JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 22, No. 3: 399~410, Mar. 2018

백그라운드 서비스가 안드로이드 스마트폰의 성능에 미치는 영향

안우현[1] · 오윤석[2] · 오재원[3*]

The Effect of Background Services on Android Smartphone Performance

Woo Hyun Ahn[1] · Yunseok Oh[2] · Jaewon Oh[3*]

[1]School of Software, Kwangwoon University, Seoul 01897, Korea
[2]Naver Corporation, Seongnam 13561, Korea
[3*]School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Korea

요 약

안드로이드 스마트폰에서 많은 앱들이 백그라운드로 실행되기 위해 서비스 앱으로 개발된다. 메모리가 부족하면 오랫동안 CPU를 사용하지 않은 사용자 앱뿐만 아니라 서비스 앱도 강제로 종료된다. 하지만, 서비스 앱은 잠시 후 자동으로 재실행되기 때문에 메모리 공간을 지속적으로 소비한다. 본 논문은 사용자들의 스마트폰에서 실행 중인 서비스 앱의 개수와 메모리 사용량을 조사한다. 서비스 앱의 개수는 전체 실행 중인 앱 개수의 최대 65%, 서비스 앱의 메모리 사용량은 전체 메모리의 최대 55%까지 차지한다. 또한, 실행 중인 서비스 앱의 개수가 스마트폰과 앱의 응답 시간에 미치는 영향을 분석한다. 서비스 앱의 개수가 증가할수록 사용자 앱의 시작 시간이 최대 22배까지 증가한다. 부팅 시간과 앱 설치 시간이 서비스 앱의 개수가 증가함에 따라 크게 증가한다.

ABSTRACT

In Android smartphones, many apps are developed as service apps to run in the background. If the memory is insufficient, Android forcibly terminates not only user apps that have not used the CPU for a long time, but also service apps. However, a service app is automatically re-launched after a short period of time, so that it continuously consumes memory space. This paper analyzes the number of running service apps and their memory usage in users' smartphones. The number of service apps accounts for up to 65% of the total number of running apps, and their memory usage accounts for up to 55% of the total memory. Moreover, we investigate the effect of the number of running service apps on the response time of smartphones and apps. As the number of service apps increases, the launching time of user apps increases to 22 times. The booting time and app installation time significantly increase with the number of service apps.

키워드 : 안드로이드 운영체제, 앱 상태, 서비스 앱, 메모리 관리

Key word : Android operating system, App state, Service app, Memory management

Received 27 December 2017, **Revised** 5 January 2018, **Accepted** 17 January 2018

*** Corresponding Author** Jaewon Oh(E-mail:jwoh@catholic.ac.kr, Tel:+82-2-2164-4869)
School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Korea

Open Access http://doi.org/10.6109/jkiice.2018.22.3.399 pISSN:2234-4772

Ⅰ. 서 론

스마트폰에서 사용자와 상호 작용 없는 백그라운드 실행으로 음악을 재생하거나, 뉴스, 날씨 정보를 주기적으로 업데이트하거나 SNS 앱에 알림이 전달될 수 있다. 사용자에게 보이지 않는 모든 앱은 백그라운드 앱으로 정의된다. 백그라운드 앱에 속하는 앱 중 한 가지로 서비스 앱이 있으며, 서비스 앱은 안드로이드가 제공하는 서비스 컴포넌트[1]를 포함하는 앱이다. 서비스 앱은 한번 실행되면 시스템 종료까지 종료되지 않으며, 부팅 시 자동으로 실행되어 CPU, 메모리, 배터리를 지속적으로 소비한다. 따라서 서비스 앱의 개수가 증가할수록 스마트폰의 성능을 저하시킬 수 있다.

기존 연구[2,3]는 서비스 앱을 포함한 백그라운드 앱들이 화면이 꺼졌을 때에도 적지 않은 배터리, 네트워크 등의 자원을 소비하는 문제를 해결하기 위한 기법을 제시하였다. 이들 연구는 서비스 앱만이 아닌 전반적인 백그라운드 앱의 자원 소비를 고려했지만, 백그라운드 앱들 중에 서비스 앱의 비율과 서비스 앱들의 자원 소비량을 분석하지 않았다. 또한 서비스 앱들이 메모리를 지속적으로 소비함에도 불구하고 서비스 앱이 야기한 메모리 부족 현상이 사용자와 상호 작용하는 포그라운드 앱의 성능에 미치는 영향을 분석하지 않았다.

시스템과 앱의 성능을 좌우하는 요소 중 한 가지가 메모리 관리 기법이다. 안드로이드는 LRU 기반의 메모리 관리 기법[4]을 지원한다. 메모리 부족 시 가장 오랫동안 CPU를 소비하지 않은 앱을 희생 앱으로 선택하고 강제 종료하여 가용 메모리를 확보한다. 종료된 앱을 다시 실행하면 앱의 이전 상태가 모두 손실되었기 때문에 앱은 초기 상태로 실행된다. 이때 앱이 실행될 때까지 걸리는 시간, 즉 시작 시간(launching time)은 종료되지 않은 앱에 비해 종료된 앱을 실행할 때 더욱 크다. 기존 연구[5,6]는 앱의 시작 시간을 줄이기 위한 메모리 관리 기법들을 제안하였다. 하지만, 이들 연구는 서비스 앱의 실행으로 야기되는 과도한 메모리 사용을 고려하지 않았다.

메모리가 극도로 부족하면 안드로이드의 메모리 관리 기법은 사용자 앱뿐만 아니라 서비스 앱도 강제적으로 종료한다. 하지만, 본 논문은 강제 종료된 서비스 앱이 사용자의 실행 요청이 없더라도 잠시 후 자동으로 재실행될 수 있음을 안드로이드 소스 코드의 분석을 통해 확인하였다. 결국 서비스 앱은 실질적으로 종료 없이 메모리 공간을 지속적으로 소비한다. 그래서 서비스 앱의 개수가 증가할수록 가용 메모리 공간이 크게 감소된다. 이런 감소는 자주 사용되는 앱들을 종료시켜서 이후에 다시 실행할 때의 시작 시간을 증가시킨다.

본 논문은 스마트폰에서 서비스 앱이 소비하는 메모리가 시스템과 사용자의 앱의 성능에 미치는 영향을 분석하기 위한 기법을 소개하고, 이 기법을 사용하여 다음과 같은 분석을 수행하였다. 첫째, 사용자의 스마트폰에서 백그라운드 앱의 개수와 이들 중 서비스 앱의 비율을 측정한다. 이들 서비스 앱이 얼마나 많은 메모리 소비를 하는지 분석하고, 안드로이드 메모리 관리 기법이 가용 메모리 확보에 효과적인 기법인지를 확인한다. 둘째, 사용자의 스마트폰에서 분석된 서비스 앱의 개수를 기반으로 서비스 앱이 시스템의 부팅 시간, 앱의 시작 시간과 설치 시간에 미친 영향을 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구, 3장에서는 연구 방법을 설명한다. 4장에서는 실험과 분석 결과에 대해 설명하고, 5장에서는 결론을 도출하여 정리한다.

Ⅱ. 관련 연구

안드로이드 앱은 액티비티, 서비스, 브로드캐스트 리시버(이하 리시버), 콘텐트 프로바이더(이하 프로바이더) 컴포넌트들로 구성된다[7]. 액티비티는 화면에서 사용자가 대면하는 UI를 제공한다. 서비스는 화면 없이 백그라운드에서 동작하며, 사용자와 상호 작용을 하지 않는다. 리시버는 시스템이나 앱이 이벤트의 한 종류인 브로드캐스트를 발생시키면 이 브로드캐스트를 위해 정의된 동작을 수행한다. 프로바이더는 앱의 데이터를 다른 앱이 접근할 수 있도록 인터페이스를 제공한다.

