The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 5, pp.33-38, Oct. 31, 2016. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2016.16.5.33
JIIBC 2016-5-6

안드로이드 IPC 가속화를 위한 커널 바인더 캐쉬의 설계 및 구현

Design and Implementation of Kernel Binder Cache for Accelerating Android IPC

연제성*, 고건**, 이은지***

Jeseong Yeon*, Kern Koh, Eunji Lee*****

요 약 현재 안드로이드는 유저 레벨의 데몬들을 통하여 시스템에 필요한 기능을 서비스 함수로 지원하고 IPC를 사용해 호출되도록 한다. 그런데 서비스 함수를 관리하는 작업이 빈번하게 사용되는 Critical Path 임에도 불구하고, 유저 레벨 프로세스가 수행하도록 되어 있다. 이러한 분리된 구조는 모듈성과 유연성의 관점에서 효율적이지만 복잡한 소프트웨어 스택 및 컨텍스트 스위치 오버헤드 등으로 서비스 응답 시간이 상당히 저하된다. 본 논문에서는 안드로이드 IPC 매커니즘의 병목점이 되는 부분을 분석하고 개선함으로써 이러한 문제를 해결한다. 우리는 IPC 지연 시간 중 55%가 커널과 컨텍스트 매니저 사이의 커뮤니케이션 오버헤드인 것을 발견하고, 서비스 함수 중 자주 접근되는 것들에 대한 정보는 커널 내에 캐쉬 형태로 유지하는 기법을 제안한다. 제안된 IPC 캐쉬는 안드로이드 5.0에 구현되었으며, 다양한 모바일 벤치마크를 통해 성능평가를 수행한 결과 52.9%의 성능이 향상되었다.

Abstract In Android platform, as applications invoke various service functions through IPC (Inter-Process Communication), IPC performance is critical to the responsiveness in Android. However, Android offers long IPC latency of hundreds of micro-seconds due to complicated software stacks between the kernel Binder and the user-level process Context Manager. This separation provides modularity and flexibility, but degrades the responsiveness of services owing to additional context switching and inefficient request handling. In this paper, we anatomize Android IPC mechanisms and observe that 55% of IPC latency comes from the communication overhead between Binder and Context Manager. Based on this observation, this paper proposes a kernel Binder cache that retains a popular subset of service function mappings, thereby reducing the requests transferred to the user-level daemon. The proposed Binder cache is implemented in Android 5.0 and experimental results with various benchmarks show that the proposed cache architecture improves performance by 52.9% on average.

Key Words : Android, IPC (Inter-process Communication), Mobile System, Binder System

Ⅰ. 서 론

안드로이드는 현재 모바일 시스템에서 가장 널리 사용되고 있는 운영체제 플랫폼이다[1]. 안드로이드는 기본적으로 리눅스 커널 위에서 구동되지만, 모바일 환경에 최적화된 서비스 기능을 추가적으로 제공한다. 이러한

*준회원, 충북대학교 소프트웨어학과
**정회원, 서울대학교 컴퓨터공학과
***정회원, 충북대학교 소프트웨어학과(교신저자)
접수일자 : 2016년 7월 1일, 수정완료 : 2016년 8월 5일
게재확정일자 : 2016년 10월 7일

Received: 1 July, 2016 / Revised: 5 August, 2016 /
Accepted: 7 October, 2016
***Corresponding Author: eunji@cbnu.ac.kr
Dept. of Computer Science, Chungbuk National University, Korea

서비스 기능은 유저 레벨의 데몬들에 의해 제공되므로, 사용자 수준 애플리케이션은 IPC (Inter-Process Communication) 매커니즘을 통해 시스템 서비스 기능을 호출한다. 그러나 이러한 구조는 소프트웨어 플랫폼의 모듈성 및 유연성 측면에서는 효율적이지만 사용자 프로세스와 커널 사이에 빈번한 커뮤니케이션을 유발하기 때문에 고속의 IPC 지원이 필수적이다.

