메모리 부족상황에서 앱 실행시간에 관한 연구

성민철 ○ 권세준 정진규

성균관대학교

smc149@skku.edu, sejun000@csl.skku.edu, jinkyu@skku.edu

Study of Applications Launching Time on Low-Memory Condition in Mobile Systems

Mincheol Sung○ Sejun Kwon Jinkyu Jeong

Sungkyunkwan University

요 약

안드로이드에서는 사용자가 앱(application)을 종료할때, 앱을 메모리에 caching하여 빠른 재실행을 보장한다. 메모리 부족상황에서 새로운 앱을 실행하면, 커널은 메모리 회수하기위해 Low Memory Killer(LMK)를 작동시켜 caching된 프로세스를 강제종료한다. 이는 앱 실행에 오버헤드로 작용하게 된다. 본 논문은 메모리부족 상황에서 앱을 실행할 때 오버헤드로 작용하는 LMK의 process kill 과정에 대해 분석한다. 먼저 메모리 부족상황에서 앱의 실행시간의 변화를 측정하였고 실제 LMK에 의해 프로세스가 강제종료 되는 과정을 Ftrace function tracer을 이용하여 분석하였다. 분석 결과 커널은 한번에 여러개의 프로세스를 강제종료 하고, 때문에 LMK의 오버헤드도 크게 나타났다.

1. 서론

음악, 게임과 같은 엔터테인먼트 분야뿐만 아니라 뉴스, 메신저, 소셜 네트워킹 서비스 등 다방면에 활용되는 스마트폰은 최근에 들어 데스크탑 PC보다 현대인의 일상에 밀접하고 정확한 의미에서의 'personal computer'가 되었다. 스마트폰은 다양한 운영체제를 기반으로 출시되고 있으며, 그 중 안드로이드는 전 세계 스마트폰 운영체제의 82.8%[1]를 차지할 정도로 높은 점유율을 갖고 있다.

안드로이드는 사용자가 홈버튼이나 뒤로가기 버튼을 사용하여 기존의 앱으로부터 빠져나올때 그 앱을 완전하게 종료하지 않는다. 나중에 사용자가 다시 그 앱을 실행시켰을때 빠른 실행속도를 보장해주기 위하여 앱을 일시정지 상태로 만든 뒤 메모리에 저장시킨다(app caching)[2]. 하지만 때로는 많은 앱을 캐싱(caching)하다보면 메모리가 꽉차게 되고, 새로운 앱을 실행시키기 위해선 메모리공간을 확보해야할 필요성이 있다.

이를 위해 안드로이드 커널모듈인 *Low Memory Killer*(LMK)[3]는 우선순위에 따라 메모리에 있는 캐싱된 앱을 강제종료시켜(process kill) 메모리를 회수한다. 하지만, 이 과정은 메모리 공간을 해제하는 것 이외에도 많은 일을 동반하기 때문에 어떤 앱의 실행시 메모리를 확보하는 데에 큰 오버헤드로 작용할 수 있다.

만약 LMK에 의해서 프로세스가 죽는 과정에서 메모리 공간을 해제하는 과정을 우선적으로 처리한다면, 실행되는 앱이 좀 더 빨리 메모리를 할당받을 수 있게 되고, 앱의 실행 속도를 향상 시킬 수 있을 것이다. 하지만 이러한 앱의 강제종료 과정의 변화는 커널의 정상적인 작동을 보장하기 위해 프로세스를 죽이는 각 과정들의 의존성(dependency)을 고려해야 한다. 예를들어 메모리 공간이 해제되기전에 저장되어야 할 데이터 또는 다른 프로세스들과의 공유하고 있는 것들을 고려하지 않는다면 데이터의 손실, 심지어는 커널 패닉을 일으킬 수 있기 때문이다.

본 논문에서는 안드로이드 환경에서 메모리 부족시 나타나는 앱의 구동 지연을 측정하고, LMK에 의해 프로세스가 kill되어 메모리 공간을 해제 하기까지의 과정을 분석하고, 소요되는 시간을 측정 및 분석하며, 이를 바탕으로 향후 연구 방향을 제시한다.

2. 배경 지식

2.1 앱 캐싱(App caching)

안드로이드와 같은 모바일 플랫폼에서는 앱 실행의 높은 응답성을 제공하는 수단으로 app caching을 제공한다. App caching은 유저가 앱으로 부터 빠져나왔을때 앱을 종료시키는 대신에 일시정지 상태로 메모리에 저장하는 것이다.[2, 6] Caching된 앱을 재실행시키는 것은 앱의 구동시 초기화 비용이 발생하지 않기 때문에 종료된 앱을 재실행하는 것보다 훨씬 빠르다. 하지만 app caching은 메모리를 차지하기 때문에 한정된 메모리공간에서 메모리부족을 야기할 수 있다.