앱은 리눅스 프로세스에서 실행되며, 프로세스는 앱 구성 요소와 동작을 바탕으로 8개의 상태[8]를 가진다. 포그라운드(foreground)는 화면에 보이는 앱의 상태, 비저블(visible)은 화면에 보이지만 희미하게 가려진 앱의 상태, 퍼셉터블(perceptible)은 음악 앱처럼 화면에 보이지 않지만 인지할 수 있는 앱의 상태, 홈(home)은 백그라운드로 실행되는 홈 또는 런처 앱의 상태, 서비스 A와 B는 서비스 컴포넌트를 가지며 백그라운드에 있는

앱의 상태, 프리비어스(previous)는 바로 이전에 실행되었으며 백그라운드에 있는 앱의 상태, 캐시드(cached)는 앞의 상태에 해당되지 않는 백그라운드 앱의 상태를 나타낸다. 아울러, 최근에 실행된 서비스 앱은 서비스 A 상태를 가지며, 실행된 지 30분 이상 지난 서비스 앱은 서비스 B 상태로 전환된다.

앱은 포그라운드 상태에서 사용자가 백 버튼을 누르면 캐시드 상태로, 프리비어스 상태에서 다른 앱이 실행되면 캐시드 상태로 전환된다[9]. 사용자가 자주 사용하는 앱은 이 두 가지 동작으로 인해 캐시드 상태, 즉 캐시드 앱으로 전환될 수 있다. 포그라운드를 제외한 모든 상태의 앱은 백그라운드 앱으로 분류된다. 아울러, 서비스 앱은 리시버를 실행 중이거나 startForeground()를 호출하면 비록 화면에 보이지 않더라도 포그라운드 상태로 전환된다.

Table. 1 App states and kill priorities in the Android operating system

State	Kill priority	State	Kill priority
Foreground	0	Home	6
Visible	1	Previous	7
Perceptible	2	Service B	8
Service A	5	Cached	9 ~ 15

LMK(Low Memory Killer)[4] 프로세스는 메모리가 부족할 때 앱의 종료 우선순위를 통해 앱들을 종료시켜서 메모리를 확보한다. 표 1은 앱의 상태에 따라 설정된 종료 우선순위 값을 나타내며, 정지 상태로 오랫동안 실행되지 않을수록 높은 종료 우선순위 값으로 변한다. 메모리 부족 시 실행되는 LMK는 종료 우선순위가 가장 높은 앱부터 종료시킨다. 종료되는 앱은 CPU 소비 없이 가장 오랫동안 정지된 앱이기 때문에 희생 앱을 선택할 때 LRU 교체 방법을 사용하는 것과 동일하다. 아울러, 다른 앱들을 실행하다가 이전에 실행한 앱을 실행할 때 초기 화면부터 보이거나 실행이 오래 걸리는 경우가 있다. 그 이유는 LMK에 의해 종료된 앱을 다시 실행하였기 때문이다. 결국 메모리가 부족하면 사용자가 자주 사용하는 앱이 빈번히 종료되며, 이는 사용자에게 앱을 실행할 때 불편함을 줄 수 있다.

서비스 앱은 스마트폰 외부 또는 다른 앱으로부터 요청을 받으면 해당 요청에 대한 서비스를 수행한다. 언제든지 발생할 수 있는 요청을 서비스하기 위해서 지속적인 실행을 보장해야 한다. 그래서 서비스 앱은 LMK에 의해 강제 종료되더라도 짧은 시간 내에 재실행된다. 하지만, 서비스 앱이 오랫동안 사용되지 않을 수 있음에도 불구하고 메모리 공간을 불필요하게 소비하여 실질적인 가용 메모리 공간을 감소시킨다. 이런 감소는 메모리 부족을 더욱 심화함으로써 다음과 같은 문제를 야기한다. 첫째, LMK로 하여금 자주 사용하는 앱을 종료하게 한다. 이후 사용자가 종료된 앱을 다시 실행하면 시작 시간이 크게 증가할 수 있다. 둘째, LMK의 실행 횟수를 증가시킨다. LMK 실행은 CPU 부하를 야기하고, 앱 실행을 간섭하기 때문에 실행 횟수가 증가하면 시스템 성능과 사용자의 앱 실행 성능을 저하시킨다. 셋째, LMK가 캐시드 앱뿐만 아니라 서비스 앱들까지 강제 종료하게 할 수 있다. 하지만 이들 서비스 앱은 잠시 후 재실행된다. 이런 반복적인 재실행은 CPU 부하를 야기하기 때문에 사용자와 상호 작용하는 앱의 실행을 방해한다.

기존 연구[5][6]는 메모리 부족 시 앱의 시작 시간을 줄이기 위한 메모리 관리 기법을 제안한다. [5]는 순차적으로 실행되는 앱들의 패턴을 분석하여 미래에 실행될 가능성이 높은 앱들을 예측한다. 메모리 부족 시 실행될 가능성이 높은 앱들을 제외한 앱들을 강제 종료하여 사용자가 빈번히 실행하는 앱의 시작 시간을 감소시킨다. [6]은 앱의 과거 실행 빈도, 시작 시간과 메모리 사용량을 사용하여 메모리 부족 시 강제 종료할 앱을 선택하는 기법을 제안하였다. 하지만, 이들 기법은 서비스 A와 B 상태가 아닌 캐시드 앱들만 교체 대상으로 선택한다. 메모리가 매우 부족하면 캐시드 앱이 없을 경우가 자주 발생할 수 있다. 이런 상황에서 이들 기법은 앱 시작 시간을 줄이는 데 효과를 발휘하지 못한다.

기존 연구[2,10]는 화면이 꺼져 있을 때 서비스 앱의 동작으로 인해 배터리가 많이 소비된다는 문제를 해결하기 위한 기법을 제시한다. [2]는 사용자에게 사용하지 않는 서비스 앱을 보여 주고, 사용자가 선택한 서비스 앱을 특정 시간 동안 정지 상태로 전환하여 CPU 소비를 줄인다. [10]은 지수 백오프(exponential backoff)를 사용하여 서비스 앱 실행을 제한하여 배터리 소비를 줄인다. 그러나 이들 기법은 사용자가 사용하지 않는 서비스 앱이 메모리를 불필요하게 점유하여 시스템 성능

을 저하하는 문제점을 고려하지 않는다.

최근 스마트폰의 절전을 위해 삼성전자 갤럭시에 스마트 매니저 앱[11]을 탑재하였다. 스마트 매니저는 3일 동안 사용하지 않은 앱을 절전 상태로 전환한다. 절전 상태로 변경된 앱은 더 이상 실행되지 않기 때문에 배터리뿐만 아니라 CPU와 메모리 소비를 줄일 수 있다. 하지만, 절전 상태의 앱이 서비스 앱이라면 실행되지 않기 때문에 알림 기능이 동작하지 않는 문제가 있다.

CleanMaster[12]는 안드로이드 스마트폰에서 효율적인 메모리 관리를 위해 가장 많이 사용되는 앱이다. 이 앱에서 메모리 정리 기능을 실행하면 서비스 A, B 및 캐시드 상태의 많은 앱을 강제 종료하여 가용 메모리를 확보한다. 하지만, 서비스 앱의 특성 때문에 메모리 정리 후 짧은 시간 내에 종료된 서비스 앱들이 자동으로 실행된다. 결국 이들 서비스 앱의 재실행은 CleanMaster의 실행 이전과 비슷한 수준의 메모리 부족을 다시 야기할 수 있다.

