공학석사학위논문

연쇄적 우선순위 향상에 의한 안드로이드 기반 스마트폰의 사용자 반응성 향상

Improving Interactivity via Chained Priority Boosting for Android Smartphone

2013년 2월

서울대학교 대학원

전기컴퓨터공학부

이 중 현

연쇄적 우선순위 향상에 의한 안드로이드 기반 스마트폰의 사용자 반응성 향상

Improving Interactivity via Chained Priority Boosting for Android Smartphone

지도교수 홍 성 수

이 논문을 공학석사학위논문으로 제출함

2013년 2월

서울대학교 대학원

전기컴퓨터공학부

이 중 현

이중현의 석사학위논문을 인준함

2013년 2월

위 원 장 문 수 묵 (인)

부 위 원 장 홍 성 수 (인)

위 원 김 태 환 (인)

초록

지난 2008년 처음 등장한 이후 안드로이드는 지속적으로 발전하면서 현재 가장 많이 사용되는 모바일 기기용 OS가 되었다. 경쟁 OS 대비 안드로이드의 가장 큰 특징 중 하나는 리눅스 커널을 기반으로 하여 멀티태스크 실행 환경을 지원한다는 것이다. 모바일 기기에서 멀티태스킹이 가능하게 됨으로써 안드로이드의 사용자 편의성은 크게 증대되었다.

그러나 다중 태스크가 동시에 실행되는 경우 순간적으로 사용자 터치에 대한 반응이 지연될 수 가 있다. 이는 여러 태스크가 CPU를 놓고 경쟁하는 과정에서 사용자와 상호 교류하는 태스크가 다른 태스크들에 밀려 원하는 만큼 CPU를 얻지 못하기 때문에 발생한다.

본 논문에서는 안드로이드 기반 스마트폰에서 다중 태스크가 동시에 실행되는 환경에서 사용자와 반응하는 태스크를 우선적으로 처리하기 위한 연쇄적 우선순위 부스트 기법을 제시한다. 안드로이드 프레임워크 단에서는 사용자가 화면을 터치하면서 발생하는 시스템 스레드들을 커널에게 알려주고, 커널은 사용자의 터치가 발생할 때마다 기록된 시스템 스레드들의 우선순위를 부스트 시킨다. 또한 현재 사용자와 상호 교류하고 있는 안드로이드 응용의 백그라운드 스레드들을 식별하여 부스트 시켜줌으로써 사용자 터치에 대한 응답시간을 줄인다.

제안한 기법을 상용 스마트폰에 적용하여 유용성을 검증하였다. 실험 결과에 따르면 기존 안드로이드에 제안한 기법을 적용한 경우 반응시간이 평균 기존의 31.91%로 감소하였다.

주요어 : 안드로이드, 스마트폰, 사용자 반응성, 우선순위 부스팅

학 번 : 2011-20907

목차

표 목차

그림 목차

제 1 장 서론

구글이 안드로이드를 처음 공개하였을 때만 해도 안드로이드는 조악한 수준에 머물러 있었다. 그러나 오픈 소스라는 특징으로 수정이 용이하여 여러 번 판올림을 거치면서 빠르게 개선되어 왔다. 또한 안드로이드는 자바 가상 머신을 기반으로 하고 있어 이종 하드웨어에 포팅이 용이하다. 이는 안드로이드가 멀티코어와 같은 최신 아키텍처에 쉽게 적용될 수 있다는 것을 의미한다. 그 결과 현재 모바일 기기에서 가장 널리 쓰이는 플랫폼으로 부상하였다.

그러나 안드로이드가 오랜기간에 걸쳐 내포하고 있던 여러 문제점을 조금씩 수정해 왔음에도 불구하고 단 한 가지 문제에 대해서는 오랫동안 지적받아왔다. 그것은 경쟁 플랫폼 대비 낮은 사용자 반응성이다. 안드로이드는 리눅스 커널을 기반으로 한다. 리눅스 커널은 여러 개의 태스크를 동시에 실행시킬 수 있는 멀티태스킹을 지원한다. 이러한 멀티태스킹 기능은 사용자 편의성에 큰 기여를 하는 것은 사실이다. 그러나 상대적으로 제한된 컴퓨팅 자원을 가진 모바일 기기에서 멀티태스킹은 실제 사용자가 사용하고 있는 응용에게 적절한 자원 배분을 보장할 수 없다.

안드로이드의 개발사인 구글 역시 이러한 문제를 인지하고 지난 x월 사용자 반응성 개선에 주안점을 둔 안드로이드 4.2 버전 Jellybean을 출시하였다.[1] Jellybean은 이전 버전에 비해 사용자 반응성이 크게 향상된 모습을 보여주었다. Jellybean에서 적용된 사용자 반응성 개선 메커니즘은 크게 두 가지로 분류할 수 있다. 첫 번째 사용자 터치 입력을 디바이스로 받아서 사용자 응용에게 전달하는 과정을 간소화시킨 것이다. 두 번째 기법은 Wait for VSync 기법으로서 화면에 출력하는 주기를 일정

하게 한 것이다. 이로 인해 화면 변화가 스크린에 나타나기까지의 시간을 특정 시간 이내로 제한할 수 있었다. 이 두 가지 기법은 Jellybean에서 사용자 반응성을 개선하는 데 크나큰 기여를 했다.

그러나 Jellybean에서의 두 기법은 터치에 따른 사용자 요청을 처리하는 시간을 고려하지 못했다. 특히 다수의 응용이 동시에 실행되는 경우에서는 응용 간에 CPU를 놓고 경쟁하는 상황이 종종 벌어진다. 이러한 상황은 반응 지연을 야기한다. 본 논문에서는 다수의 응용이 동시에 실행되는 상황에서 현재 스크린에 보여지는 사용자 응용에 CPU 우선권을 부여하는 방식으로 사용자 반응성을 개선하는 기법을 소개한다. 그리고 제안한 기법을 상용 안드로이드 플랫폼 기반의 스마트폰에 직접 적용하고 효용성에 대해 검증한다.