2.2 Low Memory Killer

Low memory killer(LMK)는 이러한 메모리 부족시 캐싱된 앱을 강제종료시켜 메모리를 확보하는 도구이다. 기존의 리눅스 *OOM killer*(Out of Memory killer)는 캐싱된 앱과 시스템 서비스 프로세스를 구분하지 않고 프로세스를 선택/종료시키기 때문에 안드로이드와 같은 앱 캐싱 환경에서는 부적합하다. 반면, LMK는 안드로이드 프레임웍으로부터 캐싱된 앱과 이의 우선순위를 전달받아, 메모리 부족시 캐싱된 앱만을 종료시켜 안드로이드 환경에 더 적합하다.

캐싱된 앱의 우선순위는 ActivityManager[4]가 *adj*값을 통해

제공한다. 이는 화면 상태에 따른 우선순위, 앱의 최근 사용정도에 따른 우선순위를 반영한다. 이와 함께 미리 설정된 유휴 메모리(free memory)의 하한선(*minfree*)에 도달하면 캐싱된 앱을 선택하여 강제종료를 수행한다. *adj*값과 *minfree*값은 (0, 1024), (8, 4096)과 같은 조합을 이루고 있다. 예를들어, (8, 4096)의 경우는 메모리가 4096개의 페이지보다 작은 상황이 되면 *adj*값 8 이상의 앱을 강제종료 한다.[5]

3. 메모리 부족이 앱 구동에 미치는 영향 분석

3.1 실험환경

본 논문은 메모리 부족시 LMK로 인한 앱의 구동 지연을 측정하기 위해 Nexus 5를 활용하였다. Nexus 5[8]는 2.26 GHz quad-core Krait 400 CPU에 2GB의 메인메모리를 장착하였으며, 안드로이드 5.1.4_14, 리눅스 커널 3.4.0을 탑재하고 있다. Nexus 5에는 안드로이드 디버그 브릿지(Android Debug Bridge)[7]를 통해 접속하여 워크로드를 구동하고 시간 및 메모리 사용량을 측정하였다.

3.2 메모리 상황에 따른 앱 실행시간 변화

그림1은 cached app으로 인해 메모리가 부족한 상황에서 대상 앱들의 구동 시간을 측정한 결과이다. 시간측정은 adb logcat[9]의 ActivityManager Displayed 시간을 기준으로 하였고, 5개의 앱을 대상으로 하였다.

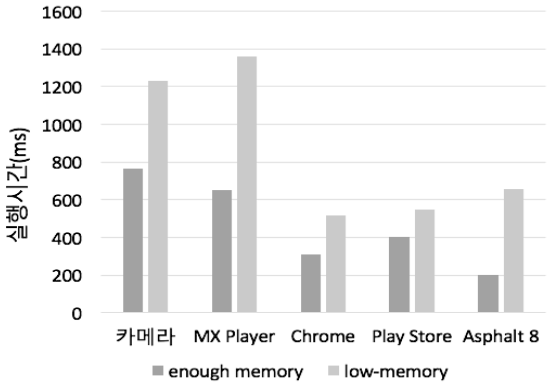

그림 1 대상 앱의 실행시간

먼저 앱이 실행하는데 필요한 메모리를 측정하였는데, 앱을 실행하기 전후 /proc/meminfo의 FreeMem 값 변화량을 계산하여 얻었다. 그리고 메모리가 부족한 상황과 충분한 상황에서의 앱 실행시간을 측정하였다. 메모리가 부족한 상황을 위해 balloon driver를 이용하여 인의적으로 시스템의 메모리를 늘리고 다른 기타 앱들을 캐싱시켜 여유 메모리 크기를 줄였다. 반대로 안드로이드의 부팅 직후의 상황을 여유 메모리가 충분한 상황으로 설정하였으며, 앱의 구동 시간은 adb logcat에서 ActivityManager의 Displayed 메세지가 출력된 시점을 앱이 실행된 것으로 설정하였다.

1. 안드로이드 기본 카메라앱은 74,932KB가 필요하였고, 메모리가 부족한 상황에서 앱의 실행시간은 그렇지 않은 상황보다 약 60.4%증가하였다.
2. 동영상 플레이어인 MX Player는 119,464KB의 메모리를 필요로하고 108.8% 느리게 실행되었다.
3. 안드로이드 기본 브라우저 Chrome은 103,900KB의 메모리를 필요로 하고, 67.6% 더 많은 실행시간이 측정됐다.
4. Play Store는 57,756KB의 메모리를 필요로 하고, 실행시간이 36.4% 증가 하였다.
5. 게임 앱인 Asphalt8은 502,748KB의 메모리를 필요로 하고 228.6% 많은 실행시간이 측정됐다.