현재 안드로이드는 커널 모듈의 가상 디바이스 드라이버인 바인더를 통해 프로세스들 간의 IPC를 지원한다. 바인더는 애플리케이션이 유저 레벨의 서비스 제공자가 제공하는 외부 서비스 함수를 호출 할 때 사용된다. 예를 들어, 애플리케이션이 배터리 상태를 확인하기 위해 batterystats() 함수를 호출하기 위해서는 /dev/binder/에 IPC 데이터를 쓰기 형태로 전송하고, 바인더 드라이버는 요청된 함수가 현재 시스템에서 제공되고 있는지 검색을 수행하게 된다[2]. 문제는 현재 바인더는 서비스 함수 검색 작업을 직접 수행하지 않고 유저 레벨의 프로세스인 컨텍스트 매니저에게 전달하여 수행토록 한다.

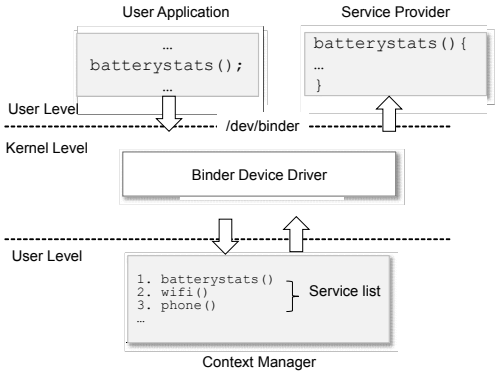


그림 1. 바인더 및 컨텍스트 매니저의 IPC 매커니즘
Fig. 1. IPC mechanism of the binder and context manager

그림 1 은 바인더 가상 디바이스 드라이버 및 컨텍스트 매니저(Context Manager), 서비스 제공자(Service Provider)로 구성된 안드로이드 IPC 매커니즘을 보여준다. 컨텍스트 매니저는 현재 시스템에서 제공되는 함수들을 Service List 형태로 유지하고 바인더를 통해 특정 서비스함수에 대한 검색요청이 발생하면 서비스 리스트를 참조하여 적절한 접근 정보를 제공한다. 이분화 된 검색구조는 작고 강력한 커널 코어를 유지하는 데에는 효율적이지만 컨텍스트 스위치가 빈번하게 발생해 IPC에 상당한 지연을 초래하게 된다.

본 논문에서는 자주 요청되는 서비스 함수를 커널 내부에서 접근 정보를 캐싱함으로써 IPC 성능을 향상시키고자 한다. 분석한 바에 따르면 서비스 함수의 접근패턴은 인기도 상위 25%의 함수에 총 접근의 80%가 발생하는 높은 지역성을 나타내었다. 이에 바인더 캐쉬는 소량의 메모리로 대다수의 서비스 요청을 커널 레벨에서 즉시 제공함으로써 응답성을 크게 향상시켰다.

II. 관련 연구

IPC 효율을 향상시키기 위한 기술은 Micro Kernel 기반의 영역에서 광범위하게 연구되었는데, 커널의 기능을 최소화하고 사용자 레벨 데몬으로 실제 시스템 서비스를 구현하는 것이다. Micro Kernel 의 기본적인 철학은 정책과 매커니즘을 분리함으로써 모듈성과 유연성을 제공하기 위함이다. 이러한 시스템에서는 사용자 프로세스가 IPC를 통해 시스템 서비스를 호출하기 때문에 IPC 오버 헤드를 최소화하는 것이 매우 중요하다.

Liedtke 등은 최적화 기술을 통하여 고속의 메시징 시스템을 제공하는 L4 운영체제를 제안하였다[3]. L4는 프로세스레지스터 안에 직접 작은 메시지를 저장함으로써 추가적인 메모리 접근을 피하였다. 또한, 발신자와 수신자 사이의 빠른 메시지 전송을 위해 일시적인 가상 주소 맵핑을 사용함으로써 IPC 성능을 향상시켰다. Wentzlaff 등은 시스템의 확장성에 초점을 맞춘 fos 운영체제를 기반으로 Micro Kernel 시스템을 설계 및 구현하였다[4]. Elphinstone 등은 20년 동안의 L4 운영체제의 발전과정에 대해 분석하였으며 IPC Latency가 Micro Kernel 시스템의 성능에 가장 중요한 기준으로 남아있음을 보여준다[5]. Baumann 등은 멀티코어 시스템 구성을 가정하고 메시지 전달을 통해 통신하는 분산 처리에 전통적인 운영체제 기능을 옮긴 멀티커널 OS를 제안한다[6]. Choi 등은 IPC 성능 관점에서 별도의 가상 주소 공간을 갖는 것보다 프로세스간 하나의 주소 공간을 공유하는 것이 높은 효율성을 지님을 보여준다[7].