이 논문은 다음과 같이 구성된다. 2장에서는 안드로이드 Jellybean의 소프트웨어 스택과 구조에 대한 간략한 소개 및 안드로이드 프레임워크 단에서 사용자 터치를 감지하고 처리하는 과정을 살펴본다. 3장에서는 안드로이드 플랫폼 기반의 모바일 기기에서 발생할 수 있는 CPU 경쟁으로 인한 사용자 터치 반응 지연 문제를 정의한다. 4장에서는 3장에서 제기한 문제들을 극복하는 기법을 설명한다. 5장에서는 제안한 기법의 효용성에 대해 검증한다. 마지막으로 6장에서는 본 연구의 의의와 향후 발전 방향에 대해 논의한다.

제 2 장 배경

본 장에서는 멀티태스킹 환경에서 발생할 수 있는 안드로이드의 사용자 입력 반응 지연 문제를 분석하고 해결책을 제시하기 위해 필요한 내용을 서술한다. 먼저 목표 시스템인 안드로이드 운영체제의 소프트웨어 스택과 구조에 대해서 간략히 설명한다. 다음으로 안드로이드 운영체제에서 사용자의 터치 이벤트를 처리하는 과정을 분석한다. 마지막으로 입력 반응 지연 문제를 다룬 관련 연구들을 소개한다.

2.1 안드로이드

안드로이드 운영체제의 소프트웨어 스택은 아래의 그림과 같다. 안드로이드는 크게 네 가지 계층으로 나눌 수 있는데 리눅스 커널 계층과 라이브러리 계층, 안드로이드 런타임 계층, 어플리케이션 프레임워크 계층, 어플리케이션이 그 것이다. 각 계층에 대한 설명은 아래에 기술한다.

- **리눅스 커널**

안드로이드 운영체제는 시스템에 탑재된 여러 하드웨어들을 제어하고 관리하기 위한 커널로 리눅스를 사용하고 있다. 안드로이드 2.3 버전인 Gingerbread까지는 리눅스 커널 2.6 버전을 기반으로 하였고 안드로이드 4.0 버전인 Icecream Sandwich부터는 리눅스 커널 3.0 버전을 사용하고 있다. 가장 최신 버전인 안드로이드 4.1.2 버전인 Jellybean은 리눅스 커

널 3.1.3을 채택하였다.
리눅스의 main 태스크 스케줄러는 초기의 라운드 로빈 스케줄러, O(N) 스케줄러, 2.4 버전의 O(1) 스케줄러를 거치면서 개선되어 왔다. 현재 리눅스는 2.6.23버전부터 CFS(Completely Fair Scheduler)를 채택하고 있다. 현재 안드로이드는 앞서 설명한대로 리눅스 2.6 버전 이상을 기반으로 하고 있기 때문에 CPU 스케줄러로 CFS를 사용한다고 할 수 있다.
안드로이드 운영체제의 리눅스 커널은 장치 제어 및 관리 기능뿐만 아니라 DVFS 등을 이용한 전력 관리 기능, binder나 ashmem과 같은 Inter-Process Communication(IPC) 기능, Low memory Killer 기능 등을 제공한다.

- **안드로이드 런타임**

안드로이드 런타임은 달빅(Dalvik) 가상 머신과 코어 라이브러리로 이뤄져 있다. 달빅 가상 머신은 JAVA 언어를 처리하기 위한 Register 기반의 가상 머신이다. 모바일 기기의 부족한 하드웨어 자원을 고려하여 메모리 사용량을 최적화하였다. 또한 동시에 여러 개의 가상 머신을 동시에 수행시킬 수 있게끔 설계하여 안드로이드의 멀티태스킹 기능의 근간이 된다.
코어 라이브러리는 안드로이드 프레임워크의 핵심 기능을 제공하는 라이브러리로 구성되어있다. DOM, SAX, XMLPullParser등 과 같은 XML parser 라이브러리와 Apache Commons, 데이터베이스를 위한 SQLite 등을 포함한다.

- **라이브러리**

안드로이드 운영체제는 대부분이 JAVA로 작성되었다. 그렇기 때문에 비교적 이종 하드웨어로의 포팅이 용이하다. 그러나 JAVA는 구조상 필연적으로 성능 하락을 야기한다. 그러므로 안드로이드 운영체제는 고성능이 필요하지 않은 부분은 JAVA로 작성하고 성능이 중요한 기능은 C/C++ 언어로 작성하였다. 안드로이드의 라이브러리 계층은 이러한 C/C++로 작성된 함수들의 집합을 말한다. 라이브러리 계층은 표준 C 라이브러리, 멀티미디어를 위한 Media Framework, 화면 출력을 관리하는 Surface Manager를 포함한다. 또한 현용 웹브라우저의 근간이 되는 Webkit 라이브러리와 2D/3D 렌더링을 위한 OpenGL|ES 라이브러리도 포함된다.

- **어플리케이션 프레임워크**

어플리케이션 프레임워크 계층은 사용자 응용을 구현하는데 필요한 대부분의 함수들의 집합이다. 이 계층은 안드로이드 사용자 응용의 기본 구성 요소를 제공한다. 사용자 응용의 실행 주기와 동작에 대해 정의한 Activity와 background 작업을 위한 Service, 응용과 다른 응용이 상호작용할 수 있게 도와주는 Content Provider, 응용의 UI 배치를 정의하는 View, 기반 하드웨어를 응용이 사용할 수 있도록 도와주는 Location Manager 등이 포함된다. 나중에 주로 살펴볼 InputReader와 InputDispatcher들 역시 이 계층에 속한다.
이 계층에 속하는 함수들은 각각의 목적에 맞게 구분되어 리눅스 단일 스레드로 동작하게 된다. 앞으로 이러한 스레드를 시스템 스레드로 통칭

한다. 이러한 구분은 다음에 나올 어플리케이션 계층의 스레드들과 구분하기 위함이다.

