안드로이드 플랫폼에서 스왑기법의 성능 분석

김민지[0], 권혁진, 신동군
성균관대학교 컴퓨터공학과
kimmj0857@skku.edu, hjkwon0124@skku.edu, dongkun@skku.edu

Performance Study on Swap-Enabled Android Platform

Min Ji Kim, Hyukjin Kwon, Dongkun Shin
Department of Computer Engineering, Sungkyunkwan University

요 약

리눅스 기반의 운영체제인 안드로이드는 스마트 디바이스를 위해 만들어진 운영체제로 기존의 리눅스에서 제공하던 메모리 관리 기법인 스왑(Swap)기능을 기본적으로 제공하지 않고 있다. 하지만, 스마트 디바이스의 활용도가 높아짐에 따라 디바이스 상에서 여러 가지 기능을 제공하게 되었고 그로인해 기존의 메모리 공간보다 더 큰 메모리 공간을 요구하게 되었다. 따라서 메모리 관리에 있어 스왑기법과 같은 소프트웨어적 접근에 대한 관심이 높아지고 있다. 본 논문에서는 안드로이드 운영체제에 대한 스왑기법의 적용에 초점을 맞추어 스왑기법의 적용 시 성능 저하 문제를 기존의 LMK 기법과 비교하여 제시하고 있다.

키워드: 안드로이드, 스왑기법, 메모리 관리

1 서론

안드로이드(Android)[1]는 리눅스(Linux) 기반으로 만들어진 운영체제로 모바일 기기에 최적화된 운영체제이다. 따라서 기본적으로 지원하는 리눅스의 기능들 중 상당수가 제외되었는데 그 중 하나가 가상 메모리 관리와 관련된 스왑(Swap) 기법이다. 스왑기법이란, 가용 가상 메모리 공간을 늘리는 기법으로 메모리 공간이 부족하게 되면 하드 디스크 내의 지정된 스왑 공간으로 현재 메모리상에 있는 페이지를 선택하여 내보내는 기법이다. 이렇게 하면 가상 메모리 공간의 크기는 실제 물리적 메모리 공간의 크기와 하드 디스크 상의 스왑 공간의 크기를 합한 것과 같아지므로 실질적으로 더 큰 메모리 공간을 사용할 수 있게 된다.

스왑기법을 사용하게 되면 현재 쓰이지 않거나, 자주 쓰이지 않는 페이지를 스왑 공간으로 내보냄으로써 현재 메모리를 필요로 하는 애플리케이션에게 메모리를 우선적으로 할당할 수 있다는 장점이 있다. 하지만 기존의 메모리(RAM)에 비해 하드 디스크의 읽기 속도는 매우 느리므로 프로그램의 실행 속도가 현저히 느려지게 된다는 단점이 있다. 따라서 스마트폰 플랫폼인 안드로이드에서는 스왑공간으로 활용할 플래시 메모리의 느린 성능을 고려하여 스왑공간이 존재하지 않는 운영체제로 디자인하였다.[2] 하지만 모바일 시장이 확대됨에 따라 모바일 기기 즉, 스마트폰 또는 태블릿 컴퓨터를 활용하여 여러 가지 기능들을 수행할 수 있게 되었고, 그에 따라 사용자들은 안드로이드가 탑재된 모바일 기기를 이용하여 많은 동작들을 하게 됨으로써 실제 메모리 공간보다 더 많은 메모리 공간을 요구하게 되었다. 따라서 실제로 대용량의 메모리를 사용하는 것과 더불어 소프트웨어적으로도 메모리 공간을 효율적으로 확보하는 기법의 필요성이 커졌다. 또한, 초창기의 플래시 메모리에 비해서 최근의 SD 카드나 eMMC 장치는 높은 성능을 제공하고 있기 때문에 기존의 리눅스에서 메모리 관리 기법으로 제공하였던 스왑기법에 대하여 고려할 수 있게 되었다.

