안드로이드 기반 스마트폰의 사용자 응답성 향상을 위한 프레임워크 지원 우선순위 부스트 기법

손용석[1], 허승주[1], 유종훈[2], Richard Taylor[2], 홍성수[1, 2]
[1]서울대학교 융합과학기술대학원 지능형융합시스템학과
[2]서울대학교 전기컴퓨터공학부
{ysson, sjhuh, jhyoo, taylorr, sshong}@redwood.snu.ac.kr

Framework-assisted Priority boosting for Improving Interactivity of Android Smartphones

Yongseok Son[1], Sungju Huh[1], Jonghun Yoo[2], Richard Taylor[2], Seongsoo Hong[1, 2]
[1]Department of Intelligent Convergence Systems, Graduate School of Convergence Science and Technology, SNU
[2]School of Electrical and Computer Engineering, SNU

요 약

최근 안드로이드 플랫폼을 탑재한 스마트폰이 널리 보급되면서 안드로이드 플랫폼에 대한 관심은 더욱 커지고 있다. 하지만 안드로이드 스마트폰은 종종 양질의 사용자 응답성을 제공하지 못하는 것으로 알려져 있다. 이는 안드로이드 상에서 대화형 태스크가 다른 태스크와 구별되지 않고 동일한 우선순위로 스케줄링 되기 때문에 사용자 입력을 처리하는 동안 여러 번의 선점을 당해 긴 응답시간을 초래할 수 있기 때문이다. 이 논문은 안드로이드 스마트폰의 사용자 응답성 향상을 위해 프레임워크 지원 우선순위 부스트 기법을 제시한다. 제안된 기법은 프레임워크 레벨에서 대화형 태스크를 식별하고 이를 커널에게 전달하며, 커널 레벨에서는 식별된 태스크의 우선순위를 선별적으로 부스트 시킴으로써 선점 없이 사용자 입력을 처리할 만큼 충분한 시간을 보장해 준다. 실험 결과는 제안된 기법이 기존 시스템보다 최대 22% 단축된 응답 시간을 보여 제안된 기법의 효용성을 입증하였다.

1. 서 론

안드로이드는 현재 가장 널리 사용되는 스마트폰 소프트웨어 플랫폼임에도 불구하고 여전히 사용자 응답성 측면에서 부정적인 평가가 보고되고 있다 [1]. 대화형 응용태스크의 성능은 사용자 입력을 처리하여 화면에 그 결과를 표시하는 응답시간으로 정량화될 수 있다. 이 시간 동안 커널은 입력 장치로부터 발생한 인터럽트를 처리하고, 응용 태스크는 이를 전달받아 최종적으로 출력 장치에 결과를 표시한다. 스마트폰 사용자는 이 과정이 매우 짧은 시간 내에 완료될 것을 예상한다.

그런데 수많은 태스크가 동시에 수행되는 안드로이드 스마트폰에서 짧은 응답 시간을 보장하는 것은 매우 어려운 문제이다. 대화형 태스크는 CPU를 차지하기 위해 통상 수십 개의 다른 태스크들과 경쟁해야 하며 이로 인한 간섭이 저조한 사용자 응답성의 주요 원인으로 지적되어 왔다.

안드로이드에서 태스크들은 리눅스의 기본 스케줄러인 Completely Fair Scheduler(CFS)에 의해 스케줄링 된다. CFS는 주목적은 각 태스크에게 공정한 CPU 시간을 할당하는 것이다. 이를 위해 CFS는 각 태스크에게 가중치에 비례하는 time slice를 할당하고 이 시간 동안 선점 없이 수행될 수 있도록 보장한다 [2]. CFS는 다수 사용자가 컴퓨팅 자원을 공평하게 나누어 사용해야 하는 서버 환경에서 특히 유용하다.

그러나 CFS는 사용자 응답성 측면에서는 만족스러운 성능을 발휘하지 못하는 것으로 알려져 있다 [3][4]. 이는 CFS가 대화형 태스크를 식별하지 않아 대화형 태스크에게 다른 태스크에 비해 CPU 자원을 보다 유리하게 할당해 주는 조치를 취할 수 없기 때문이다. 그 결과 대화형 태스크는 사용자 입력 처리 중 time slice를 소진하게 될 때마다 다른 태스크에게 선점 당하여 긴 응답시간을 초래한다.