- **어플리케이션**

안드로이드 운영체제에서 수행되는 사용자 응용을 말한다. 실제 사용자와 상호 교류하는 최상위 계층이다. 특수한 경우를 제외하면 JAVA 언어로 작성되며 해당 응용의 속성을 설정하는 manifest 파일과 응용이 사용하는 자원을 담고 있는 파일, JAVA code 등으로 구성된다.
앞서 말했듯이 안드로이드 운영체제에서는 여러 사용자 응용이 동시에 수행될 수 있다. 그러나 일반적인 데스크탑 운영체제와 달리 모바일 기기에서는 플랫폼 한계로 인해 오직 하나의 응용만 화면에 보일 수 있다. 본 논문에서는 화면에 나타나는 유일한 응용을 사용자와 상호작용을 하는 응용으로 정의한다. 이외 화면에 보이지 않는 다른 응용을 백그라운드 사용자 응용으로 정의하도록 한다.
일반적인 안드로이드 사용자 응용은 main 스레드와 여러 개의 보조 스레드로 구성된다. main 스레드는 사용자의 입력에 반응하고 화면에 레이아웃을 배치하는 등의 UI에 관련된 작업을 수행한다. 보조 스레드의 경우 주로 CPU나 Memory 등 컴퓨팅 자원을 많이 소모하며 시간이 오래 걸리는 작업을 수행한다. 이렇게 분리된 이유는 UI를 담당하는 main 스레드의 처리가 부하가 큰 작업으로 인해 지연되지 않게끔 하기 위함이다. 이런 이유로 본 논문에서는 보조 스레드들을 worker 스레드로 통칭하기로 한다.

2.2 안드로이드의 터치 이벤트 처리 과정

본 절에서는 안드로이드 운영체제에서 터치 이벤트가 처리되는 과정을 기술한다. 사용자가 입력한 터치가 디바이스 드라이버와 Input 관련 시스템 스레드 등을 거쳐 사용자 응용에게 전달된 후 결과가 화면에 반영되기까지의 일련의 과정에 대해 분석한다. 전체적인 과정은 아래의 그림과 같다.

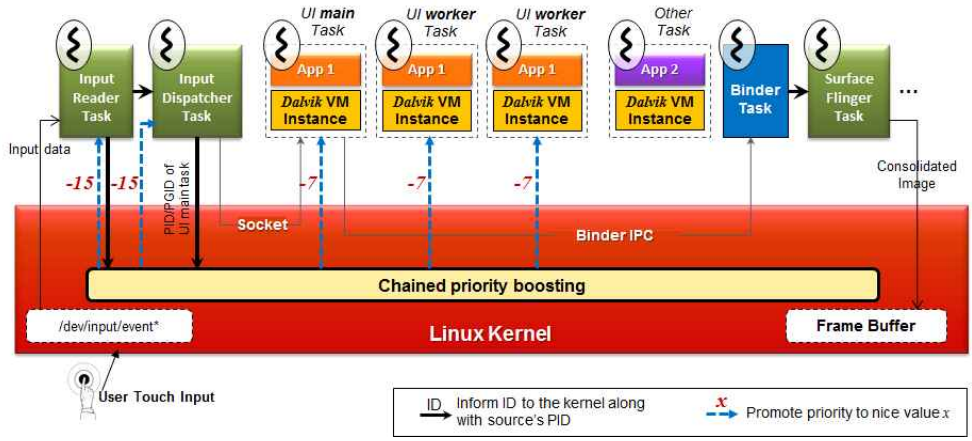

그림 1. 안드로이드의 터치 이벤트 처리 과정

사용자가 화면을 터치하면 터치 스크린 디바이스 드라이버에 의해 인터럽터 핸들러가 구동된다. 인터럽트 핸들러는 /dev/input/event 파일에 터치에 대한 정보를 기록한다.

ueventd 데몬 프로세스는 /dev/input/event 파일을 주기적으로 확인한다. 만약 사용자의 입력이 들어왔다면 Input Reader 시스템 스레드를 sleep 상태에서 깨운다.

Input Reader가 sleep 상태에서 깨어나면 입력에 대한 정보를 분석한다. 어떤 장치에서 발생한 신호인지, 어떤 형태의 입력이 들어왔는지를 구분한다. 만약 현재 들어온 입력이 터치스크린으로부터 발생한 것이라면,

Input Reader가 터치된 지점의 좌표와 사용자가 스크린을 누른 것인지 아닌지 등을 확인한다. 이러한 정보는 특정 구조체에 저장된다.

Input Reader가 해당 입력 이벤트에 대한 분석이 끝나면 Input Dispatcher가 호출된다. Input Dispatcher는 시스템의 응용을 관리하는 WindowManager 시스템 스레드에게 요청하여 현재 foreground 응용에 대한 정보를 얻어낸다. 그리고 Input Reader가 가공한 정보를 foreground 응용에게 IPC를 통해 전달한다.

Foreground 응용은 Input Dispatcher에 의해 깨어나서 입력에 관한 정보를 받게 된다. 보통 foreground 응용은 여러 개의 스레드로 구성되는데 그중 main 스레드가 입력 정보를 받는다. 일반적으로 main 스레드 사용자와 상호 교류를 담당한다. Main 스레드는 받은 입력의 형태에 따라 응용 개발자가 정의한 이벤트 리스너 함수를 호출한다. 예를 들어 사용자로부터 받은 입력이 특정 버튼을 누르는 것이었다면, 해당 버튼에 정의된 리스너 함수를 호출하게 된다.