한편, 안드로이드의 IPC 매커니즘에 대한 기존 연구는 대부분 보안 위험을 해결하는 것에 초점을 맞추었다. Chin 등은 안드로이드의 바인더를 통해 응용프로그램 간 메시지를 전달하는 과정에서 발생할 수 있는 보안 위협 및 개인정보 보호 위반을 분석하였다[8]. Ongtang 등은

애플리케이션 검증을 위해 커널 바인더 장치 드라이버에 IPC 데이터를 가로채어 사용하는 세인트라는 보안 구조를 제안하였다[9]. 일부 연구들은 실시간 시스템 상태를 모니터링하기 위해서 바인더의 IPC 데이터를 이용하였다. Yoon 등은 바인더의 IPC 데이터에 근거하여 애플리케이션의 파워 소모를 예측하고 프로파일링하는 전력 관리 기법을 제안하였다[10]. 이러한 연구 외에, 연구의 상당수는 최근의 안드로이드 IPC의 보안 관점에서 행해졌다[11]. 그러나 안드로이드 플랫폼의 IPC 성능을 향상시키기 위한 연구는 전무하며 안드로이드 IPC의 성능을 분석하고 개선하기 위한 연구는 본 논문에서 최초로 시도되는 것이다.

Ⅲ. 안드로이드의 IPC 매커니즘

안드로이드에서 IPC가 발생하는 경우 세 종류의 유저 레벨 프로세스가 참여하는데, 각각 컨텍스트 매니저, 서비스 함수 제공자, 서비스 클라이언트 애플리케이션이다. 컨텍스트 매니저는 시스템의 모든 서비스 함수를 관리하는 유저 레벨의 프로세스로 부팅 시에 실행되어 서비스의 등록 및 검색 기능을 수행한다. 서비스 제공자는 시스템 수준에서 필요한 기능을 실행하는 프로세스이다. 서비스 클라이언트는 서비스 제공자가 제공하는 시스템 함수들을 사용하는 프로세스이다.

1. 서비스 함수 등록

서비스 함수 등록은 함수를 관리하는 컨텍스트 매니저에게 요청된다. 컨텍스트 매니저는 부팅 시 실행되어 타 서비스 제공자들이 함수 등록 요청을 보낼 수 있도록 대기한다. 컨텍스트 매니저는 바인더 드라이버에 자신의 IPC 데이터 수신 버퍼를 확보하고 대기 상태에 진입한다. 향후 서비스 제공자가 자신이 제공하고자 하는 함수를 시스템에 등록하려면 바인더 드라이버를 통해 IPC데이터를 보내야 한다. 예를 들면, 서비스 프로바이더는 자신의 수신 버퍼를 메모리에 할당받고, 컨텍스트 매니저에게 전송할 RPC 코드, RPC데이터, 핸들 값 0번(컨텍스트 매니저를 지칭하는 값임)을 담은 IPC데이터를 생성한다. 서비스 제공자는 ioctl() 시스템 콜을 이용해 바인더 드라이버에 전송하고, 대기 상태에 진입한다. 바인더 드라이버는 ioctl() 시스템 콜을 이용해 컨텍스트 매니저에 전송한다. 컨텍스트 매니저는 IPC데이터의 RPC 코드(ADD_SERVICE), RPC데이터(서비스 이름)를 분석하여 서비스 등록과 서비스 검색을 파악하고, 서비스 이름을 파악해 서비스 목록에 등록한다. 서비스가 정상적으로 등록이 되면 IPC응답 데이터를 생성하여 서비스 제공자에게 전달하고 IPC사이클을 마무리한다.