실질적으로 안드로이드 운영체제에 스왑기법을 적용하기 위해서는 여러 가지 고려할 점들이 존재한다. 첫 번째로 고려할 점은 성능적 측면으로 스왑기법의 단점으로 작용하였던 성능 저하가 안드로이드에서 얼마나 일어나는 지에 대한 고려가 있어야 한다. 두 번째로는 스왑 공간으로 사용할 micro SD카드의 수명적 측면이다. 일반적으로 micro SD카드는 낸드 플래시 기반의 이동식 플래시 메모리 카드로 덮어쓰기가 불가능하여 페이지 단위로 쓰고 해당 영역을 다시 쓰려면 블록 단위로 지워야 한다는 단점이 존재한다. 따라서 블록 단위로 지워진 횟수에 따라 정해진 수명이 있는데 일반적으로 micro SD카드의 경우 블록 당 10000번 정도의 지움 횟수를 보장한다고 알려져 있다. 따라서 스왑기법을 사용하였을 때 스왑 영역이 쓰이고 지워지는 횟수를 고려하여 스왑기법을 적용할 필요성이 있다.

본 논문에서는 안드로이드 운영체제에 스왑기법을 적용하기 위한 관련 실험 수행 내용과 그 결과를 분석한 결론 및 향후 연구 주제에 대한 방향을 제시한다.

2 관련 연구

현재 안드로이드 운영체제에서 micro SD카드를 활용한 스왑기법의 적용에 대한 연구 자료는 거의 없는 실정이다. 하지만 그와 유사하게 SSD(Solid-State Disk)를 가상 메모리 공간으로 활용함과 동시에 데이터의 크기에 따라 압축 여부를 결정하여 스와핑을 적용하는 기법에 대한 특허 자료를 찾을 수 있었다.[3] 정해진 크기의 페이지들에 저장된 데이터는 만약 그것을 압축한 크기와 메모리상에 작은 블록

본 연구는 지식경제부 및 정보통신산업진흥원의 지원사업의 연구결과로 수행되었음.

분할 영역의 크기가 맞을 때 압축하여 저장된다. 제시된 알고리즘은 데이터의 약 90%를 압축할 수 있기 때문에 스왑 공간의 사용을 최소화 할 수 있다는 장점이 있다.

3 메모리 스왑기법

3.1 가상 메모리

가상 메모리란 메모리 관리 기법 중 하나로 다중 작업을 수행하는 커널(Kernel)을 위해 만들어졌다. 실제 컴퓨터 데이터 스토리지(랜덤 접근 메모리, 하드 디스크)를 하나의 공간으로 가상화하여 실제 바로 접근이 가능한 주 기억 장치와 같이 보이도록 하여 더 큰 가상 메모리 공간을 확보할 수 있다.

3.2 스왑 공간

스왑 공간이란 가상 메모리 시스템을 지원하는 운영체제에서 메모리의 확장으로 사용되는 스토리지의 특정 부분을 말한다. 특정 프로그램이 메모리를 요구하는 상태에서 실제 할당 가능한 메모리가 부족한 경우 현재 DRAM에 로드되어 있는 페이지 들 중 하나를 선택하여 스왑 공간으로 교체한다. 리눅스에서 스왑 공간에 저장되는 페이지는 익명 페이지(anonymous page)로 런타임에 할당받은 힙(heap)또는 스택 영역이다. 스왑 동작 시 한 번에 이동되는 단위는 페이지 단위이며 스왑-아웃이 되는 시점은 커널 상에 미리 지정되어 있는 swappiness값에 의하여 결정되는데 swappiness값이 커질수록 최대한 스왑공간을 활용하게 된다. 시스템 기본 값으로 설정되어있는 swappiness=60의 값을 메모리의 크기로 산출해주는 swap tendency공식에 적용하여 계산해보면 80% 이상의 메모리공간이 사용될 때 스왑-아웃이 발생하게 된다.