경우에 따라 사용자의 터치 명령이 추가적인 스레드를 생성할 수도 있다. 사용자의 명령이 단일 스레드에서 처리하기 힘든 부하를 가지고 있다면 여러 개의 자식 스레드를 생성하여 병렬적으로 처리한다. 안드로이드는 사용자 응용에서 추가적인 스레드를 생성할 수 있는 방법으로 크게 두 가지를 제공한다. 그 중 AsyncTask를 이용한 방법이 부하가 적고 사용하기 쉽기 때문에 자주 사용된다. 이벤트 리스너 함수가 AsyncTask를 생성하고 처리를 기다린다.

사용자 명령에 대한 처리가 끝났다면 그 결과를 반영한다. 응용에 따라 다양한 형태로 결과를 출력한다. 예를 들어 파일 관리 응용의 경우 플래시 드라이브에 접근하여 특정 파일을 읽고 쓴다. 그림 그리기 응용에서는 화면을 갱신하도록 한다.

2.3 관련 연구

멀티태스킹 환경에서 사용자와 상호 교류하는 응용의 반응 지연 문제에 대한 연구는 리눅스 2.4 버전의 main 스케줄러였던 O(1) 스케줄러까지 거슬러 올라간다. 당시에도 리눅스의 경우 CPU의 성능에 비해 많은 태스크를 수행시킬 경우 사용자의 입력에 대한 반응이 지연되는 일이 종종 발생했다. O(1) 스케줄러는 이러한 상황을 경험적인 기법을 통해 사용자와 상호 교류하는 태스크를 찾아내어 우선순위를 조절하는 방식으로 타개하려 시도하였다. 그러나 O(1) 스케줄러의 경험적인 방식은 많은 corner case를 야기하였고 심각한 부작용이 뒤따랐다.

O(1) 스케줄러의 경험적인 기법의 한계를 극복하고자 Multi-Level Feedback Queue 스케줄러가 등장하였다. MLFQ 스케줄러는 사용자와 상호 교류하는 태스크를 간단한 통찰을 통해 찾았다. 태스크마다 time slice를 부여한 뒤 CPU를 더 필요로 하면 추가적으로 우선순위를 조절해주어서 각 태스크마다 적절한 CPU time을 받을 수 있도록 하였다.

안드로이드가 등장하고 마찬가지로 사용자 반응 지연 문제를 해결하고자 하는 시도가 있었다[3], [4]. 두 논문 모두 사용자와 상호 교류하는 응용의 우선순위를 조절함으로써 응용이 필요로 하는 CPU time을 보장하는 방식을 채용하였다. 차이점은 사용자와 상호 교류하는 응용을 찾는 방식에 있다. 두 논문 모두 사용자 반응 지연 문제를 우선순위 부스팅으로서 극복하려 했으나 안드로이드의 기본 성질에 대한 고려가 부족하여 충분한 성능향상을 얻어내지 못했다. 본 논문에서는 기존 연구에서 미처 고려하지 못한 부분을 개선하여 보다 향상된 기법을 제안하고자 한다.

제 3 장 문제 정의

앞의 2장에서 설명한대로 사용자의 터치가 발생하게 되면 관련 시스템 스레드가 순차적으로 깨어나서 각각의 역할을 수행한다. 시스템의 부하가 적은 경우라면 일련의 과정들이 연속적으로 일어나므로 지연없이 사용자의 터치 이벤트를 처리할 수 있다.

그러나 여러 응용이 동시에 수행되는 상황에서는 앞에서 언급한 시스템 스레드의 실행이 방해받을 수 있다. 아래의 그림()은 이러한 상황을 나타낸 것이다.

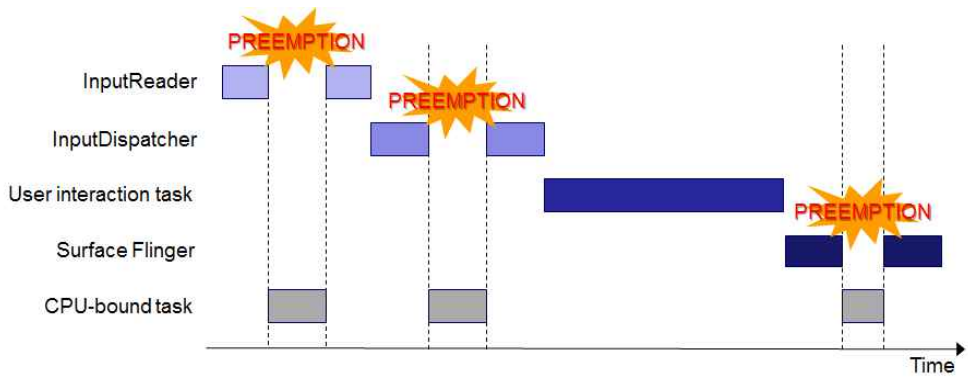

그림 2. 시스템 스레드가 실행 도중 background 응용에 의해 방해 받는 상황

그림에서 알 수 있듯이, background에서 수행되는 응용이 InputReader와 InputDispatcher와 같은 터치 입력 처리 관련 시스템 스레드의 실행을 방해하는 경우가 존재한다. 이러한 상황은 앞서 말한 리눅스 커널의 주 스케줄러인 CFS의 동작 메커니즘으로 인해 발생한다. 즉 현재 실행중인 스레드의 time slice가 다 소진되면, 실행큐에 있는 다른 스레드로 즉시 교체되기 때문이다. 만약 Input Reader나 Input Dispatcher가 자신