2. 서비스 함수 검색

서비스 함수 검색은 애플리케이션이 특정 서비스를 사용하기 위해 해당 함수의 주소값을 찾는 과정이다. 검색은 컨텍스트 매니저에게 요청하게 되는데 그림 2는 검색과정을 단계적으로 보여준다. 애플리케이션은 검색할 서비스 이름(RPC데이터)와 컨텍스트 매니저의 서비스 검색 함수인 GET_SERVICE() 함수이름, 컨텍스트 매니저의 핸들 값인 0을 담은 IPC 데이터를 생성한다. 그 후 ioctl() 시스템 콜을 통해 바인더 드라이버에 전달하고(그림 2의 step 1), 바인더 드라이버는 핸들 값이 0 인 프로세스인 컨텍스트 매니저에 IPC데이터를 전달한다. 컨텍스트 매니저는 서비스 검색 함수를 호출하여 해당 서비스 이름을 서비스 목록에서 검색하고, 서비스 기술자를 IPC를 통해 바인더에 전송한다(그림 2의 step 2). 바인더는 응답데이터를 수신할 애플리케이션을 찾아,(그림 2의 step 3). 바인더는 해당 기술자에 해당하는 참조 노드를 컨텍스트 매니저의 참조 트리에서 찾아서 요청한 애플리케이션 참조 트리에 추가한다(step 4, 5). 그 후 애플리케이션에 해당하는 참조 노드 번호를 애플리케이션에게 IPC 를 통해 전송하면, 애플리케이션은 자신의 참조 노드를 이용하여 서비스 함수를 호출할 수 있게 된다.

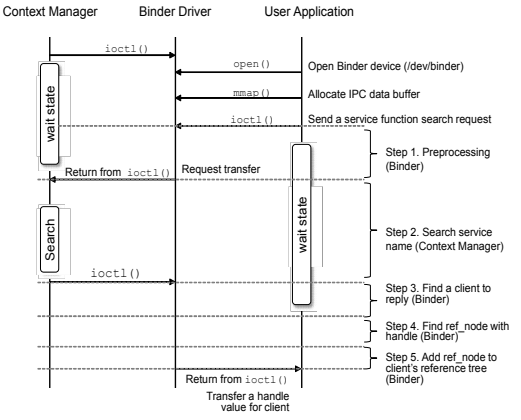


그림 2. 안드로이드에서 서비스 함수를 검색하는 과정.
Fig. 2. Service function search process in Android.

Ⅳ. 안드로이드 커널 캐쉬 설계

1. IPC 성능 저하 요인 분석

안드로이드 IPC는 커널 바인더 와 유저 레벨의 컨텍스트 매니저를 지원하기 때문에 빈번한 컨텍스트 스위치와 소프트웨어 스택 오버헤드가 발생한다. 그림 3은 그림 2의 서비스 함수 검색에 소요되는 시간을 단계적으로 보여준다. 단계 1은 프로세스가 서비스 검색 요청을 하기 위해서 IPC 데이터를 만들어 바인더에 보내는 데에 걸리는 시간이다. 단계 2는 바인더가 컨텍스트 매니저에게 IPC 데이터를 전송하고, 컨텍스트 매니저가 요청을 처리 후 바인더에게 IPC 응답 데이터를 전송하는데 걸리는 시간이다. 단계 3은 컨텍스트 매니저가 요청한 애플리케이션을 찾는데 걸리는 시간이다. 단계 4, 5는 서비스 응답데이터를 바인더에서 파싱 후 서비스에 해당하는 참조 노드를 컨텍스트 매니저의 참조 트리에서 찾은 후 애플리케이션 참조 트리에 추가하는 데 걸리는 시간이다. 서비스 검색시간 중 가장 많이 차지하는 부분은 바로 컨텍스트 매니저에서 서비스를 검색하고, 컨텍스트 스위치가 발생한는 부분이다.

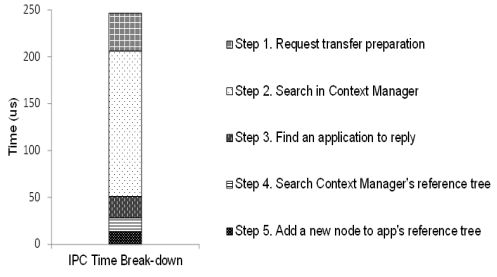


그림 3. 서비스 검색 단계별 소요시간.
Fig. 3. Search time break-down in each step.

또한, 컨텍스트 매니저는 서비스 검색 시 서비스의 참조 횟수와 상관없이 서비스 이름을 가지고 서비스 리스트에서 선형 검색을 수행한다. 그래서 컨텍스트 매니저는 서비스 검색 시 가장 많은 시간을 소비한다.

2. 바인더 캐쉬 디자인 및 구현

우리의 목표는 현재 안드로이드 구조의 견고성 및 안정성을 해치지 않고 안드로이드 IPC 성능을 향상하는 것이다. 이를 위해 자주 접근되는 서비스의 맵핑 정보를 유지하기 위해 커널 바인더 캐시를 채택하였다. 맵핑 정보가 캐시에 존재하는 경우, 컨텍스트 매니저를 호출하지 않고 서비스 기능의 검색을 허용한다.

