

(KCC2015 우수논문)리눅스 기반 모바일 기기에서 사용자 응답성 향상을 위한 프레임워크 지원 선별적 페이지 보호 기법

(Framework-assisted Selective Page Protection for Improving Interactivity of Linux Based Mobile Devices)

김 승 준[†]
(Seungjune Kim)

김 정 호[‡]
(Jungho Kim)

홍 성 수[§]
(Seongsoo Hong)

요약 스마트폰과 같은 모바일 기기가 널리 보급됨에 따라 사용자들은 모바일 기기 응용들을 사용하면서 빠른 응답성을 제공받기를 바란다. 하지만 모바일 기기 응용들은 종종 사용자가 기대하는 수준의 응답성을 제공하지 못한다. 응답성을 저해하는 주 원인들 중 하나는 과도한 페이지 폴트 발생에 따른 대화형 태스크 수행의 지연이다. 이는 대화형 태스크의 상주 페이지(resident page)들이 비대화형 태스크와의 페이지 캐시 경쟁에 의해 더욱 빈번히 희생될 페이지(victim page)으로 선정되어 스토리지로 쫓겨나기 때문이다. 이 논문은 이러한 문제를 해결하기 위해 프레임워크 지원 선별적 페이지 보호 기법을 제시한다. 제안한 기법은 프레임워크 레벨에서 대화형 태스크를 식별하고 이를 커널에 전달하여 페이지 replacement 시에 대화형 태스크의 페이지를 보호하고, 사용자 입력 처리 중에 발생하는 페이지 폴트를 줄인다. 실험 결과 제안된 기법은 기존 시스템에 비해 페이지 폴트 횟수를 37% 감소시켰고, 응답시간을 11% 단축할 수 있었다.

키워드 : 사용자 응답성, 리눅스, 모바일 기기, 페이지 캐시

Abstract While Linux-based mobile devices such as smartphones are increasingly used, they often exhibit poor response time. One of the factors that influence the user-perceived interactivity is the high page fault rate of interactive tasks. Pages owned by interactive tasks can be removed from the main memory due to the memory contention between interactive and background tasks. Since this increases the page fault rate of the interactive tasks, their executions tend to suffer from increased delays. This paper proposes a framework-assisted selective page protection mechanism for improving interactivity of Linux-based mobile devices. The framework-assisted selective page protection enables the run-time system to identify interactive tasks at the framework level and to deliver their IDs to the kernel. As a result, the kernel can maintain the pages owned by the identified interactive tasks and avoid the occurrences of page faults. The experimental results demonstrate the selective page protection technique reduces response time up to 11% by reducing the page fault rate by 37%.

Key words : Linux, interactivity, mobile device, page cache

* 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT연구센터 육성 지원사업의 연구결과로 수행되었음(IITP-2015-(H8501-15-1015))

* 이 논문은 KCC2015에서 ‘리눅스 기반 모바일 기기에서 사용자 응답성 향상을 위한 프레임워크 지원 페이지 보호 기법’의 제목으로 발표된 논문을 확장한 것임

[†] 학생회원 : 서울대학교 융합과학기술대학원 지능형 융합시스템학과 sjkim@redwood.snu.ac.kr

[‡] 학생회원 : 서울대학교 융합과학기술대학원 지능형 융합시스템학과 jhkim@redwood.snu.ac.kr

[§] 종신회원 : 서울대학교 전기정보공학부 교수 sshong@redwood.snu.ac.kr

논문접수 : 2015년 8월 3일
심사완료 : 2015년 9월 21일

Copyright©2015 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: xxxx 제 xx권 제x호(2015.xx)

1. 서론

스마트폰 보급 됨에 따라 사용자들이 고사양, 고성능 응용들을 요구하고 있다[1]. 특히 이러한 응용들의 응답성은 사용자 경험에 있어서 중요한 요소들 중 하나이다. 스마트폰 제조사들은 모바일 기기의 사용자 응답성을 개선시키기 위해 많은 노력을 하고 있다. 하지만 이러한 노력에도 불구하고 여전히 양질의 응답성을 제공받지 못하는 경우가 보고되고 있다[2].