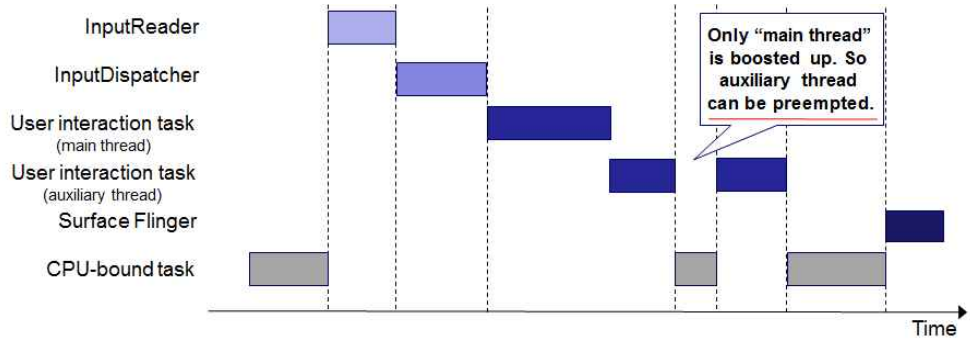

그림 3. Foreground 응용의 자식 스레드가 background 스레드에 의해 방해받는 경우

에게 할당된 time slice보다 더 많은 시간동안 작업을 해야 한다면 반드시 한번 이상은 다른 스레드들에 의해 선점당한다.

시스템 스레드뿐만 아니라 사용자 응용이 생성하는 background 스레드도 같은 이유로 방해를 받을 수 있다. 보통 background 스레드는 사용자 명령에 대한 처리를 담당하기 때문에 background 스레드의 실행 지연은 심각한 사용자 반응성 저하를 가져온다.

제 4 장 사용자 반응성 향상을 위한 기법

본 장에서는 안드로이드 플랫폼에서 터치에 의한 사용자 요청 처리 지연 시간을 줄이기 위한 연쇄적 우선순위 부스트 기법에 대해 설명한다. 제안된 연쇄적 우선순위 부스트 기법은 런타임에 사용자 터치에 반응하기 위해 필요한 스레드의 우선순위를 순간적으로 올려준다. 우선순위를 높여주는 대상 스레드들은 사용자 터치를 인지하고 처리하기 위해 동작하는 시스템 스레드들과 사용자의 요청을 처리하기 위해 발생하는 background 스레드들이 있다.

4.1 문제 해결을 위해 고려할 사항들

본 절에서는 제안한 기법을 구현하기 위해서 고려할 요소들에 대해서 언급한다. 연쇄적 부스트 기법을 안드로이드 플랫폼에 적용하기 위해서는 (1) 어떤 스레드들의 우선순위를 향상시킬 것인지, (2) 어떻게 우선순위를 올려줄 스레드를 구별해낼 것인지 (3) 얼마나 우선순위를 올려줄 것인지 (4) 언제 우선순위를 조절해줄 것인지 등과 같은 사항을 결정해야 한다.

4.1.1 부스트 대상 스레드

앞의 2장에서 설명했듯이, 사용자가 입력한 터치 정보는 디바이스 드라이버를 거쳐 InputReader, InputDispatcher를 거쳐 사용자 foreground

응용에게 전달된다. 그리고 이러한 사용자 터치 처리 관련 시스템 스레드들은 bakcground 응용에 의해 CPU 사용을 제한받을 수 있다. 이를 막기 위해 연쇄적 우선순위 부스트 기법은 InputReader와 InpuDispatcher를 우선순위를 높여주도록 한다.

또한 고려할 스레드로는 사용자 foreground 응용이 생성하는 background 응용이 있다. 일반적으로 사용자의 요청은 단일 스레드에서 처리된다. 그러나 부하가 큰 응용(인용)의 경우, 사용자 foreground 응용에서 자식 스레드를 생성하여 요청에 대한 처리를 병렬적으로 수행한다. 이러한 경향은 안드로이드 플랫폼 기반 스마트폰이 멀티코어를 탑재함으로써 더욱 가속화되고 있다. 멀티 스레드로 응용을 작성함으로써 CPU의 활용도를 높일 수 있는 것이다. 연쇄적 우선순위 부스트 기법에서는 이런 자식 스레드들을 찾아내어 우선순위를 상승시켜 준다.

4.1.2 부스트 대상 스레드 식별과정

스레드의 우선순위를 커널이 조절하기 위해서는 해당 스레드의 PID를 알고있어야 한다. 안타깝게도 리눅스 커널이 부스트 대상 스레드를 파악하기는 어렵다. 리눅스 커널 입장에서는 시스템 스레드나 사용자 응용 스레드나 단지 일반적인 태스크로 파악되기 때문이다. 특정 스레드의 우선순위를 조절하기 위해서는 커널에게 여러 스레드들 중 우선순위를 올려주고자 하는 스레드를 구별해내어 알려주어야 한다. 그러므로 연쇄적 우선순위 부스트 기법은 기본적으로 안드로이드 프레임워크 단이 가지고 있는 정보를 활용하여 커널 단에게 대상 스레드를 식별할 수 있도록 도와주도록 한다.

InputReader나 InputDispatcher와 같은 터치 이벤트를 처리하는 스레드

들은 상대적으로 식별해내기 용이하다. 이러한 스레드들은 사용자가 터치가 실행될 때 마다 반드시 실행되어야 하기 때문이다. 따라서 InputReader나 InputDispatcher가 실행되고 사용자의 입력이 감지되면, 특정 시스템 콜을 통해 리눅스 커널에게 알려주도록 한다.

터치 시점에서의 foreground 응용을 식별해 내는 것도 이와 유사하다. InputDispatcher는 터치 정보를 foreground 응용에게 전달하는 과정에서 IPC의 일종인 Binder를 사용한다. 이 때, InputDispatcher는 foreground 응용의 PID를 알 수 있게 되므로 시스템콜을 통해 커널에게 foreground 응용의 PID를 전달해주도록 한다.