Ⅲ. 서비스 앱의 특성과 영향 분석 기법

본 논문은 기존 연구에서 이루어지지 않은, 서비스 앱이 소비하는 메모리가 시스템과 사용자 앱의 성능에 미치는 영향을 분석하기 위해 다음과 같은 측정 및 분석 기법을 소개한다. 첫째 사용자의 스마트폰에서 서비스를 포함한 백그라운드 앱의 특성과 메모리 사용량을 측정하는 기법을 구현한다. 둘째, 사용자의 스마트폰에서 분석된 서비스 앱의 정보를 기반으로 상용 스마트폰에서 서비스 앱의 개수 증가가 시스템의 성능에 어떠한 영향을 미치는지를 측정하는 기법을 제시한다.

3.1. 백그라운드 앱의 특성과 메모리 사용량 측정

사용자의 스마트폰에 설치된 앱의 정보(예로, 상태, 메모리 사용량 등)와 시스템의 부팅 정보를 수집하기 위해 루팅(rooting) 또는 안드로이드의 수정 없이 다양한 스마트폰에 탑재 가능한 측정 앱을 개발하였다. 이 측정 앱은 기존 연구[13]의 AppMemTracker 앱에 본 논문에서 소개되는 다양한 측정 기법을 추가하는 방식으로 개발되었다. 안드로이드를 수정하지 않은 이유는 수정된 안드로이드를 구글 레퍼런스 폰 외의 다른 상용 스마트폰에 설치할 수 없기 때문이다. 측정 앱을 사용자의 스마트폰에 설치만 하면 사용자의 스마트폰에 설치된 앱의 정보를 획득한다.

측정 앱은 1개의 액티비티와 3개의 서비스로 구성된다. 3개의 서비스는 백그라운드로 실행하면서 다음과 같은 정보를 수집한다. 첫째, 사용자가 설치한 앱의 개수와 제조사, 통신사의 기본 탑재 앱의 개수를 조사한다. 둘째, 다양한 종류의 백그라운드 앱들이 소비하는 메모리 사용량을 수집한다. 셋째, 종료된 백그라운드 앱들이 재실행될 때까지 걸리는 시간을 분석한다. 수집된 정보는 엑셀 파일 형식으로 저장장치에 저장되고, 사용자의 스마트폰에서 주기적으로 원격 서버로 전송된다. 마지막으로 액티비티는 3개의 서비스가 측정한 정보를 사용자에게 보여주는 역할을 담당한다.

스마트폰의 부팅 시 자동으로 실행된 앱의 개수와 이들 앱의 상태를 분석하는 기법을 구현하였다. 시스템 부팅이 완료되면 ACTION_BOOT_COMPLETED 브로드캐스트가 방송된다. 측정 앱은 이 시점에 실행 중인 앱의 목록을 획득한다. 이 목록은 액티비티와 앱의 생명주기를 관리하는 액티비티 매니저(activity manager)를 통해 획득된다. 액티비티 매니저가 제공하는 getRunningAppProcesses 함수를 호출하면 실행 중인 앱의 목록을 얻을 수 있다. 아울러, 앱의 목록에 포함된 앱들의 상태를 확인하기 위해 각 앱의 pid(process id)를 획득한다. 이 pid에 대응하는 /proc/[pid]/oom_adj 파일에서 해당 앱의 종료 우선순위 값을 획득한다. 종료 우선순위 값에서 앱의 상태를 확인하기 위해 표 1에 기술된 앱의 상태와 종료 우선순위 값의 관계를 사용한다.

실행 중인 앱의 목록에 포함된 앱들의 pid를 사용하여 이들 앱이 사용하는 메모리 용량을 획득한다. 액티비티 매니저가 제공하는 getProfessMemoryInfo 함수를 호출하고, 인자로 실행 중인 앱들의 pid을 전달한다. 이 함수는 이들 pid를 가지는 앱들의 메모리 사용 정보를 포함하는 MemInfo 클래스의 객체를 반환한다. 측정 앱은 이 객체가 제공하는 getTotalPss 함수를 호출함으로써 앱의 메모리 사용량을 획득한다.

부팅 후 자동 실행된 앱이 사용자가 설치한 앱인지 제조사, 통신사의 기본 탑재 앱인지 구분하는 기법을 구현한다. 이 구현은 앱의 설치, 삭제 및 모든 설치된 앱의 정보를 관리하는 패키지 매니저(package manager)를 통해 이루어진다. PackageInfo 클래스는 설치된 앱의 정보(예로, 앱 이름, 설치 시간, 권한)를 가진다. 우선

패키지 매니저의 getInstalledPackages 함수를 호출하여 설치된 모든 앱의 PackageInfo 객체를 획득한다. 이들 객체 중에, 실행 중인 앱의 목록에 포함된 앱의 패키지 이름을 가지는 PackageInfo 객체들을 획득한다. 이 객체의 ApplicationInfo 값이 FLAG_SYSTEM 또는 FLAG_UPDATED_SYSTEM_APP이면 기본 탑재 앱이고, 그렇지 않으면 사용자가 설치한 앱이다.

실행 중인 앱을 모두 종료한 후, 종료 전 전체 앱의 개수 대비 70%, 80%, 90%의 앱이 재실행될 때까지 걸리는 시간을 측정한다. 이 실험 목적은 CleanMaster와 같은 기존 메모리 정리 앱이 안드로이드의 메모리 관리를 효과적으로 이용하는지를 확인하기 위함이다. 모든 앱들을 종료하기 위해 앱의 목록에 포함된 앱들의 pid를 획득하고, 이들 pid를 가지는 앱들에게 강제 종료 시그널인 SIGNAL_KILL을 보낸다. 그 후 1초마다 재실행되는 앱이 있는지를 검사한다. 재실행된 앱의 개수가 종료 전의 전체 앱 개수 대비 70%, 80%, 90%에 이르면 종료 시점부터 걸린 시간을 측정한다.

3.2. 서비스 앱이 시스템 성능에 미치는 영향 측정

스마트폰과 앱을 사용할 때의 응답 시간에 서비스 앱이 미치는 영향을 분석하는 기법을 구현한다. 3.1절의 측정 앱으로 확인된, 실험 대상자들의 스마트폰 사양과 비슷한 상용 스마트폰에 실험 환경을 구축하였다. 이 실험 환경에서 서비스 앱의 개수별 사용자 앱의 시작 시간과 설치 시간, 시스템 부팅 시간을 분석하였다. 또한 실행 중인 앱들의 상태, CPU 및 메모리 사용량을 측정하여 이들과 사용자 앱의 시작 시간, 설치 시간의 연관성을 분석한다.

시스템 부팅 시간을 측정하기 위해 부팅할 때 발생하는 ACTION_BOOT_COMPLETED 브로드캐스트를 사용한다. 이 브로드캐스트가 방송되면 수신을 원하는 앱이 실행되고, 앱에 등록된 리시버인 onReceive 함수가 호출된다. 이 브로드캐스트를 원하는 앱이 한 개 이상이기 때문에 이들 앱의 리시버는 동시에 호출되지 않고, 순차적으로 처리된다. 부팅 시간은 부팅 시작부터 이들 앱이 순차적인으로 실행된 후 해당 브로드캐스트를 마지막으로 수신할 때까지 걸리는 시간이다. 이때 측정 앱이 이 브로드캐스트를 마지막으로 수신하고, 수신 시 호출되는 리시버에서 부팅 시간을 획득한다. 측정 앱의 리시버가 마지막에 호출될 수 있도록 리시버의 호출 순서를 결정하는 우선순위를 가장 작은 값으로 설정하였다.