본 논문에서는 이 문제를 해결하기 위해 프레임워크 지원 우선순위 부스트 기법을 제시한다. 제안된 기법은 프레임워크 레벨에서 대화형 태스크를 식별하고 이를 커널에게 전달하며, 커널 레벨에서 식별된 태스크의 우선순위를 임시적으로 높여 줌으로써 사용자 입력을 처리하기 위해 충분한 time slice를 할당 받을 수 있도록 한다. 우리는 제안된 기법을 안드로이드 Froyo 기반 개발 보드 상에 구현하였다. 실험 결과 대화형 태스크의 응답시간이 기존 시스템에 비해 최대 22% 단축됨을 보여 제안된 기법의 효용성이 입증되었다.

이 논문의 나머지 부분은 다음과 같이 구성된다. 2장은 안드로이드의 입력 이벤트 처리과정을 분석하고 3장에서 긴 선점 지연시간 문제를 정의한다. 4장에서 제안된 기법을 설명하고 5장에서 실험 결과를 보이며 6장에서 논문의 결론을 맺는다.

2. 안드로이드의 입력 이벤트 처리 과정 분석

이 장에서는 제안된 기법의 이해를 돕기 위해 안드로이드에서 사용자가 발생시키는 입력 이벤트가 처리되는 과정을 분석한다. 그림 1은 안드로이드의 입력 이벤트 처리를 위해 수행되는 리눅스 커널, 시스템 서버, 그리고 응용간의 상호 작용을 보인다. 시스템 서버는 응용에게 각종 서비스를 제공하기 위해

수행되는 프로세스로서 각 서비스마다 하나의 스레드를 생성한다. 이 중 입력 이벤트 처리를 담당하는 스레드는 *InputDeivceRead*와 *InputDeivceDispatcher*이며, 스마트폰 스크린에 출력을 담당하는 스레드는 *SurfaceFlinger*이다. 다음은 안드로이드에서 사용자 입력 이벤트를 처리하는 네 단계를 보인다.

① 커널은 입력 장치가 발생시키는 인터럽트를 디바이스 드라이버로 전달하고 이는 다시 리눅스 입력 계층을 통해 안드로이드의 시스템 서버로 해당 이벤트를 전달한다.
② 시스템 서버의 *InputDeviceRead*는 커널이 발생시키는 모든 타입의 입력 이벤트를 큐에 저장한다. *InputDeviceDispather*는 큐에 저장된 이벤트를 타입 별로 분류한 후, 이벤트를 기다리고 있는 응용 태스크에게 전달한다. 이 때 시스템 서버와 응용 간 통신에 Binder IPC가 사용된다.
③ 응용 태스크는 이벤트를 처리하고 화면에 표시할 이미지를 우선적으로 메모리에 저장한 후, 그 결과를 *SurfaceFlinger*에게 Binder IPC를 사용하여 전달한다.
④ *SurfaceFlinger*는 전달받은 데이터를 커널의 frame buffer에 저장하고, 그 결과 사용자는 입력에 대한 응답을 확인할 수 있다.

3. 문제 정의

이 논문에서는 사용자 응답성을 정량적으로 평가하기 위해 사용자 입력의 응답 시간을 척도로 사용한다. 응답 시간이란 스마트폰이 입력 장치에 이벤트가 발생한 시점부터 화면에 결과가 표시되기까지 걸리는 총 소요 시간으로 정의된다. 임의의 입력 처리 응용 태스크 τ의 응답 시간 RT(τ)는 다음과 같이 세 개의 구성 요소로 이루어진다.

$$RT(\tau) = T_S + T_P + T_E \quad (1)$$

이 때 T_S, T_P, T_E는 각각 스케줄링 지연시간, 선점 지연시간 그리고 태스크의 수행 시간을 나타낸다. 스케줄링 지연시간은 태스크가 wake-up되는 시점부터

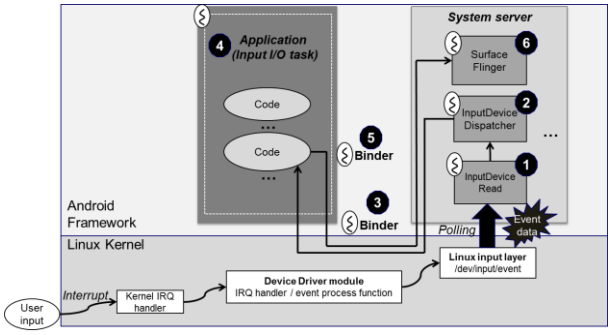

그림 1. 안드로이드 입력 이벤트 전달 경로.