만약 InputDispatcher가 터치 정보를 전달하는 시점에서 foreground 응용이 자식 스레드들을 가지고 있다면, 이러한 자식 스레드들의 PID들 역시 알아내야 한다. 그런데 리눅스 커널에서는 자식 스레드의 PID의 경우 부모 스레드의 Process Control Block에 저장되기 때문에 추가적인 식별 과정은 필요없다. 이 경우 부모 스레드인 foreground 응용의 PID를 InputDispatcher로부터 전달받아 부모 스레드의 Process Control Block를 탐색하여 자식 스레드들을 알아낼 수 있다.

문제가 되는 상황은 foreground 응용이 InputDispatcher로부터 터치 정보를 받은 후, 사용자 요청을 처리하기 위해 실행 도중에 자식 스레드를 생성하는 경우이다. 특히 앞서 설명한 AsyncTask는 fork() 시스템콜을 통해 생성되기 때문에 PID를 새롭게 부여받는다. 그러므로 AsyncTask의 PID는 안드로이드 프레임워크에서도 알기 어렵다.

제안한 기법에서는 AsyncTask의 PID를 파악하기 위해 AsyncTask가 생성될 때 마다 커널에게 시스템콜을 통해 알려준다. 커널은 시스템콜을 호출한 PID를 현재 foreground 응용의 PID와 비교함으로써 생성된 Process가 foreground 응용의 AsyncTask인지 확인할 수 있다.

4.1.3 우선순위 상승 정도

특정 스레드의 우선 순위를 조절할 수 있게 되었다면, 해당 스레드의 우선순위를 얼마나 조정해줄 것인지를 정해야 한다. 관련 연구(인용)에서는 사용자 foreground 응용이 사용자가 조절할 수 있는 정해진 매개변수 값만큼 우선순위를 조절해 주었다.

그러나 이러한 매개변수는 커널 컴파일 시 결정되는 값으로서 각 사용자 응용이 필요로 하는 CPU time을 보장한다고 보기 어렵다. 예를 들어 1ms 만큼 CPU를 필요로 하는 응용 1과 10ms 만큼 CPU를 필요로 하는 응용 2가 모두 동일하게 5ms를 보장받도록 조절하는 것은 공평하지 못하다. 이 경우 응용 2는 주어진 time slice 내에 해당 작업을 완료하지 못하게 된다.

이런 문제점을 해결하기 위해 제안하는 기법에서는 적응적으로 우선순위 부스팅을 적용한다. 이는 사용자와 상호작용하는 응용이 할당받은 time slice 내에 작업을 모두 끝나지 못했을 때, 추가적으로 우선순위를 부스팅하는 것을 말한다. 이로써 각 태스크마다 적합한 우선순위를 가지도록 할 수 있다.

제안한 기법에서 추가적으로 고려해야할 것은 안드로이드의 스레드간 관계이다. 앞서 설명했듯이 안드로이드 플랫폼은 시스템 내의 스레드들을 특정 기준에 따라 분류하고 Group으로 묶어서 관리한다. 각 group은 고유한 우선순위를 부여받는다. 만일 이러한 관계를 고려하지 않고 부스팅한다면 우선순위 역전 현상이 발생할 수 있다. 예를 들어 사운드 관련 스레드인 AudioService는 생성시에 -19의 nice value를 받는다. 그런데 우선순위 상승으로 인해 일반적인 스레드가 위의 사운드 처리 스레드보

다 더 높은 우선순위를 가지게 되면, 사운드 처리 스레드의 실행이 지연됨으로써 오히려 사용자 반응성을 낮아지는 결과를 야기할 수 있다.
제안한 기법에서는 사용자의 매개변수를 고려하되 각 group 별로 부여받은 우선순위의 차이를 유지하도록 한다. Foreground 응용의 경우 기본적으로 0의 nice value를 부여받고 InputReader, InputDispatcher는 -8의 nice value를 부여받는다. Time slice 소모 여부에 따라 단계적으로 우선순위를 조절하면서 정해진 최대 우선순위에 도달하면 더이상 우선순위 부스팅을 하지 않도록 한다. 제안한 기법에서 사용자 응용의 최대 우선순위는 -7로 정해져 있다. InputReader 등과 같은 시스템 스레드는 최대 우선 순위를 -15로 정해져 있다. 각각 보다 높은 우선순위를 가진 그룹의 최소 우선순위보다 한 단계 씩 낮게 정함으로써 우선순위 부스팅으로 인해 정해진 응용 그룹간 관계를 해치지 않도록 하였다.

4.1.3 우선순위 상승 및 하강 시점

제안한 기법은 사용자 터치에 대한 처리가 최대한 빠르게 이뤄지도록 하는 것이 목적이다. 따라서 스레드의 우선순위 상승 시점은 사용자가 터치 입력을 한 순간부터로 정하는 것이 합리적이다. 제안한 기법은 사용자가 터치 입력을 하고 InputReader와 InputDispatcher가 연이어 실행되면서 각 스레드의 우선순위를 상승시킨다. 또한 InputDispatcher가 터치 정보를 사용자 foreground 응용에게 전달하는 순간 사용자 foreground 응용과 자식 스레드의 우선순위를 상승시켜준다. AsyncTask의 경우 생성되면서 foreground 응용에서 파생되었음이 감지되면 즉시 우선순위를 상승시켜 준다.
고려할 점은 우선순위 하강 시점이다. 만일 사용자의 터치가 끝난 후에

도 여전히 우선순위를 내리지 않는다면 background 응용가 상대적으로 CPU를 덜 점유하게 되어 전체적인 throughput이 감소하게 된다. 제안한 기법에서는 사용자가 터치스크린에서 손을 뗀 것을 감지하는 순간부터 InputReader와 InputDispatcher 응용의 우선순위를 감소시킨다. 이 때 Foreground 응용 및 AsyncTask의 우선순위는 조절하지 않는다. Foreground 응용이 생성한 AsyncTask가 종료되지 않고 남아있다는 것은 Foreground 응용이 처리해야할 일이 더 남아있다는 것을 의미하기 때문이다. Foreground 응용과 AsyncTask의 경우 안드로이드 플랫폼의 홈 버튼이 눌려졌을 때에 우선순위를 하강시킨다. 안드로이드 플랫폼에서 홈 버튼은 현재 foreground 응용을 background로 보내는 것을 의미하기 때문이다. 따라서 홈 버튼이 눌려진 직후 foreground 응용과 AsyncTask의 우선순위를 하강하여 원래의 우선순위를 가지도록 설계하였다.