사용자 앱의 시작 시간을 측정하기 위해 안드로이드에서 지원되는 접근성(accessibility)[14]을 사용한다. 화면의 변화가 생길 때마다 onAccessibilityEvent 함수가 호출되며, 이 함수의 인자로 AccessibilityEvent 클래스의 객체를 전달된다. 이 클래스는 해당 화면을 가지는 앱의 정보, 화면 변화의 완료 시간, 화면 변화의 상태 등의 정보를 가진다. 앱 실행을 위해 홈 화면의 앱 아이콘을 누른 시간과 앱이 실행되어 화면 변화가 완료되는 시간의 차이를 앱 시작 시간으로 정의한다. 이때 화면 변화의 완료 시간은 AccessibilityEvent 클래스의 getEventTime 함수를 통해 획득된다.

앱의 설치 시간을 측정하기 위해 패키지 매니저를 사용한다. 앱의 설치가 완료되면 다음과 같은 동작을 하는 측정 앱을 실행한다. 패키지 매니저가 제공하는 getPackageInfo 함수를 호출하고, 인자로 설치된 앱의 이름을 전달한다. 이 함수의 호출로 반환된 PackageInfo 객체에 있는 firstInstallTime 값을 획득한다. 이 값은 설치된 시간을 나타낸다. 결국 사용자가 구글 플레이에서 앱 설치 버튼을 누른 시간과 설치된 시간의 차이로 앱의 설치 시간을 측정한다.

Ⅳ. 실험

4.1. 백그라운드 앱의 특성과 메모리 사용량 분석

측정 앱을 15명의 컴퓨터 전공 학생의 스마트폰에 설치하여, 설치된 앱의 개수, 부팅 후 자동 실행된 백그라운드 앱의 개수, 이들 앱의 상태별 개수와 메모리 사용량, 이들 백그라운드 앱을 강제 종료한 후에 주어진 시간 내에 재실행되는 앱의 개수 등을 측정하였다. 이들 사용자가 사용한 스마트폰의 기기 명, DRAM 용량, 탑재된 안드로이드 버전 정보는 표 2에 나와 있다. 이들 중 2G DRAM을 탑재한 스마트폰이 10명으로 가장 많았고, 안드로이드 Lollipop 5.0.X 버전을 탑재한 스마트폰이 8명으로 가장 많았다.

그림 1에서 스마트폰에 설치된 전체 앱들 중에 부팅 시 자동으로 실행된 앱의 개수를 확인할 수 있다. 설치된 앱은 사용자가 직접 설치한 앱과 구글, 제조사, 통신사의 기본 탑재 앱을 포함한다. 모든 사용자에서 전체 설치된 앱의 70% 이상이 기본 탑재 앱이고, user2에서

90%의 앱이 기본 탑재 앱이다. 설치된 앱 중 15~23%의 앱이 시스템 부팅 시 백그라운드로 자동 실행된다. 특히 5명의 사용자(user1~5)에서 80개 이상의 많은 앱들이 자동 실행된다. 자동 실행 이유는 시스템 부팅 시 발생하는 ACTION_BOOT_COMPLETED 브로트캐스트를 처리할 리시버를 가진 앱들이 실행되기 때문이다. 자동 실행되는 앱의 개수가 부팅 완료 시간에 영향을 줄 수 있다.

Table. 2 Specification of users' smartphones

User No.	Device	DRAM size(GB)	Android version
1	Galaxy S5	3	5.0.1
2	Galaxy S5	3	5.0.1
3	Galaxy Alpha	2	5.0.2
4	Galaxy Alpha	2	5.0.2
5	Galaxy Alpha	2	5.0.2
6	Galaxy S4	2	5.0.1
7	Galaxy Note2	2	4.4.2
8	Galaxy S3(SHW-M440S)	1	4.3
9	Galaxy S3(SHV-E210S)	2	4.4.4
10	Galaxy Alpha	2	5.0.2
11	Galaxy S4	2	4.4.2
12	LG G2	2	5.0.1
13	VEGA Iron2	3	4.4.2
14	LG G3	3	4.4.2
15	Galaxy Note3	2	4.4.2

흥미로운 결과로는, 일부 사용자(user6, 8, 11, 12, 13, 14)를 제외한 모든 사용자에서 직접 설치한 앱보다 많은 개수의 앱이 자동으로 실행되었다. 특히 user2, 4, 7, 15에서 사용자가 직접 설치한 앱 개수보다 43개, 27개, 24개, 24개의 더 많은 앱이 실행되었다. 이는 사용자가 직접 설치한 앱 중 일부 앱뿐만 아니라 기본 탑재 앱 중에 일부가 실행되었기 때문이다. 결국 사용자가 설치한 앱뿐만 아니라 기본 탑재 앱이 부팅 시간에 영향을 미칠 수 있음을 확인하였다.

그림 2는 그림 1에서 자동 실행된 앱의 상태를 보여준다. 전체 사용자에서 서비스 앱의 비율은 자동 실행된 앱의 28~65%이며, 개수는 18~37개에 이른다. 이때 자동 실행된 앱이 다른 상태를 가지는 이유는 부팅 시 ACTION_BOOT_COMPLETED 브로트캐스트를 받는 앱의 리시버가 서비스를 실행하면 서비스 A 또는 B 상태가 되며, 그렇지 않으면 앱의 동작에 따른 상태를 가지기 때문이다. 실행 중인 서비스 앱의 개수는 설치된 앱의 개수에 비례하지 않고, 앱 개발자가 리시버에서 서비스 컴포넌트를 실행하도록 구현했느냐에 따라 결정된다.

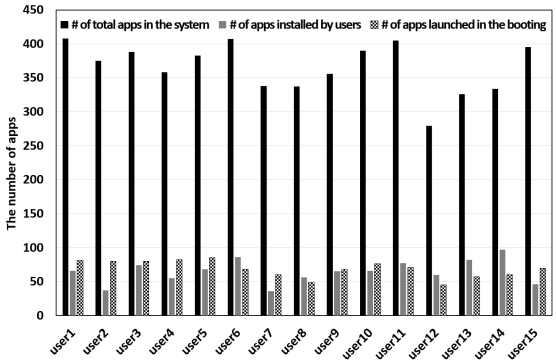


Fig. 1 The numbers of installed apps in smartphones and automatically launched apps in the booting

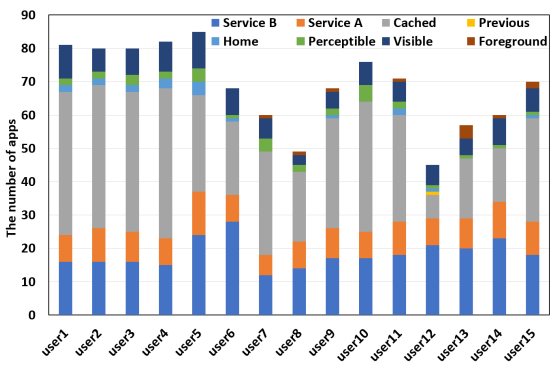


Fig. 2 States of automatically launched apps in the booting

그림 2의 6명 사용자(user5, 6, 12, 13, 14)에서 서비스 앱이 캐시드 앱보다 많이 실행되었다. 이들 중 user5를 제외한 사용자에서 서비스 앱이 50%이상 차지한다. 이렇게 많은 서비스 앱으로 인해 메모리가 부족해진 상황에서 사용자가 실행한 앱 i가 캐시드 상태로 전환되면, LMK에 의해 종료 우선순위가 서비스 앱보다 높은 앱 i가 강제 종료될 수 있다. 결국 user5, 6, 12, 13, 14가 작은 메모리 용량을 가지는 스마트폰이라면, 사용자가 실행시킨 앱들이 강제 종료되는 현상이 빈번히 발생할 수 있다.