CPU에 할당되어 실제로 첫 번째 명령어가 수행되는 시점까지의 시간이다. 선점 지연시간은 태스크가 입력 처리 도중 다른 태스크들에 의해 선점되는 총 시간이다. 한편 수행시간 T_E는 *InputDeviceRead*를 비롯한 시스템 서버 태스크들과 τ의 수행시간을 합한 것이다.

이 중 T_S는 임의의 태스크에게 할당될 수 있는 최대 time slice의 길이로 바운드되며 T_E는 입력 처리 응용 태스크의 구현에 의존한다. 반면, T_P는 τ가 자신에게 부여된 time slice를 소진한 후, 다른 태스크에게 선점 당할 때 마다 증가한다.

본 논문에서는 선점지연시간의 감소를 통해 안드로이드 스마트폰의 사용자 응답성 향상을 달성하고자 한다. 안드로이드에서 높은 선점 지연시간이 발생하는 주된 원인은 태스크 스케줄러인 CFS가 입력 처리 응용 태스크와 다른 태스크들을 구별하지 않기 때문이다. 안드로이드에서 모든 응용 태스크들은 동일한 가중치(nice value 0)를 부여 받는다. 따라서 CFS는 이 태스크들에게 동일한 time slice를 할당하고 스케줄링을 수행한다. 불행히도, CFS가 한 태스크에게 부여하는 time slice는 입력을 처리하기에 충분하지 않다. 따라서 입력 처리 도중 여러 번의 선점을 당해 결과적으로 높은 선점 지연시간가 야기된다.

4. 프레임워크 지원 우선순위 부스트 기법

본 장에서는 안드로이드의 선점 지연시간을 줄이기 위한 프레임워크 지원 우선순위 부스트 기법에 대해 설명한다. 우리의 기법은 크게 두 단계로 이루어진다. 첫째, 프레임워크 레벨에서 입력 처리 응용 태스크가 식별되며, 둘째, 커널 레벨에서 식별된 태스크의 우선순위가 임시로 상향 조정된다.

제안된 기법의 핵심 아이디어는 입력 처리 응용 태스크를 런타임에 효과적으로 식별하고 우선순위를 부스트 하는 것이다. 불행히도 커널 레벨에서 입력 처리 응용 태스크를 구별하는 것은 어려운 일이다. 실제로 리눅스의 O(1) 스케줄러는 확률과 경험에 따른 식별을 시도하였으나 종종 오동작하여 태스크 기아현상 등의 문제가 발생한다고 보고되었다 [5].

우리는 안드로이드 프레임워크에서 사용자 입력에 대한 이벤트가 반드시 *InputDeviceDispather*를 통해 입력 처리 응용 태스크에게 전달되어야 한다는 사실에 착안하여 프레임워크 레벨에서 입력 처리 응용 태스크를 식별하고 그 결과를 커널에게 전달한다. 커널은 식별된 태스크의 우선순위는 일정수준 부스트 한다. 결과적으로 입력 처리 응용 태스크에게 긴 time slice가 할당된다. 그림 2는 제안된 기법을 보인다. 2장에서 언급했듯이, 사용자 입력에 대한 이벤트는 */InputDeviceDispatcher*와 binder IPC를 통해 입력 처리 응용 태스크에게 전달된다. 우리는 새로운 시스템 콜을 추가하여 입력 처리 응용 태스크의 ID를 커널에게 전달

하도록 *InputDeivceDispatcher*를 확장한다.

커널은 프레임워크로부터 입력 처리 응용 태스크의 ID를 전달받아 해당 태스크의 우선순위를 부스트 시킨다. 구체적으로, 입력 처리 응용 태스크가 wake-up 될 때, 커널은 프레임워크로부터 전달된 스레드 ID와 wake-up된 태스크의 스레드 ID를 비교한다. 만일 두 스레드 ID가 일치하면, 커널의 스케줄러인 CFS는 해당 태스크를 입력 처리 응용 태스크로 인식하고, 일시적으로 우선순위를 높여준다. 입력 처리 응용 태스크가 종료되거나 blocking이 되면, 다시 원래의 우선순위로 낮춰준다. 제안된 기법은 입력 처리 응용 태스크에게 높은 우선순위를 할당하기 때문에 다른 태스크들의 성능에 영향을 끼칠 수 있다. 하지만, 사용자 응답성이 중요시되는 모바일 환경에서는 대화형 태스크가 더욱 중요하기 때문에 다른 태스크들의 성능 저하는 감수될 수 있다.

