android/lowmemorykiller.c, `http://lxr.free-electrons.com/source/drivers/staging/android/lowmemorykiller.c?v=2.6.29`
14. Linpack for Android, `http://www.greenecomputing.com/apps/linpack`