고사양 응용들은 대용량 컨텐츠들과 라이브러리를 사용하기 때문에 빈번하게 스토리지를 사용한다. Linux 커널에서 스토리지에 대한 접근은 요구 페이징(demand paging)에 의해 처리 된다. 응용이 요청한 페이지가 페이지 캐시에 부재한 경우, 페이지 폴트 핸들러가 페이지 폴트를 발생시켜 해당 페이지가 페이지 캐시에 적재된다. 특히, 메모리 제약이 심한 모바일 기기에서는 페이지 캐시의 크기가 데스크탑 환경에 비해 작기 때문에

페이지 폴트 발생 빈도가 높아진다. 실험을 통해 확인한 결과 사진 갤러리 응용은 USB와 같은 비대화형 응용의 I/O 작업이 있을 때, 최악의 경우 약 10000 번 이상의 페이지 폴트가 발생한다. 따라서 사용자 응답성 개선에 있어서 페이지 캐시 관리를 최적화하여 페이지 폴트 발생 횟수를 줄이는 것은 중요하다.

페이지 캐시 적중률을 높이기 위한 페이지 관리 정책은 선페이징(prepaging) 정책과 페이지 교체(page replacement) 정책으로 나뉜다. 리눅스에서는 선페이징 정책으로 readahead 메커니즘을 제공한다. 파일 페이지를 읽을 경우 순차적으로 읽는 패턴이 있음을 가정하고 읽은 파일 페이지 이후의 일부 파일 페이지를 순차적으로 미리 메모리에 적재함으로써 페이지 캐시 적중률을 높인다[3]. 페이지 교체 정책은 기본적으로 LRU-2(least recently used)에 기반한다. 이를 위해 커널은 모든 프로세스의 작업 세트(working set)를 포함하는 active_list와 물리 메모리 회수 대상인 남은 페이지들을 포함하는 inactive_list를 관리한다. Johnson et al.에 따르면 이 정책은 다른 LRU-k에 비해 5~10% 정도의 높은 페이지 캐시 적중률을 갖는다[4]. LRU-k는 페이지의 최근 k번의 참조 정보를 토대로 희생될 페이지를 결정하는 페이지 교체 정책이다.

그러나 이러한 리눅스의 페이지 캐시 정책은 사용자 응답성 측면에 있어서 한계가 있다. 입력 처리 과정에서 사용자와 상호작용하는 대화형 태스크의 페이지들이 다른 페이지들과 식별되지 않기 때문에 페이지 캐시 자원을 유리하게 할당해줄 수 없기 때문이다. 결과적으로 사용자 입력을 처리하는 과정 도중에 대화형 프로세스의 페이지들이 다른 페이지들에 의해 페이지 아웃 당하고 페이지 폴트를 발생시키며 다시 적재 되는 긴 응답시간을 초래한다.

본 논문에서는 이 문제를 해결하기 위해 프레임워크 기반 선별적 응용 페이지 보호 기법을 제시한다. 제안된 기법은 대화형 태스크를 프레임워크 레벨에서 식별하고, 커널로 해당 정보를 전달한다. 그리고 커널은 프레임워크로부터 전달받은 대화형 태스크 정보를 이용하여 대화형 태스크의 페이지가 희생될 페이지로 선정되는 것을 방지한다. 이에 따라 대화형 태스크의 페이지 폴트 발생이 줄어들게 된다. 우리는 제안된 기법을 안드로이드 KitKat 4.0 기반과 리눅스 커널 3.4가 탑재된 넥서스 5S 스마트폰에 구현하였다. 실험결과 기존 시스템 대비 페이지 폴트 횟수를 37%, 응답 시간 11%를 단축시킴으로써 제안된 기법의 효용성을 입증하였다.

이 논문의 나머지 부분은 다음과 같이 구성된다. 2 장에서는 리눅스의 페이지 캐시 관리 메커니즘을 분석한다. 3 장은 사용자 응답성 및 문제를 정의하고 4 장에서 제안된 기법을 설명한다. 그리고 5 장에서 실험 결과를 보이며 6 장에서 관련 연구를 설명한다. 그리고 7 장에서 논문의 결론을 맺는다.

2. 리눅스에서의 페이지캐시 관리 메커니즘

페이지 캐시는 디스크 I/O 작업을 통해서 메모리에 적재된 페이지들 중에서 빈번하게 사용될 것으로 선별된 페이지를 유지하는 메모리 공간이다. 이를 통해 같은 페이지에 대한 디스크 I/O 작업 횟수를 줄여 메모리 접근 시간(memory access time)을 줄이는 것이다. 이 장의 나머지 절에서는 본문의 이해를 돕기 위해 리눅스에서의 페이지캐시 구조 및 관리 메커니즘에 관해 개괄적으로 설명하고 페이지 교체 메커니즘과 Lumpy 회수(reclamation) 메커니즘에 대해 상세히 기술한다.