5. 실험 및 검증

본 장에서는 제안된 기법에 의한 응답시간 단축을 검증하기 위한 실험 결과를 보인다. 우리는 제안된 기법을 안드로이드 2.2 Froyo와 리눅스 2.6.32 커널이 탑재된 Nvidia Tegra2 보드 위에 구현하고 drawing 응용인 bord[6]의 응답 시간을 측정하였다. 응답시간 측정에는 커널 레벨 프로파일링 도구인 kernel shark[7]이 사용되었다.

우리는 drawing application인 bord[6]을 두 개의 CPU-intensive 태스크와 동시에 실행하였다. 기존 안드로이드 시스템에서 모든 응용 태스크는 우선순위 120을 부여 받기 때문에 우리는 CPU-intensive 태스크의 우선순위를 모두 120으로 설정하였다. 이때, 낮은 숫자는 보다 높은 우선순위를 나타낸다. 우리는 제안된 기법이 부스트 시키는 태스크의 우선순위를 115, 110, 105, 100으로 변경해가며 실험하였다.

그림 3은 제안된 기법을 도입함에 따른 응답시간 감소를 상향되는 우선순위를 변경해 가며 측정한 결과이다. 실험 결과에서 볼 수 있듯이, 입력 처리 응용 태스크의 우선순위를 높여 줄수록 응답시간이 감소하는 것을 볼 수 있다. 입력 처리 응용 태스크의 우선순위를

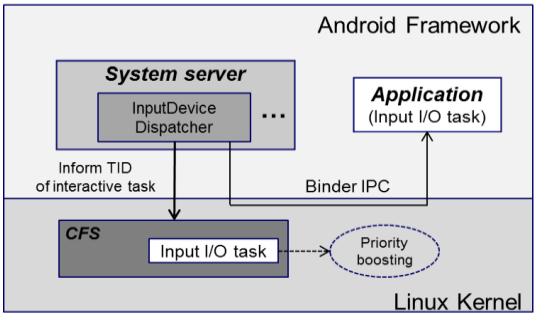

그림 2. 프레임워크 지원 우선순위 부스트 기법

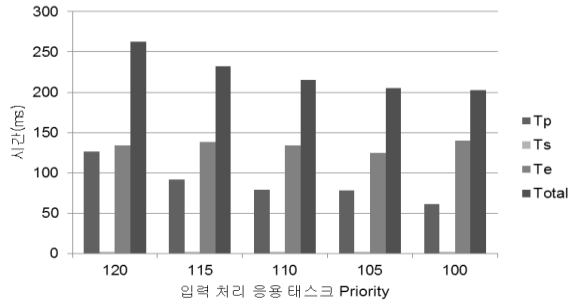

그림 3. 부스트 된 우선순위에 따른 응답시간 감소.

100으로 부스트 시켰을 때 응답시간이 기존 시스템에 비해 최대 22% 감소하였다.

6. 결론

본 논문에서는 안드로이드의 저조한 사용자 응답성 문제를 분석하고 이를 해결하기 위한 프레임워크 지원 우선순위 부스트 기법을 제안하였다. 이를 위해 먼저 안드로이드의 입력 이벤트 처리 과정을 분석하고, 이어서 응답시간에 가장 큰 영향을 끼치는 지연 요소가 선점 지연시간임을 밝혔다. 제안된 기법은 런타임에 입력 처리 응용 태스크를 프레임워크 레벨에서 식별하고 커널 레벨에서 우선순위를 높임으로써 선점 지연시간을 효과적으로 감소시킨다. 실험 결과는 제안된 기법이 기존 시스템 대비 최대 22% 단축된 응답시간을 보였다.

참고 문헌

[1] http://techland.time.com/2011/12/07/is-android-doomed-to-lag-more-than-ios/

[2] I. Molnar. The Completely Fair Scheduler, http://people.redhat.com/mingo/cfs-scheduler/.

[3] L. A. Torrey, J. Coleman, and B. Miller, “A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler” Software: Practice and Experience, vol. 37(4), pp. 347~364, 2007.

[4] S. Wang, Y. Chen, W. Jiang, P. Li, T. Dai, and Y. Cui, “Fairness and Interactivity of Three CPU Schedulers in Linux” Proceeding of RTCSA, pp. 172~177, August, 2009.

[5] K. Salah, A. Manea, S. Zeadally, and J.M. Alcaraz Calero, "On Linux starvation of CPU-bound processes in the presence of network I/O", Proceeding of Computers & Electrical Engineering, 2011, 37, (6), pp. 1090-1105.

[6] http://www.pixle.pl/bord/

[7] http://rostedt.homelinux.com/kernelshark/