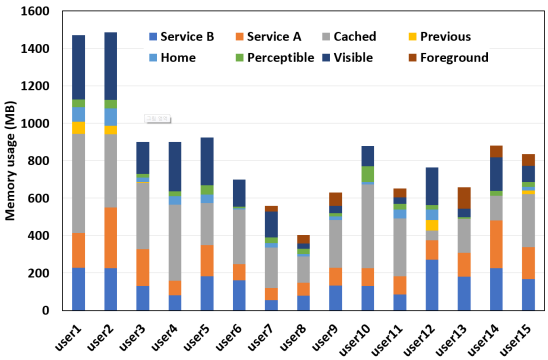


Fig. 3 The amounts of memory usage of automatically launched apps in the booting

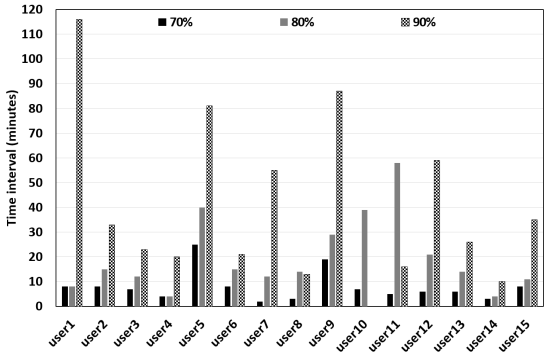


Fig. 4 Time intervals between forcibly termination and re-launching of apps

그림 3은 그림 2에서 부팅 시 자동 실행된 앱들이 차지하는 메모리 사용량을 나타낸다. user3, 4에서 실행된 앱의 개수는 user1, 2와 비슷하지만 메모리 사용량이 적다. 이는 앱 개수가 아닌 각 앱의 메모리 사용 패턴이 메모리 사용량에 영향을 주기 때문이다. 하지만, 실행된 앱 개수가 현저히 작은 user7, 8, 12, 13의 경우 메모리 사용량이 작다. 이를 통해 가능한 사용자가 부팅 시 자동 실행되는 앱의 개수를 줄일수록 메모리 사용량을 줄일 수 있을 것이라 기대할 수 있다.

서비스 앱들의 메모리 사용량이 전체 메모리의 18~21%를 차지하는 사용자(user4, 11)가 있는가 하면, 3명의 사용자(user2, 3, 5)에서 37%, 4명의 사용자(user12~15)에서 서비스 앱이 전체 메모리의 40% 이상을 점유하였다. 특히 user14에서 서비스 앱이 전체 메모리의 55%를 소비하기도 하였다. 한편, 기존 연구[3]에 따르면 90% 이상의 실험 대상자들이 설치된 앱 중 50% 이상을 사용하지 않고, 심지어 50% 이상의 대상자들이 설치된 앱 중 75%를 사용하지 않았다. 이 연구를 기반으로, 사용되지 않는 서비스 앱들이 많다면 4명의 사용자(user12~15)에서 불필요한 메모리 소비를 유발하여 실질적인 메모리 감소를 크게 유발할 수 있다고 판단할 수 있다.

그림 4는 실행 중인 앱을 모두 종료한 후, 종료 전 전체 앱의 개수 대비 70%, 80%, 90%의 앱이 재실행될 때까지 걸리는 시간을 나타낸다. 두 명(user5, 9)을 제외한 사용자에서 종료 후 2~8분 사이에 70%, 세 명(user5, 10, 11)을 제외한 사용자에서 30분 이내에 80%, 모든 사용자에서 116분 이내에 90%의 앱이 자동으로 재실행되었다. 스마트폰을 사용할 때 배터리 상태, 무선망 연결, Wi-Fi 연결, GPS 연결, 이어폰 연결, 화면 ON/OFF, SMS 또는 전화 수신 등의 동작이 시스템 상태를 변화시킨다. 이 변화에 대해 시스템은 상태 변화별로 정의된 브로드캐스트를 빈번히 발생시킨다. 시스템은 해당 브로드캐스트를 원하는 앱들 중에 종료된 앱의 리시버를 호출하여 종료된 앱을 재실행시킨다. 결국 실행 중인 앱의 강제 종료를 통해서 가용할 수 있는 메모리의 확보는 일시적이며, CleanMaster와 같은 기존 메모리 정리 앱은 가용 메모리 확보와 같은 시스템 성능 개선에 큰 도움을 주지는 못한다.

4.2. 서비스 앱이 시스템 성능에 미치는 영향

서비스 앱이 시스템 성능과 다른 앱의 동작에 미치는 영향을 확인하기 위해 서비스 앱의 개수에 따른 실험을 진행하였다. 실험 스마트폰은 Quad-Core 2.4GHz CPU, 2GB DRAM의 LG Nexus 5이고, 안드로이드 Lollipop 5.0.1을 탑재하였다. LG Nexus 5를 사용한 이유는 4.1절에서 분석된 사용자의 스마트폰이 2GB DRAM 메모리와 안드로이드 Lollipop 5.0.X를 가장 많이 탑재하였기에 이들 스마트폰과 비슷한 사양의 스마트폰에서 시스템 성능을 분석하기 위해서이다. Android debug bridge(adb)[15]의 meminfo 명령어를 통해 2GB RAM 중에 1.9GB 메모리 공간이 안드로이드 운영체제와 앱의 공간으로 사용됨을 확인하였다.

실험을 위해 구글 플레이 스토어에서 인기순위 1위부터 시작하여 낮은 순위의 앱들 대상으로 서비스를 가지는 50개의 앱을 선정하였다. 이들 선정된 50개의 서비스 앱에서 메모리 사용량의 크기가 큰 앱부터 10개씩

개수를 늘리면서 스마트폰에 설치 한 후 실행시킨다. 표 3은 10개씩 앱을 늘리면서 설치할 때 10개 단위의 앱의 평균 메모리 사용량을 나타내며, 순번은 앱 설치 순서를 의미한다. 설치한 서비스 앱의 개수에 따라 자동으로 실행되는 앱의 개수와 부팅 시간을 측정하였다.

Table. 3 Average sizes of installed service apps according to their installing sequences

Sequence of installed service apps		Average size of service apps(MB)
(1)	1~10	170
(2)	11~20	61
(3)	21~30	34
(4)	31~40	25
(5)	40~50	18

실험 스마트폰에서 adb의 memfree 명령어를 통해 다음과 같은 LMK 동작을 확인하였다. 가용 메모리 공간이 180MB으로 줄면, LMK가 실행되어 종료 우선순위가 15를 가지는 캐시드 앱들을 종료시킨다. 이 종료 우선순위의 캐시드 앱이 없고 가용 메모리가 144MB가 되면, LMK는 종료 우선순위가 9부터 14를 가지는 앱들을 강제 종료한다. 캐시드 앱이 더 이상 존재하지 않고 가용 메모리가 126MB일 때, 서비스 앱들을 강제 종료하기 시작한다. 그렇지만 LMK로 인해 종료된 서비스 앱들은 잠시 후 자동으로 재실행된다.

그림 5는 서비스 앱의 설치 개수별로 부팅 시 자동으로 실행된 앱의 개수와 상태별 앱의 개수를 나타낸다. 서비스 앱을 10개 설치할 때 자동으로 실행된 앱 중 약 27%가 캐시드 앱이며, 29%가 서비스 앱이다. 설치하는 서비스 앱의 개수가 20개, 30개로 증가하면 캐시드 앱은 각각 12%, 6%로 크게 감소하고, 서비스 앱 비율은 각각 46%, 58%로 크게 증가한다. 서비스 앱을 40개 설치할 때부터는 캐시드 앱이 존재하지 않고, 서비스 앱이 60% 이상 차지한다. 그 이유는 메모리 관리 정책으로 인해 메모리 부족 시 서비스 앱보다 종료 우선순위가 높은 캐시드 앱이 먼저 종료되기 때문이다. 마지막으로 서비스 A에 비해 서비스 B 앱의 개수가 증가하고 있다. 이는 서비스 A 앱 개수는 전체 서비스 앱 중에 특정 비율로 제한되며, 이 비율보다 서비스 앱이 많아지면 서비스 A앱이 서비스 B 상태로 변경되기 때문이다.