swap tendency[4] = 사용 중 메모리 크기 / 2 + distress* + swappiness

4 안드로이드 메모리 관리

4.1 Low memory killer

Low memory killer(LMK)는 안드로이드 메모리 관리를 위해 작성된 리눅스 커널 모듈의 하나이다. 기존의 리눅스 메모리 정책인 Out of memory killer(OOM killer)가 임베디드 환경에 적합하지 않다는 것에 착안하여 개발되었다. 실제 애플리케이션은 자신의 상태를 항상 갖고 있으며(adj값) 운영체제는 최소 메모리 하한선(minfree값)에 따라 다단계로 메모리 문제를 해결한다. adj값과 minfree값은 서로 짝을 이루어 (0, 1024), (8, 4096)등과 같은 조합을 이루고 있으며 만약 남은 메모리의 양이 minfree값 이하가 되면 해당 minfree값에 대응되는 adj값 이상의 adj값을 가진 애플리케이션을 메모리에서 제거한다.

4.2 Out of memory killer

Out of memory killer는 리눅스 커널의 기본 메모리 정책으로 스왑 공간을 포함한 메모리 공간상에 더 이상 할당 가능한 메모리 공간이 남아있지 않을 때 적용된다. 유닉스 기반의 운영체제에서는 우선순위가 낮은 프로그램을 메모리에서 제거하게 되는데 일반적으로 메모리를 가장 많이 할당받고 있는 프로세스를 선택하여 제거한다.

5 실험 결과 및 평가

5.1 실험 환경

본 논문의 실험에서는 안드로이드 디버그 브릿지(Android Debug bridge)[6]를 사용하여 실험을 진행하였으며 대상으로 한 모바일 기기의 모델은 삼성 갤럭시S2 GT-I9100로 기기의 사양은 다음과 같다.

- CPU : 삼성 S5PC100 AP 듀얼코어 1GHz 프로세서
- 메모리 : 1GB RAM, 4GB ROM
- micro SD카드 : 8GB Class 6

기존의 안드로이드 커널은 스왑기법을 제공하지 않기 때문에 커널 빌드 시 환경설정에서 익명 페이지의 사용을 가능하게 만든 뒤 빌드하여 실험하였다. 스왑 공간은 micro SD카드 전체를 스왑 공간으로 지정하여 실험하였고, 스왑 공간을 사용하기 위해서 자동으로 swap on/off 명령어를 실행시켜주는 스크립트로 이루어진 「swapper2」라는 애플리케이션을 이용하여 swap on/off를 조작하였다.

5.2 안드로이드 프로파일링(profiling) 도구

본 논문의 실험을 위하여 안드로이드를 프로파일링할 수 있는 여러 가지 도구들을 사용하였다.

- top : 실시간으로 프로세스 진행상황을 출력해주는 명령어로 특정 PID(Process ID)를 지정하면 해당 프로세스의 CPU 사용량, 메모리 사용량, 스왑 공간 사용량을 출력한다. 실험에서는 앱 별 프로세스가 얼마나 많은 메모리를 할당받고 있는지를 확인하기 위해 해당 명령어가 사용되었다.
- /proc/kmsg : 커널이 출력하는 메시지들이 작성되는 파일로 LMK가 언제, 어떤 프로세스를 선택하여 종료시키는지를 확인하기 위하여 실제 커널 소스(lowmemorykiller.c)에 printk를 삽입하여 그 동작을 확인하였다.
- blktrace : blktrace는 블록 계층에서 입출력을 관찰하는 도구로 지정된 블록 디바이스에 대한 요청 큐(queue)동작 정보를 제공한다. 실제 스왑 공간으로 읽기와 쓰기가 어떻게 진행되었는지에 대한 분석을 위해 사용하였다.
- free : 실제 메모리와 스왑 공간의 용량과 각각의 남은 공간, 사용량을 보여주는 명령어로 전체적인 메모리 사용 상태를 확인할 때 사용하였다.