2.1 페이지캐시 구조 및 관리 메커니즘 개관

이 절에서는 우선 페이지캐시의 데이터 구조를 설명하고 이를 관리하기 위한 리눅스의 동작 메커니즘을 설명한다. 그림 1은 리눅스에서의 페이지 교체 흐름을 나타낸다.

페이지 캐시에 속한 페이지는 active_list와 inactive_list라는 두 개의 LRU 리스트들 중 하나에 속하게 된다. active_list에 속한 페이지들은 최근에 접근된 페이지를 포함하는 LRU 리스트이며 페이지 회수 작업 시에 희생될 페이지로 선정되지 않고 메모리에 유지된다. inactive_list는 한동안 접근되지 않은 페이지를 포함하는 LRU 리스트로 메모리 회수 작업의 대상 페이지들이 된다. 페이지의 참조 여부를 확인하기 위해 페이지 마다 참조 비트(reference bit)가 존재한다. 이 비트는 리눅스에서 struct page 구조체마다

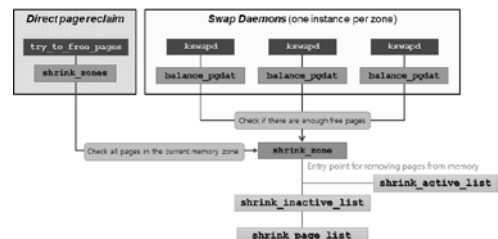


그림 1 리눅스에서의 페이지 교체 흐름도
Fig 1. Flow diagram of page replacement in Linux

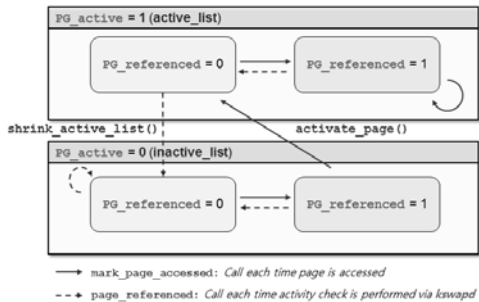


그림 2 LRU리스트간의 페이지 플래그에 따른 페이지 이동

Fig 2. Moving page between LRU lists with respect to page flag

PG_referenced 로 구현되어 있다.

페이지캐시를 관리하기 위한 동작 메커니즘은 페이지 삽입과 제거 동작으로 구분된다. 페이지 삽입 동작은 요구 페이지와 선편제에 의해 수행된다. 요구 페이지의 경우 mmap() 시스템 콜로 할당된 가상 메모리 공간에서 페이지들이 참조될 때 페이지 폴트 처리기를 통해 해당 페이지들이 페이지캐시에 삽입된다. 선편제의 경우에는 명시적, 내재적 작업으로 구분된다. 명시적 작업은 유저 레벨 코드가 직접 시스템 콜을 호출하여 페이지를 삽입하는 것을 말한다. 리눅스에서 이러한 작업에 관련된 시스템 콜로는 read()와 MAP_POPULATE 플래그를 인자로 받는 mmap() 이 있다. 내재적 작업은 리눅스의 readahead 작업을 통해 수행된다.

페이지 제거 메커니즘은 페이지 교체 메커니즘과 Lumpy 회수 메커니즘의 연계를 수행한다. 페이지 교체 메커니즘은 페이지캐시에서 희생될 페이지를 선정하며, Lumpy 회수 메커니즘은 할당 요청된 페이지의 크기가 2 개 이상일 때 그 희생될 페이지를 기준으로 연속된 페이지들을 추가로 희생될 페이지로 선정하여 메모리로 회수한다. 이러한 메커니즘들은 kswapd 라는 리눅스 커널 데몬과 커널 메모리 할당자에 의해 수행된다. Kswapd 는 주기적 또는 사건 기반(event-driven)으로 low watermark 를 체크하여 페이지 교체 메커니즘을 수행한다. 커널 메모리 할당자는 사건 기반으로 min watermark 를 체크하여 페이지 교체 메커니즘을 수행한다. 이러한 작업은 리눅스 커뮤니티에서 direct reclaim 으로 불린다. Low 와 min watermark 들의 값은 리눅스에 미리 정의되어 있는 임계값이며 그 수치는 물리 메모리 크기에 따라 달라진다. 2GB 의 시스템에서 그 값들은 각각 7,190KB, 5,752KB 이다[5].