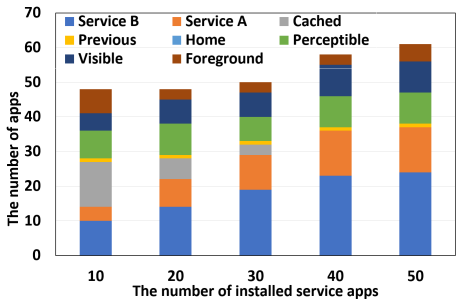


Fig. 5 States of automatically launched apps in the booting, according to the number of installed service apps

그림 5에서 설치된 서비스 앱의 개수가 10개부터 20개, 30개로 증가하지만, 전체 실행된 앱의 개수가 별로 증가하지 않는다. 설치된 서비스 앱의 개수를 늘리면 메모리가 부족하여 캐시드 앱이 강제 종료되어 캐시드 앱의 개수가 감소한다. 결국 설치된 서비스 앱의 개수는 증가하지만, 감소한 캐시드 앱의 개수 때문에 전체 실행된 앱의 개수는 거의 동일하다. 서비스 앱을 20개 설치할 때에 비해 30개 설치할 때에 강제 종료된 캐시드 앱의 개수가 작지만, 퍼셉터블 앱의 개수가 감소하여 전체 실행된 앱의 개수가 거의 증가하지 않는다.

서비스 앱을 40, 50개 설치할 때 실행된 앱의 개수가 증가한다. 서비스 A의 앱 개수 제한으로 40개 이전에 설치된 서비스 앱의 일부가 서비스 B 상태로 전환된다. 이때 서비스 앱을 추가하면 캐시드 앱이 없기 때문에 서비스 B의 앱을 강제 종료한다. 이 강제 종료된 서비스 앱의 메모리 사용량은 표 3의 (1)~(3)에 해당되지만, 40개, 50개에 설치되는 서비스 앱의 메모리 사용량은 표 3의 (4), (5)처럼 매우 적다. 따라서 표 3의 (1)~(3)에 설치된 서비스 앱을 강제 종료하여 확보된 큰 가용 메모리 공간에 메모리 사용량이 적은 서비스 앱을 40, 50개로 증가하여 설치하기 때문에 더 많은 앱이 실행된다.

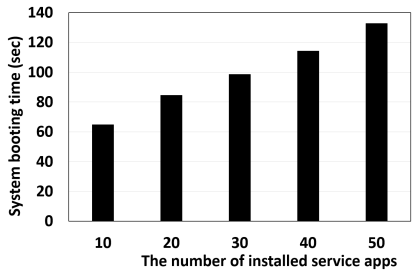
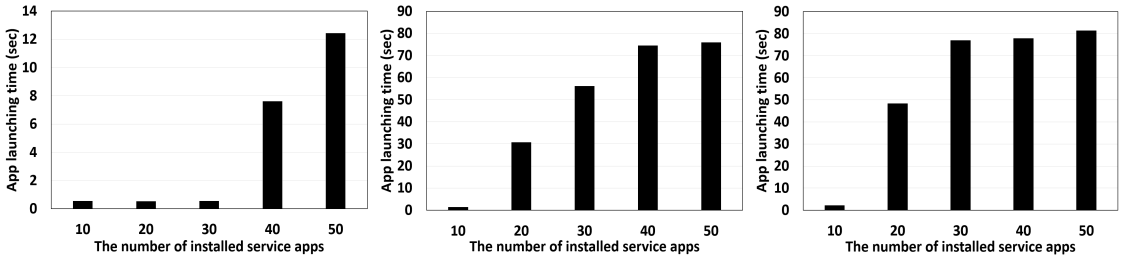


Fig. 6 System booting times according to the number of installed service apps



(A) App launching times of pattern 1 (B) App launching times of pattern 2 (C) App launching times of pattern 3

Fig. 7 App launching times according to the number of installed service apps

그림 6은 설치된 서비스 앱의 개수별 시스템 부팅 시간을 보여준다. 설치된 서비스 앱이 많을수록 부팅 시간이 비례적으로 증가한다. 그 이유는 다음과 같다. 첫째, 부팅 과정에서 서비스 앱들의 리시버가 호출된 후, 마지막으로 측정 앱의 리시버가 호출된다. 따라서 서비스 앱의 개수가 증가할수록 그만큼 측정 앱이 브로드캐스트를 수신하는 시간이 늦어진다. 둘째, 서비스 앱 개수가 늘어날수록 메모리 부족이 심해져서 CPU 부하를 야기하는 LMK의 실행 횟수가 증가하기 때문이다. 그림 5에서 20개의 서비스 앱 개수부터 메모리 부족으로 LMK가 빈번히 실행되어 캐시드 앱들을 강제 종료한다. 이런 빈번한 실행은 부팅 과정에서 큰 CPU 부하를 야기하여 브로드캐스트를 수신할 서비스 앱들의 실행을 간섭하고, 이 간섭은 브로드캐스트를 수신할 전체 앱들의 실행을 지연시킨다.

본 실험은 최근 출시된 스마트폰보다 작은 메모리를 탑재한 LG Nexus 5에서 서비스 앱이 시스템 성능에 미치는 영향을 분석하였다. 최근 갤럭시 S7, S8과 같은 스마트폰은 4GB DRAM과 같은 대용량 메모리를 탑재한다. 이런 스마트폰에서 메모리 부족이나 LMK 실행에 따른 CPU 부하가 상대적으로 덜할 수도 있다. 하지만, 새롭게 배포되는 앱의 크기가 해마다 증가하고[16], 동일 앱이 업데이트될 때마다 앱 크기가 증가하고 있다[17]. 이런 큰 앱이 실행되면 메모리 소비가 그만큼 크게 증가한다. 따라서 최신 스마트폰이 대용량 메모리를 가지더라도 큰 서비스 앱들이 많이 실행되면 가용 메모리 공간이 여전히 부족할 수 있다. 이로 인해 LMK가 빈번히 실행되어 CPU 부하가 증가할 수 있다. 결국 최신 스마트폰에서도 본 논문의 분석 결과와 비슷한 시스템 성능 저하 문제가 발생할 것으로 예상한다.

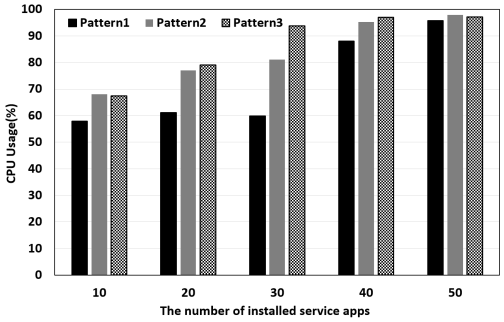
4.3. 서비스 앱이 사용자 앱의 실행에 미치는 영향

앱의 시작 시간을 측정하기 위해 구글 플레이 스토어의 소셜, 커뮤니케이션, 게임 카테고리에서 인기 있는 페이스북, 크롬, 카카오톡, 모두의 마블을 선정하였다. 앱의 시작 시간은 사용자가 홈 화면의 앱 아이콘을 누른 후 해당 앱의 UI가 뜨는데 걸리는 시간이다. 4개의 앱을 다음과 같은 3가지 실행 패턴으로 반복적으로 재실행하였다. 패턴1은 카카오톡, 크롬의 순서로 2개 앱을, 패턴2는 모두의 마블, 크롬, 카카오톡의 순서로 3개 앱을, 패턴3은 카카오톡, 크롬, 페이스북, 모두의 마블의 순서로 4개의 앱을 순차적으로 실행하는 패턴이다. 패턴별 반복 횟수는 20번이며, 각 반복 실행에 대한 평균 시작 시간을 해당 패턴의 앱 시작 시간으로 정의한다.