바인더 캐쉬의 효율을 평가하기 위해 우리는 스마트폰 넥서스 5를 사용하여 서비스 검색 요청을 수집하여 분석하였다. 그림 4는 몽키 벤치마크를 사용하여 수집한 데이터를 서비스 함수 별 서비스 접근 횟수를 나타낸다[12]. 그림에서 보는 바와 같이 서비스함수의 접근은 상당한 지역성의 특성을 보인다. 특히 상위 25%의 서비스들이 서비스 검색 요청의 80%를 차지한다. 그렇기에 바인더 캐쉬는 안드로이드 IPC 성능을 향상시킬 수 있을 것으로 보인다.

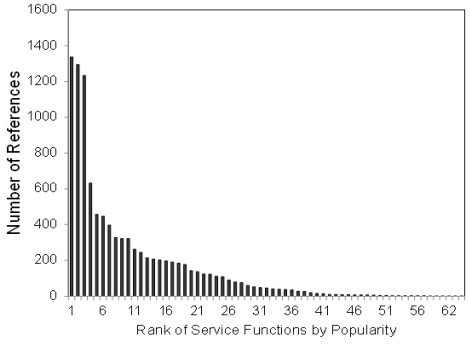


그림 4. 서비스 함수 인기도에 따른 참조횟수
Fig. 4. Locality of service function reference requests.

바인더 캐쉬는 서비스의 이름과 서비스를 나타내는 기술자로 이루어져 있다. 애플리케이션이 서비스를 검색할 경우 바인더는 서비스 이름으로 해당하는 기술자를 찾는다. 그리고 기술자에 해당하는 참조 노드를 컨텍스트 매니저의 참조 트리에서 찾고, 해당 애플리케이션의 참조 트리에 추가한다. 바인더는 애플리케이션의 참조 트리에 추가한 노드에 해당하는 기술자를 애플리케이션에게 반환한다.

바인더 캐쉬는 부팅 시 엔트리가 비어 있기 때문에, 컨텍스트 매니저에게 요청을 전달한다. 그리고 IPC 데이터와 응답 데이터를 파싱 하여 캐시의 엔트리에 추가한다. 만약 바인더 캐쉬에서 히트가 발생하지 않을 경우 캐쉬 교체 기법인 LFU를 사용하여 히트 율을 높인다. 실험을 통하여 LFU(least frequently used)는 LRU(Least Recently Used)보다 적중률이 더 높았기 때문에 LFU를 사용하였다.

바인더 캐쉬의 과제는 캐쉬의 검색 성능을 높이는 것

이다. 서비스의 이름을 가지고 캐쉬의 엔트리를 비교하기 때문에 최악의 경우 캐쉬의 모든 엔트리의 서비스 이름을 비교해야한다. 그래서 비효율적인 사례를 예방하기 위해서, 우리는 블룸 필터를 사용하여 서비스의 이름이 캐쉬에 있는지 체크한다. 블룸 필터는 다중 해시 함수를 사용하여 캐쉬에 해당 서비스가 존재하는 지를 빠르게 확인한다. 그래서 바인더 캐쉬가 미스가 났을 때의 바인더 캐쉬 전체 검색 시간을 피할 수 있었다.

V. 실험 결과

제안된 바인더 캐쉬는 안드로이드 롤리팝 5.0 에 구현되었으며 넥서스 5를 사용하여 성능을 측정하였다. 구체적인 실험 환경은 표1에서 나타나 있다. 그림 5는 다양한 모바일 애플리케이션에 대하여 바인더 캐쉬를 유지할 때의 서비스 함수 검색 시간을 기존의 바인더 기법과 비교하여 보여주는 그래프이다. 그림에서 보는 바와 같이, 제안된 바인더 캐쉬 기법은 서비스 함수 검색시간을 52.9% 단축시켰다. 이것은 바인더 캐쉬 엔트리를 10개로 유지할 때의 결과이며, 매우 소량의 메모리 사용으로 IPC의 성능을 크게 향상시킬 수 있었다.