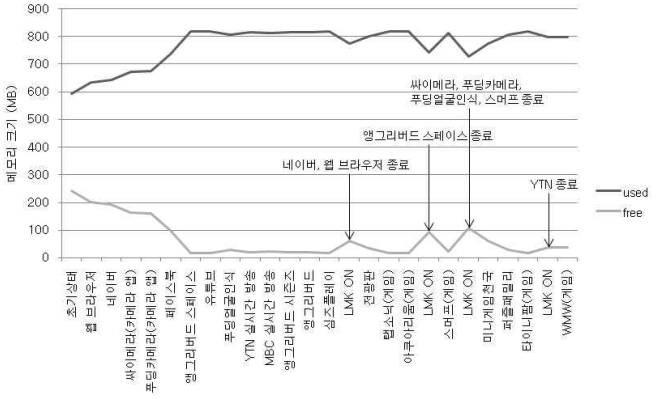

그림 1 LMK에 의한 메모리 변화

*distress : 커널이 메모리를 반환하는데 어려움을 겪는 정도를 나타내는 수치. 메모리 반환 시도가 거듭될수록 수치는 높아진다.

5.3 Low memory killer 적용에 따른 메모리 변화

LMK가 적용된 후에 메모리의 상태가 어떻게 변하는지에 대한 전체적인 변화를 관찰하기 위하여 LMK가 적용된 직후의 메모리 상태를 기록하였다. 그림 1에서 볼 수 있듯이 애플리케이션을 순서대로 실행시켰을 때 메모리 사용량이 늘어남과 동시에 남은 메모리 공간이 줄어드는 것을 볼 수 있다. 그 상태에서 LMK에 의해 애플리케이션이 종료되면 메모리가 반환되어 일시적으로 남은 메모리 공간이 늘어나는 것을 볼 수 있다. 처음 LMK가 발생한 상황을 살펴보면 심즈 플레이를 실행시킨 후의 메모리 상태는 사용 중인 메모리가 818.7MB, 남은 메모리의 크기가 16.8MB였는데 LMK에 의해 네이버와 웹 브라우저 종료가 된 후에는 사용 중인 메모리가 776.2MB, 남은 메모리의 크기가 59.3MB로 42.5MB의 메모리 반환이 일어난 것을 알 수 있다.

5.4 스왑기법 적용에 따른 로딩시간 변화

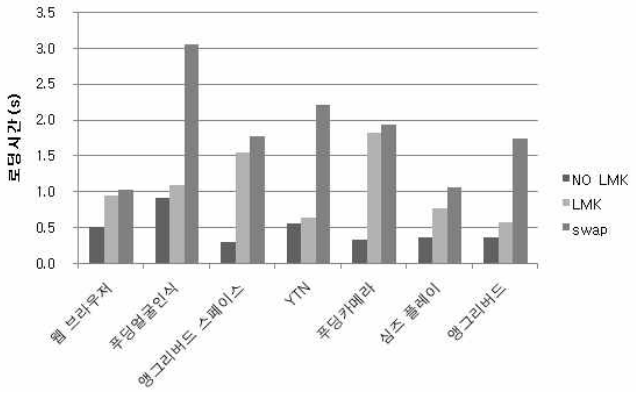

그림 2 LMK와 스왑기법에 따른 로딩시간 비교

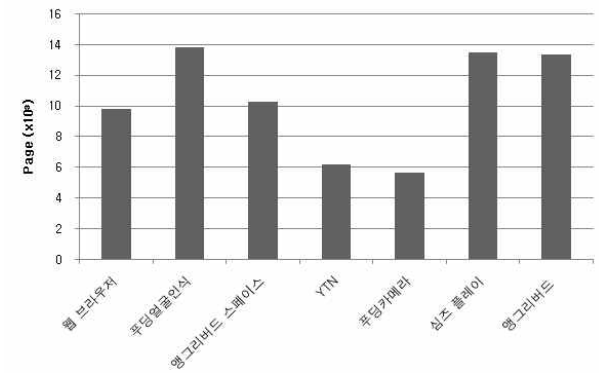

그림 3 종료된 애플리케이션 별 할당된 페이지 수

스왑기법 적용을 위해서는 기존의 메모리 관리에 사용되던 LMK 적용 후와 스왑-아웃 후 로딩 시간에 대하여 비교해 볼 필요가 있다고 판단하여 실험을 진행하였다. 해당 애플리케이션이 실행될 때 로딩 시간이 로그 기록으로 기록되는 데, 그 정보를 이용하여 로딩 시간을 분석하였다.