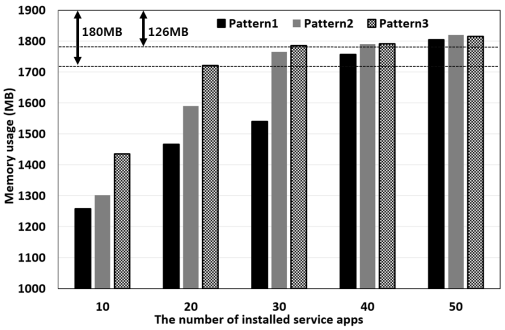
그림 7은 3개 패턴의 앱 시작 시간을 나타낸다. 패턴1에서 30개의 서비스 앱 개수까지 앱 시작 시간의 변화가 거의 없다. 그 이유는 패턴1의 CPU와 메모리 사용량을 나타내는 그림 8에서 확인할 수 있다. 비록 패턴1의 실행으로 메모리 사용량이 증가하지만 가용 메모리 공간이 LMK가 실행되는 가용 메모리 용량(180MB)보다 많아서 LMK가 거의 실행되지 않는다. 결국 LMK가 실행되지 않기 때문에 패턴1의 실행을 방해하는 CPU 부하가 발생하지 않으며, 패턴1의 앱들도 강제 종료되지 않는다. 이로 인해 패턴1의 앱 시작 시간이 30개의 서비스 앱까지 거의 변화하지 않는다.

하지만, 40개의 서비스 앱 개수에서 패턴1의 앱 시작 시간이 크게 증가하였다. 즉 이 앱 시작 시간이 30개에 비해 13배 정도 증가하였다. 이런 증가 이유는 다음과 같다. 첫째, 그림 5처럼 캐시드 앱이 존재하지 않는 상태에서 그림 8(B)의 가용 메모리 공간이 LMK가 서비스 앱을 강제 종료하는 가용 메모리 공간의 임계값(126MB)에 가까이 이르렀다. 이로 인해 LMK는 패턴1의 앱들뿐만 아니라 여러 서비스 앱들을 종료시키고 시

스템은 종료된 이들 서비스 앱을 재실행한다. 이런 종료 및 재실행은 CPU 부하를 크게 증가시키고, 패턴1의 앱들을 실행할 때 CPU 부하를 더욱 가중시킨다. 이런 동작은 40개의 서비스 앱 개수에서 CPU 사용량을 88%까지 이르게 한다. 둘째, 이렇게 CPU 사용량이 높아지면 강제 종료된 패턴1의 앱들의 실행에 CPU를 많이 할애할 수 없기 때문이다.



(A) The amounts of CPU usage



(B) The amounts of memory usage

Fig. 8 The amounts of CPU and memory usages according to app usage patterns

그림 7의 20개 서비스 앱 개수에서 패턴2, 3의 앱 시작 시간이 10개에 비해 각각 21배, 22배 정도 증가하였다. 그 이유는 그림 8(B)에서 패턴2, 3의 메모리 사용량이 다음과 같은 이유로 패턴1에 비해 크게 증가했기 때문이다. 첫째, 패턴2, 3에서 실행하는 앱 개수가 패턴1의 앱 개수보다 많다. 둘째, 패턴2, 3에서 메모리 사용량이 큰 게임 앱인 모두의 마블이 추가로 실행되었다. 패턴1보다 큰 메모리 사용량은 메모리 부족 현상을 더욱 심각하게 야기하여 LMK의 실행 횟수, 서비스 앱의 종료와 재실행 횟수를 증가시킨다. 이런 동작은 불필요한 CPU 소비를 야기함으로써 그림 8(A)처럼 패턴1에 비해 CPU 사용량을 증가시키는 원인이 된다. 또한 반복된 서비스 앱의 종료와 재실행은 CPU가 사용자의 앱을 실행할 때 방해함으로써 패턴2, 3의 앱 시작 시간을 증가시킨다.

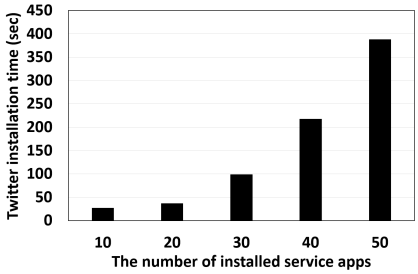


Fig. 9 Twitter installation times according to the number of installed service apps

40개부터 패턴2의 앱 시작 시간이 거의 증가하지 않는다. 그림 5에서 40개의 서비스 앱 개수부터 캐시드 앱이 존재하지 않고, 그림 8(B)에서 가용 메모리 공간이 서비스 앱들이 강제 종료되는 임계값(126MB)보다 감소하였다. 이로 인해 LMK가 빈번히 실행되어 서비스 앱들을 종료하고 시스템이 종료된 서비스 앱을 재실행하는 동작을 반복한다. 이런 동작 때문에 40개의 서비스 앱 개수에서 최대 CPU 사용량이 96%이상까지 증가하고, 대부분 CPU 사용은 패턴2의 실행이 아닌 서비스 앱의 종료와 재실행에 대부분 소비될 수밖에 없다. 결국 최대 CPU 사용량에서 패턴2의 실행에 CPU를 매우 적게 할애하기 때문에 설치된 서비스 앱이 40개보다 증가하더라도 패턴2의 앱 시작 시간은 더 이상 증가하지 않는다. 패턴3에서는 30개부터 거의 앱 시작 시간이 최대치에 이르렀다. 이는 패턴2와 달리 30개부터 메모리 사용량과 CPU 사용량이 최대에 이르렀기 때문이다.

4.4. 서비스 앱이 앱 설치에 미치는 영향

설치된 서비스 앱 개수가 새로운 앱 설치에 미치는 영향을 확인하기 위해 설치된 서비스 앱 개수별 앱 설치 시간을 측정하였다. 설치할 앱은 가장 인기 있는 소셜 네트워크 서비스(SNS) 앱인 트위터(twitter)이다. 이 앱의 크기는 19.9MB이다. 구글 플레이 스토어에서 “설치” 버튼의 클릭 시점과 설치 완료 시점의 차이로 앱 설치 시간을 측정하였다.

그림 9는 서비스 앱 개수에 따른 앱 설치 시간을 나타낸다. 서비스 앱 개수가 30개에서부터 앱 설치 시간이 지수적으로 증가한다. 특히 서비스 앱 개수가 50개일 때 앱 설치 시간이 6분 이상 걸림을 확인할 수 있다. 그 이유는 서비스 앱이 많아질수록 메모리가 부족하여 LMK가 반복적으로 실행되어 CPU 부하가 크게 증가하기 때문이다. 결국 사용자가 앱 설치 시 체감할 만큼 오랫동안 설치 과정이 지속됨을 알 수 있다. 앱 설치가 오래 걸리는 데는 네트워크 지연 등 다양한 원인이 있을 수 있지만, 사용자가 설치한 서비스 앱 개수의 증가가 앱 설치 시간을 증가시킨 원인 중의 한 가지임을 실험을 통해 확인할 수 있다.