표 1. 실험 환경 요약
Table 1. Experimental Setup

Hardware	Features
CPU	ARMv7 (2265MHz)
Main Memory	2GB
Storage	16GB
OS	Android 5.0 Lollipop
Kernel version	Linux kernel 3.4.0

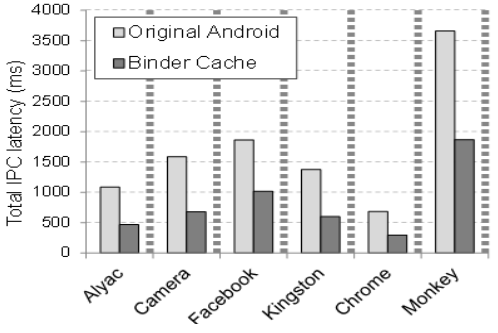


그림 5. 기존 안드로이드 IPC 대비 바인더 캐쉬 성능비교.
Fig. 5. Comparison of original Android with Binder cache.

VI. 결 론

본 논문에서는 안드로이드 IPC 매커니즘에서 컨텍스트 매니저와 바인더 사이의 통신 오버헤드에 따른 비효율성을 분석하였다. 이러한 오버헤드를 감소시키기 위해 커널 레벨에서 서비스의 맵핑 정보를 작게 저장하고 있는 바인더 캐쉬를 제안하였다. 바인더 캐쉬는 자주 사용되는 서비스 함수의 호출정보를 캐싱함으로써 컨텍스트 매니저에게 전송되는 요청들을 상당부분 흡수하였다. 안드로이드 5.0에 구현된 바인더 캐쉬는 측정 결과 IPC 응답시간을 평균 52.9% 향상시켰다.

References

[1] J. Kim, H. Bahn, “Analysis of Users’ Gestures by Application in Smartphone Touch Interfaces,” The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 15, No. 2, pp. 9-14, 2015
[2] http://elinux.org/Android_Binder
[3] J. Liedtke, “Improving IPC by Kernel Design,” Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP), 1993.
[4] D. Wentzlaff and A. Agarwal, “Factored operating systems (fos): the case for a scalable operating system for multicores,” ACM SIGOPS Operating Systems Review, Vol. 43, No. 2, pp. 76-85, 2009.
[5] K. Elphinstone and G. Heiser, “From L3 to seL4 - what have we learnt in 20 years of L4 microkernels?” Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013.
[6] A. Baumann, P. Barham, Pierre-Evariste, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, “The multikernel: a new OS architecture for scalable multicore systems,” Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP). 2011.
[7] H. S. Choi, H. C. Yun, “Context Switching and IPC Performance Comparison between uClinux and Linux on the ARM9 based Processor,” SAMSUNG Tech. Conf., 2005.

[8] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” Proceedings of the 9th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), 2011.

[9] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, “Semantically Rich Application-Centric Security in Android,” Proceedings of the 2009 Annual Computer Security Applications Conference, 2009.

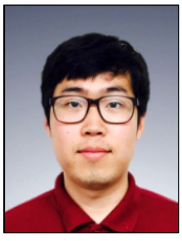
[10] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, “AppScope: application energy metering framework for android smartphones using kernel activity monitoring,” Proceedings of the USENIX Annual Technical Conference (ATC), 2012.

[11] W. Enck, D. Octeau, P. McDaniel, S. Chaudhuri, “A study of android application security,” Proceedings of the 20th USENIX Conference on Security, 2011.

[12] http://developer.android.com/tools/help/monkey.html

저자 소개

연 제 성(준회원)



• 2011년 3월 ~ : 충북대학교 소프트웨어학과 학사과정

<주관심분야 : 운영체제, 파일시스템, 모바일 시스템, 스토리지>

고 건(정회원)

• 1974년 2월 : 서울대학교 계산통계학과 학사
• 1981년 2월 : 버지니아 컴퓨터과학 박사
• 1983년 3월 ~ 2012년 8월 : 서울대학교 컴퓨터공학과 교수
• 2014년 3월 ~ : 서울대학교 명예교수

<주관심분야 : 운영체제, 실시간 시스템, 메모리 및 스토리지 시스템>

이 은 지(정회원)

• 2005년 2월 : 이화여자대학교 컴퓨터공학과 학사
• 2012년 2월 : 서울대학교 컴퓨터공학부 박사
• 2014년 3월 ~ : 충북대학교 소프트웨어학과 조교수

<주관심분야 : 운영체제, 파일시스템, 가상화, 스토리지>

※ 이 논문은 2011년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. NRF-2011-0025633)