Asphalt 8을 제외한 앱들은 LMK가 대략 2~4 개정도의 프로세스를 종료시켰다. 하지만 Asphalt 8의 경우 LMK는 17개의 프로세스를 종료시켰는데, 이는 다른 앱에 비해 많은 메모리를 필요로 하기 때문이다. 앱마다 필요한 메모리의 크기가 다른데, 많은 메모리를 필요로 하는 앱은 LMK에 의해 많은 메모리를 회수하기때문에 실행시간 변화가 크게 나타났다. 이는 섹션 6에서 자세히 설명한다.

3.3 앱 강제종료 과정 분석

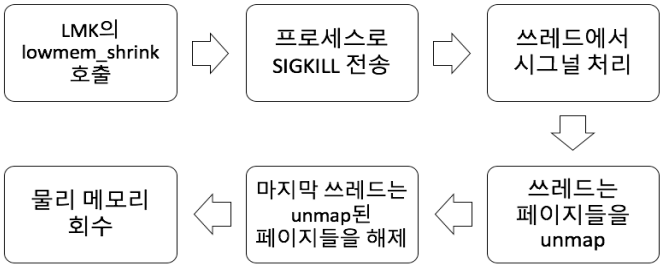

그림 2 LMK의 Process kill 처리과정

메모리가 부족한 상황에서 앱이 실행되어 메모리 할당이 필요하면, 커널은 기존의 메모리를 해제하는데, 이는 커널 스왑 데몬(kswapd)[10] 또는 실행되는 앱이 메모리 회수 과정에서 직접 LMK의 *lowmem_shrink* 함수를 호출하면서 시작된다. *lowmem_shrink* 함수는 종료시킬 앱이 선정되면, *send_sig* 함수를 통해 kill의 대상이되는 프로세스에게 SIGKILL의 시그널을 보낸다. 시그널은 *sigaddset* 함수에 의해 시그널 집합에 추가되고 *complete_signal* 함수는 시그널을 처리할 쓰레드를 찾고 *signal_wake_up* 함수를 통해 해당 쓰레드를 wake-up한다. 쓰레드는 wake-up 하면서, pending된 시그널을 처리한다.

signal_wake_up 함수는 *signal_wake_up_state* 함수를 통해 *kick_process* 함수를 호출하는데, 이 함수는 유저모드에 있는 쓰레드가 시그널 핸들러를 수행하도록 강제하기 위해 해당 쓰레드를 커널 모드로 진입시킨다.

Kill되는 쓰레드에서 *do_signal* 함수는 시그널을 처리하고, *do_exit* 함수를 호출한다. *do_exit* 함수는 쓰레드의 종료를 담당하는 함수이며 쓰레드가 가지고 있는 *mm_struct* 의 해제를 담당하는 *exit_mm* 함수를 호출한다.

쓰레드가 가지는 가상메모리는 *vm_area_struct* 구조체로 관리되며, 리스트로 연결되어 있다. 이 리스트의 주소는

*mmap*에 저장되어 *mm_struct* 구조체에 저장된다. *mm_struct* 에는 프로세스의 가상메모리와 그것에 관련된 정보를 관리하는데, 해당 *mm_struct* 를 참조하는 쓰레드 개수는 *mm_user* 에 저장된다.

exit_mm 함수는 *mmput* 함수를 호출하는데, *mmput* 함수는 *mm_user*를 1씩 감소시킨다. 종료되는 마지막 쓰레드의 경우, *mm_user* 의 값이 0이 되고 이는 *mm_struct* 를 참조하는 쓰레드가 하나도 없음을 의미하므로, 마지막으로 *mmput*을 호출한 쓰레드는 *exit_mmap* 함수를 호출하여 메모리 해제를 수행한다.

exit_mmap 함수는 *unmap_vmas* 함수를 호출하여, VMA내에 프로세스의 페이지 테이블에 맵핑된 페이지를 언맵하는 절차를 시작한다.

언맵된 페이지는 *mmu_gather*의 페이지 구조체 리스트인 *pages*에 저장된다. 종료되는 프로세스의 마지막 쓰레드에서 *tlb_flush_mmu* 함수는 *pages*에 있는 페이지들의 참조 카운터를 감소시키고, 해당 페이지에 대한 TLB를 플러쉬 함으로써 실제 메모리 회수가 이뤄진다. 이때 해당 프로세스에만 사용되던 페이지는 참조 카운터가 0으로 되어 메모리 할당자(buddy system)에 반납된다.