그림 2의 그래프에서 NO LMK로 표시된 자료는 LMK에 의해 종료되지 않은 애플리케이션의 로딩 시간을 측정한 것이고 LMK로 표시된 자료는 LMK에 의해 종료된 애플리케이션을 다시 실행하였을 때 걸리는 로딩 시간이며 마지막 swap으로 표시된 자료는 스왑-아웃된 애플리케이션을 다시 실행하였을 때 걸리는 로딩시간을 측정한 것이다. 여기서 스왑-아웃된 애플리케이션은 종료된 애플리케이션들 중 LMK와 OOM에 의해 종료된 것이 아닌 경우를 스왑에 의한 종료라고 가정하고 실험을 진행하였다. LMK에 의한 로딩시간과 스왑-아웃 후 로딩 시간 모두 증가하였으나 기존의 로딩시간과 비교하여 스왑-아웃 후의 로딩시간이 훨씬 크다는 것을 알 수 있다. 실제 LMK에 의한 애플리케이션 종료 이후 로딩시간은 약 2.68배 정도 늘어난 것에 비해 스왑-아웃으로 인한 프로세스 종료 이후의 재실행에는 로딩시간이 약 4.17배 정도 증가하였다. 그 이유는 스왑-아웃이 되면 기존의 메모리에서 micro SD카드로 프로세스가 밀려나기 때문에 다시 메모리로 로딩하기 위해서는 micro SD카드에서 읽어 와야 하는 부하가 발생하기 때문이다. 하지만, 실험에서 사용한 SD 카드는 클래스 6급으로 읽기와 쓰기 속도가 48Mbps로 알려져 있고 최근에는 읽기와 쓰기 속도가 80Mbps을 보장하는 클래스 10급의 고성능 SD 카드가 많이 사용되는 추세임을 고려할 때, 스왑으로 인한 성능 저하는 본 실험결과에 비해 미미할 것으로 예상된다. 그림 3의 그래프를 보면 각 애플리케이션 당 할당받은 페이지 수를 나타내고 있는데 그림 2의 그래프 결과와 함께 고려하면 LMK에 의해 종료된 후에 재실행된 애플리케이션 중 로딩시간이 다른 것보다 많이 늘어났던 애플리케이션의 경우, 할당된 페이지의 수가 많은 애플리케이션임을 알 수 있었다.

6 결론 및 향후 연구

본 논문에서는 안드로이드 메모리 공간을 증가시킬 수 있는 스왑기법의 적용에 대한 실험에 대하여 소개하고 있다. LMK를 적용하였을 때 메모리의 반환이 일어나 남은 메모리의 양이 늘어나는 것을 확인할 수 있었으며 LMK와 스왑-아웃이 적용된 후 로딩시간을 비교하였을 때 LMK에 비해 스왑-아웃의 경우 로딩 시간이 더 오래 걸리나 최근 micro SD카드의 성능을 고려할 때 성능 저하가 크지 않을 것을 예상할 수 있었다. 실제 class 10급의 micro SD카드의 환경에서 성능저하가 얼마나 큰 지에 대한 정확한 실험이 필요하고 micro SD카드의 수명적 측면에 대한 실험과 현재 페이지 단위로 이루어지고 있는 스왑 기법과 프로세스 단위로 이루어지고 있는 LMK를 비교하여 스왑의 성능을 증대시킬 수 있는 방법에 대한 연구를 진행하고자 한다.

참고 문헌

[1] Android(operating system)
http://en.wikipedia.org/wiki/Android_(operating_system)
[2] Dalvik VM Internals, Google I/O Developer Conference, 2008
[3] William D. Miller, Gary L. Harrington, Lawrence M. Fullerton, E.J.Weldon, Jr, Christopher M. Bellman, "Solid-state RAM data storage for virtual memory computer using fixed-sized swap pages with selective compressed/uncompressed data store according to each data size" (US5490260)
[4] swapping behavior
http://lwn.net/Articles/83588/
[5] Android Debug Bridge
http://developer.android.com/guide/developing/tools/adb.html