4.2 연쇄적 우선순위 부스트 기법

4.1절에서 언급한 설계 요소들을 종합하여 안드로이드 플랫폼 기반의 스마트폰에 적용하면 그림()와 같다.

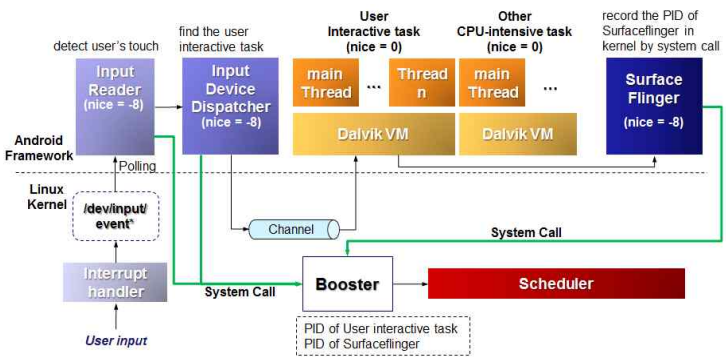

그림 4. 제안한 기법의 개괄

사용자에 의해 이벤트가 발생하면 InputReader가 깨어나 이벤트 발생을 감지해서 분석한다. 만약 사용자가 스크린을 누른 이벤트가 발생한 것이라면 자신의 우선순위를 상승시키는 시스템콜을 호출하고 터치 이벤트 정보를 InputDispatcher에게 전달한다. InputDispatcher는 InputReader로부터 전달받은 터치정보를 현재 수행되고 있는 응용 중 어떤 응용에게 전달할 것인지를 파악한다. 그리고 InputDispatcher 자신의 우선순위와 foreground 응용 및 자식 스레드의 우선순위를 상승시키는 시스템콜을 호출한다. 이 때 리눅스 커널은 현재 foreground 응용의 PID를 기록한다. Foreground 응용은 높은 우선순위를 가지고 사용자의 입력을 처리하다 경우에 따라 AsyncTask를 생성한다. 리눅스 커널은 AsyncTask의 생성을 감지하고 현재 foreground 응용이 생성한 것이라 판단되면 해당 AsyncTask의 우선순위를 상승시킨다.

사용자가 스크린에서 손을 떼면 InputReader와 InputDispatcher는 다시 자신들의 우선순위를 조절하는 시스템콜을 호출한다. 그리고 홈 버튼이 눌려지면 InputReader가 감지하고 foreground 응용과 자식 스레드들 및 AsyncTask의 우선순위를 하강시킨다.

제 5 장 평가

이 장에서는 상용 스마트폰에 제안한 연쇄적 우선순위 부스트 기법을 적용하고 유효성을 검증한다. 실제 안드로이드 응용을 사용하여 사용자 터치에 대한 반응이 느려지는 상황을 보이고 제안한 기법에 의한 성능 향상 정도를 평가한다. 마지막으로 결과에 대해 분석한 뒤 향후 개선 방향에 대해 논의한다.

5.1 검증 방법

본 절에서는 상용 스마트폰에 제안한 기법을 적용한 뒤 검증한 실험에 대해 설명한다. 또 성능 측정 방법에 대해서도 제시한다.

5.1.1 실험 환경

실험 환경은 안드로이드 레퍼런스 스마트폰인 Galaxy Nexus에서 실험하였다. Galaxy Nexus에는 리눅스 커널 3.0.3과 안드로이드 4.1 Jellybean이 탑재되어 있다. Galaxy Nexus는 듀얼 코어 TI Omap 프로세서와 1GB RAM이 장착되어 있다.

여러 응용이 동시에 실행하는 상황을 재현하기 위해 안드로이드 응용 중 Avairy, FFmpeg media encoder, AVG, PI calculator를 사용하였다. Avairy는 스마트폰 상의 이미지를 후처리 해주는 이미지 프로세싱 툴이다. FFmpeg media encoder는 동영상 파일을 다른 포맷으로 변환하는

툴이다. AVG는 스마트폰의 파일들을 검사하는 바이러스 검사 툴이다. PI calculator는 pi값 계산이 걸리는 시간을 측정하여 성능을 검증하는 벤치마크 툴이다.

5.1.2 실험 방법

위에서 언급한 응용들을 동시에 실행시킨다. 그리고 FFmpeg media encoder로 mp4 포맷의 동영상을 AVI 포맷으로 변환하는 작업을 수행했다. AVG로는 스마트폰의 전체 바이러스 검사를 수행하였다. PI calculator는 PI의 소수점 이하 2000 자리까지 계산하는 검사를 수행하였다. 위의 세 개의 응용을 모두 background로 보내고 Avairy를 foreground에서 실행시킨다. 그리고 특정 사진 파일을 불러와 광원 조절 필터를 반복해서 실행시켰다. 이 때 광원 조절에 걸리는 시간을 측정하였다.

기존 안드로이드 스마트폰을 대조군으로 하고 제안한 기법을 적용한 스마트폰을 실험군으로 하였다. 그리고 두 비교군 간 광원 조절에 걸리는 시간을 25회씩 측정하여 제안한 기법이 사용자 입력에 대한 반응 시간을 얼마나 감소시키는지 실험하였다.