3.4 앱 강제종료 과정 시간 분석

표 1 Process kill 소요시간(us)

대상 프로세스	Inputmethod.latin	com.android.camera2
LMK가 SIGKILL 전달	47	108
Thread의 SIGKILL 처리	271	1,077
메모리 회수	10,785	8,965
전체 과정	11,103	10,150

표 1은 Ftrace를 이용해 메모리가 부족한 상황(balloon driver을 이용해 시스템의 메모리 사용량을 증가시키고 Chrome, 카메라, MX player, Play store을 캐싱하였다.)에서 Asphalt 8을 실행시켰을때 종료되었던 inputmethod.latin 프로세스와 com.android.camera2 프로세스를 대상으로 시간을 측정하였다. 측정 결과, LMK의 *send_sig* 함수가 시그널을 종료대상 프로세스의 쓰레드에게 보내고, 시그널 핸들링을 위해 쓰레드를 wake-up하기 까지 약 47us의 시간이 소요됐다. Wake-up된 쓰레드가 시그널을 처리하고, 마지막 쓰레드가 메모리 회수를 위해 *exit_mmap* 함수를 호출하는데 까지 약 271us의 시간이 소요 됐으며, 실제 메모리를 회수하는데 약 10,758us의 시간이 소요되어 LMK가 하나의 프로세스를 종료시킬때 오버헤드(11,103us)의 대부분은 실제 메모리 회수과정이 차지함을 알 수 있다.

com.android.camera2의 경우 LMK의 시그널 전달에 약 108us, 쓰레드의 시그널 처리에 약 1,077us가 소요됐는데, 이는 com.android.camera2(41개), inputmethod.latin(13개)의 쓰레드 개수 차이로 인해, 많은 쓰레드에게 시그널을 보내고, 시그널을 처리하는 쓰레드가 많아서, inputmethod.latin 보다 많은 시간이 소요됐다. 하지만 메모리 회수 과정에서는 약 8,965us가 소요되어 inputmethod.latin 프로세스 보다 오히려 짧았는데, LMK에 의해 종료되는 다른 프로세스들(Inputmethod.latin이 종료될 때 다른 많은 프로세스들이 LMK에 의해 종료되었다.)과 회수되는 메모리의 크기 차이 때문이다. 이를 통해, 알 수 있는 것은 프로세스 종료 과정에서 메모리 회수는 가장 마지막에 일어나며, 그 시간 또한 전체 종료시간의 대부분을 차지한다는 것이다.

4. 결론 및 향후 연구

본 논문에서는 메모리 부족시 앱의 실행시간 측정과 LMK에 의한 프로세스 종료과정 분석 및 소요시간을 측정하였다. 메모리가 부족한 상황에서 앱의 실행시간은 증가하는 것을 확인 하였고 그것은 LMK에 의해 발생한 프로세스 종료과정의 오버헤드라는 것을 확인하였다. 많은 메모리를 얻기위해 LMK가 많은 프로세스를 종료한다면, 그 오버헤드는 더욱 커진다. 이때, 메모리 회수를 우선 수행할 수 있다면, 앱의 실행시간은 줄어들 수 있을 것이다. 향후, 좀더 일찍 메모리 회수를 하는 방법으로 앱의 실행시간을 줄이는 방법에 대해 연구하고자 한다.

5. 감사의 글

이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2014R1A1A2054658)

참 고 문 헌

[1] Smartphone OS Market Share, 2015 Q2. http://www.idc.com/prodservsmartphone-os-market-share.jsp.

[2] Sejun Kwon, Sang-Hoon Kim, JinSoo Kim, Jinkyu Jeong, “Managing GPU Buffers for Caching More Apps in Mobile Systems.”, In Proc. EMSOFT’15, Oct. 2015.

[3] Karim Yaghmour, “Embedded Android: Porting, Extending, and Customizing”, O'Reilly Media 2 edition, page 36, 2013.

[4] Android. ActivityManager. http://developer.android.com/reference/android/app/ActivityManager.html.

[5] 김민지, 권혁진, 신동군, “안드로이드 플랫폼에서 스왑기법의 성능 분석”, *한국정보과학회 2012 한국컴퓨터종합학술대회 논문집,* 제39권 제1호(A), 2012.6

[6] Android. Managing the activity lifecycle. http://developer.android.com/training/basics/activitylifecycle/index.html, Oct. 2013.

[7] Android. Android Debug Bridge. http://developer.android.com/guide/developing/tools/adb.html.

[8] Nexus 5. https://support.google.com/nexus/answer/6102470?hl=ko&rd=1

[9] Android. Logcat. http://developer.android.com/tools/help/logcat.html.

[10] David A Rusling, “The Linux Kernel”, *kos.enix.org*, 1999.

[11] Wolfgang Mauerer, “Professional Linux Kernel Architecture”, *Wrox*, page 291, 2008.