Ⅴ. 결 론

본 논문은 안드로이드 스마트폰에서 메모리 부족으로 강제로 종료된 서비스 앱이 사용자의 실행 요청이 없더라도 잠시 후 자동으로 재실행되어 메모리를 불필요하게 소비하는 문제를 제시하였다. 이런 문제가 스마트폰의 성능에 미치는 영향을 분석하기 위한 측정 앱을 개발하였다. 이 측정 앱을 15명 사용자의 스마트폰에 탑재하고 이들 스마트폰에서 서비스 앱의 개수와 메모리 사용량, 앱의 강제 종료와 재시작 간의 시간을 조사하였다. 서비스 앱의 비율은 실행된 앱의 28~65%이며, 개수는 18~37개에 이른다. 약 50%의 사용자에서 서비스 앱이 전체 메모리의 37% 이상의 큰 메모리를 점유하였다. 또한 가용 메모리 확보를 위해 강제 종료된 앱 중 90%의 앱이 116분 이내에 자동으로 재실행되었다. 따라서 앱의 강제 종료를 통한 가용 메모리의 확보는 일시적임을 확인하였다.

서비스 앱이 스마트폰의 성능에 미치는 영향을 분석한 결과, 설치하는 서비스 앱의 개수가 증가할수록 메모리 공간이 극도로 고갈되었고, 사용자가 자주 사용하는 앱을 강제 종료시키기도 하였다. 아울러 극도로 메모리가 부족한 상황에서 사용자가 실행하는 앱의 개수가 증가할수록 LMK가 더욱 빈번히 실행되어 서비스 앱들을 종료하고 시스템이 종료된 서비스 앱을 재실행하는 동작을 반복한다. 이런 동작은 최대로 CPU를 사용하게 하여 사용자 앱을 다시 실행할 때 시작 시간을 종료되지 않은 앱 실행에 비해 최대 22배까지 증가시켰다. 또한 부팅 시간이 서비스 앱의 개수에 비례하고, 앱 설치 시간이 지수적으로 증가함을 확인하였다.

서비스 앱으로 발생하는 시스템과 앱 성능 저하를 막기 위해, 메모리 부족 시 종료된 서비스 앱을 캐시드 앱처럼 재실행하지 않는다. 서비스 앱이 다시 실행되지 않기 때문에 CPU와 메모리 소비를 줄일 수 있다. 하지만 종료된 서비스 앱이 알림을 받을 수 없다. 이 문제를 해결하기 위해 종료된 서비스 앱을 필요 시 재실행하는 방법을 구현할 수 있다. 종료되는 앱이 서비스를 포함하는 앱, 즉 서비스 앱이라면 이 서비스의 정보(ServiceRecord 객체)를 획득한다. 이 서비스의 정보를 앱 ID인 패키지 이름과 함께 새롭게 정의된 테이블 자료 구조에 저장한다. 이후 서비스 앱으로 요청이 발생하면 안드로이드는 해당 서비스 앱이 종료되었는지 확인한다. 만일 종료된 서비스 앱이면 패키지 이름으로 위에 언급된 테이블에서 서비스 정보를 검색한다. 검색된 서비스 정보를 인자로 하여 performServiceRestartLocked 함수를 호출한다. 이 해결책을 Lollipop 5.0.1을 탑재한 Nexus 5에서 구현하여 종료된 서비스 앱이 재실행됨을 확인하였다. 이 기법을 기반으로 서비스 앱이 종료되더라도 알림을 처리하면서, 서비스 앱의 불필요한 메모리 소비를 줄이기 위한 메모리 관리 기법을 연구할 계획이다.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03029374 and No. 2011-0013781). The present Research has been conducted by the Research Grant of Kwangwoon University in 2018.

REFERENCES

[1] Service | android developers [Internet]. Available: https://developer.android.com/guide/components/services.html.

[2] M. Martins, J. Cappos, and R. Fonseca, “Selectively taming background android apps to improve battery lifetime,” in *Proceedings of 2015 USENIX Annual Technical Conference*, Santa Clara, pp.563-575, 2015.

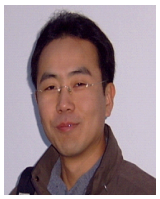
[3] I. Singh, S. V. Krishnamurthy, and H. V. Madhyastha, "ZapDroid: Managing infrequently used applications on smartphones," *IEEE Transactions on Mobile Computing*, vol. 16, no 5, pp. 1475-1489, May 2017.
[4] R. Prodduturi and D. B. Phatak, "Effective handling of low memory scenarios in Android using logs," Master thesis, Indian Institute of Technology, Bombay, 2013.
[5] W. Song, Y. Kim, H. Kim, J. Lim, and J. Kim, "Personalized optimization for android smartphones," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2s, pp. 1-25, Jan. 2014.
[6] S. Kim, J. Jeong, J. Kim, and S. Maeng, "SmartLMK: A memory reclamation scheme for improving user-perceived app launch time," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 3, pp. 1539-9087, May 2016.
[7] Application fundamentals | android developers [Internet]. Available: https://developer.android.com/guide/components/fundamentals.html.
[8] Processes and application life | android developers [Internet]. Available: https://developer.android.com/guide/topics/processes/process-lifecycle.html.
[9] Y. Oh and W. Ahn, "Classifying and analyzing process states in the android operating system," in *Proceedings of 2016 Spring KIPS Conference*, Seoul, vol. 23, no. 1, pp. 179-182, 2016.
[10] X. Chen, A. Jindal, N. Ding, Y. Hu, M. Gupta, and R. Vannithamby, "Smartphone background activities in the wild: origin, energy drain, and optimization," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, Paris, pp. 40-52, 2015.
[11] Samsung newsroom [Internet]. Available: http://goo.gl/qaRgEC.
[12] Clean master-space cleaner & antivirus [Internet]. Available: https://play.google.com/store/apps/details?id=com.cleanmaster.mguard&hl=en.
[13] Y. Oh and W. Ahn, "Monitoring memory shortage situation and booting characteristics in the android operating system," in *Proceedings of 2016 Spring KIPS Conference*, Seoul, vol. 23, no. 1, pp. 175-178, 2016.
[14] AccessibilityService | android developers [Internet]. Available https://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html.
[15] Android debug bridge | android developers [Internet]. Available: https://developer.android.com/studio/command-line/adb.html.
[16] Shrinking APKs, growing installs [Internet]. Available: https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2.
[17] D. Li, B. Guo, Y.Shen, J. Li, and Y. Huang, "The evolution of open-source mobile applications: An empirical study," *Journal of Software: Evaluation and Process*, vol. 29, no. 7, pp. 1-18, July 2017.

안우현(Woo Hyun Ahn)

1996년 경북대학교 전자공학과 (학사)
1998년 KAIST 전기 및 전자공학과 (석사)
2003년 KAIST 전자전산학과 (박사)
2003년~2005년 삼성전자 책임연구원
2006년~현재 광운대학교 소프트웨어학부 교수
※관심분야 : 운영체제, 임베디드시스템, 시스템 보안

오윤석(Yunseok Oh)

2015년 광운대학교 컴퓨터소프트웨어학과 (학사)
2017년 광운대학교 컴퓨터소프트웨어학과 (석사)
2017년~ (주) 네이버 Apollo 클라이언트팀 연구원
※관심분야 : 운영체제, 시스템소프트웨어, 모바일 플랫폼



오재원(Jaewon Oh)

2004년 서울대학교 계산통계학과(학사, 석사, 박사)
2004년~2007년 삼성전자 책임연구원
2007년~현재 가톨릭대학교 컴퓨터정보공학부 부교수
※관심분야 : Software Engineering, Web Engineering, System Software