5.2 결과 및 평가

실험 결과는 아래의 표와 같다.

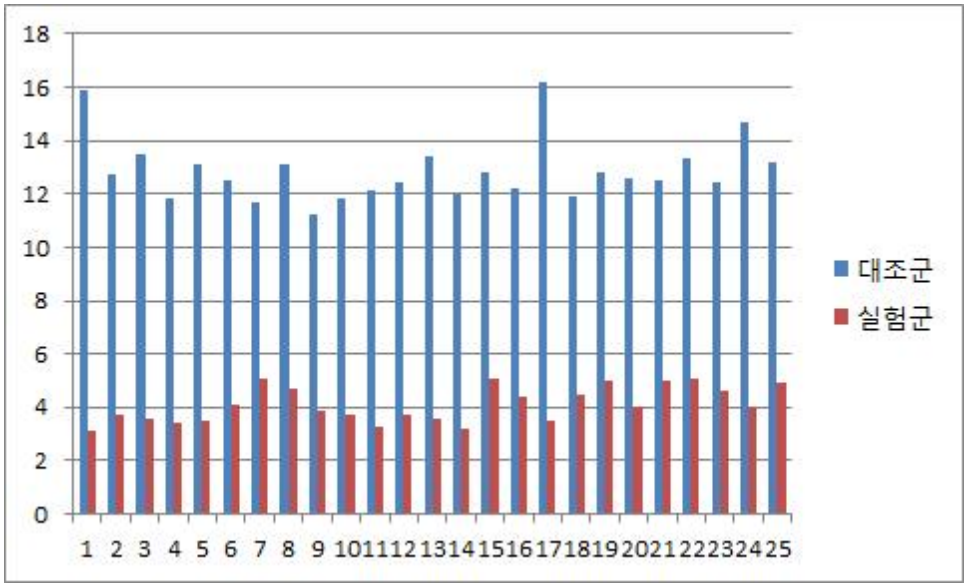

표 1. 실험 결과

실험결과	대조군	실험군
평균(s)	12.872	4.108
표준편차	1.202608276	0.67139159
최대 지연시간(s)	16.2	5.1

표 2. 제안한 기법과 기존 안드로이드의 성능 비교

광원 조절에 걸리는 시간을 각각 25회에 걸쳐 측정하였다. 위의 그림은 결과를 도표로 나타낸 것이다. 표에서 알 수 있듯이 제안한 기법에 적용된 경우는 기존 안드로이드 스마트폰에서의 결과보다 대부분 빠르게 측

정되었다. 대조군의 경우 반응시간이 평균 12.872s로 측정되었고 실험군에서는 평균 4.108s의 반응시간을 보였다. 이는 대조군에 비해 약 31.91%로 감소한 것이다.

제안한 기법은 여러 응용이 동시에 수행되는 상황에서 압도적인 성능향상을 보였다. 실험에서 사용한 응용은 CPU를 매우 많이 요구하는 응용이기 때문에 제안한 기법으로 인한 성능향상이 극단적으로 나타난 것으로 볼 수 있다.

제 6 장 결론

안드로이드의 고질적인 문제인 사용자 반응성 향상을 위해 특정 상황에서 발생할 수 있는 터치 이벤트 반응 지연에 대해 문제를 정의하였다. 이 문제를 해결하기 위해 안드로이드 내부 터치 이벤트 처리 과정과 리눅스 커널의 태스크 스케줄러인 CFS에 대해서 분석하였다. 이를 이용하여 여러 응용이 동시에 수행되는 상황에서 발생할 수 있는 반응 지연을 해결할 수 있는 연쇄적 우선순위 부스트 기법을 제안하였다.

제안한 기법은 몇 가지 설계 요소를 고려하여 구현하였다. 제안한 기법은 먼저 터치 이벤트 처리 과정에서 실행되는 시스템 스레드들의 우선순위를 상승시키고 사용자 응용이 생성할 수 있는 자식 스레드의 우선순위를 상향 조절하여 background 응용으로 부터 받는 영향을 최소화하였다.

또한 실제 상용 스마트폰에 적용하여 그 유효성을 검증하였다. 실험 결과 기존 안드로이드 운영체제에 비교해 약 31.91%로 사용자 반응성이 감소된 결과를 보였다.

참고 문헌

[1] Google I/O 2012, https://developers.google.com/events/io/

[2] Zdnet, http://www.zdnet.co.kr/news/news_view.asp?artice_id=20120912104317

[3] 배선욱, 김정한, 엄영익, "모바일 OS 환경의 사용자 반응성 향상 기법", 한국정보과학회

[4] Sungju Huh, Jonghun Yoo and Seongsoo Hong, "Improving Interactivity via VT-CFS and Framework-assisted Task Characterization for Linux/Android Smartphones", RTCSA 2012

Abstract

After the first release at 2008, Android has grown up and now it is widely used in many mobile devices. One of Android's the most attractive features is the support of multi-tasking. Multi-tasking improves user-interactivity in mobile devices. However, a reaction to user's touch may be postponed when many tasks are executed concurrently. This delay is mostly caused by competition among tasks to occupy CPU.

This paper presents a chained priority boosting which takes care of the user interactive task first. This mechanism promotes the priority of system threads which handle user's touch event, background threads as well as a foreground thread. Android application framework identifies touch event handling system threads, background thread and tells information of these threads to Linux kernel when user's touch occurs. Then Kernel promotes the priority of these threads so the response time can be reduced.

To verify usefulness of the proposed solution, the proposed solution is implemented on Galaxy Nexus and compared with legacy Android. Experimental results show that the proposed solution reduces the response time by 30%.

Keywords: User interactivity, Android, CFS, Priority boosting

Student number: 2011-20907