2.2 페이지 교체 메커니즘

리눅스에서 페이지 교체 메커니즘은 페이지 구분 작업과 희생될 페이지 선택 작업으로 나뉜다.

페이지 구분 작업에서는 페이지캐시에서 메모리에 유지될 페이지들이 active_list 에 삽입되고, 그렇지 않은 페이지들은 inactive_list 로 삽입된다. 그리고 각 페이지들의 참조 비트 변경 시점에 따라 페이지들이 다른 리스트로 이동된다. 그림 2는 각 리스트간의 교체 메커니즘에 의한 페이지 이동을 나타낸다. 구체적으로, 최초로 삽입된 페이지는 inactive_list 에 존재하며 참조 비트는 0 이다. 이후에 그 페이지가 참조되어 참조 비트가 1 이 된 경우에 kswapd 가 주기적으로 해당 페이지를 active_list 로 이동함과 동시에 참조 비트를 0 으로 설정한다. 그 페이지가 한 번 더 참조되면 1 로 설정된다. 해당 비트는 LRU-2 메커니즘을 주기적으로 수행하는 kswapd 에 의해 0 으로 set 이 된다. 해당 페이지는 다음 kswapd 의 주기에서 여전히 참조 비트가 0 이면 inactive_list 로 이동한다.

희생될 페이지 선정 작업은 inactive_list 의 페이지들에서 LRU-2 정책에 따라 여러 개의 희생될 페이지들을 선택한다. 선택된 페이지들의 개수는 시스템의 자유 페이지(free page)의 개수가 high watermark 로 도달되기 위해 필요한 개수로 결정된다. high watermark 들의 값은 리눅스에 미리 정의되어 있는 임계 값이며 그 수치는 물리 메모리 크기에 따라 달라진다. 2GB 의 시스템에서 그 값은 8,628KB 이다[5].

2.3 Lumpy 회수 메커니즘

Lumpy 회수 메커니즘은 단편화된 물리 메모리에서 2 개 이상의 연속된 페이지들을 확보하기 위한 메커니즘이다. 이 메커니즘은 커널 메모리 할당자가 2 개 이상의 연속된 페이지 할당을 요청 받았을 때, 물리 메모리 부족으로 수행되는 페이지 교체 메커니즘에 연계되어 수행된다. 구체적으로, 이 메커니즘은 페이지 교체 메커니즘을 위해 inactive_list 에서 희생될

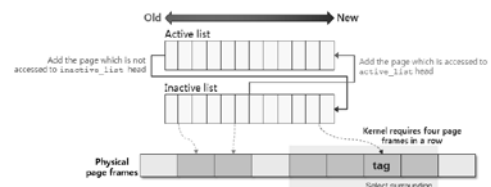


그림 3 LRU 리스트와 Lumpy reclamation
Fig 3. LRU lists and Lumpy reclamation

그리고 변경된 값은 커널의 FTI 에게 전달한다. 전달 방법은 Linux 커널에 시스템 콜을 추가하고 프레임워크레벨에서 호출하여 이것의 인자로 대화형 태스크 ID를 전달한다.

FTI는 해당 CPI의 시스템 콜 호출로 전달된 대화형 태스크 ID를 기록하여, 커널에서 대화형 태스크를 식별한다. 커널에서 추가된 시스템 콜이 프레임워크레벨에서 호출될 때마다 FTI는 대화형 태스크 ID를 인자로 전달받는다. 그리고 FTI는 현재 기록된 대화형 태스크 ID와 전달받은 태스크 ID를 기록한다. 그리고 커널에 추가된 대화형 태스크 ID 요청 함수가 호출되면 현재 기록되어 있는 대화형 태스크 ID를 반환한다.

Page identifier는 메모리에 적재된 각각의 페이지들의 구조체에 이를 요청한 태스크의 ID를 기록한다. 리눅스에서는 메모리에 적재된 페이지를 관리하기 위해 페이지프레임마다 struct page라는 페이지 구조체를 유지한다. 하지만 이 구조체로는 해당 페이지를 요청한 태스크를 구별할 수 없다. 따라서 이 구조체에 태스크 ID 멤버 변수를 추가하고 여기에 관련 태스크 ID를 기록한다.

IPP는 현재 FTI에 기록된 태스크 ID의 페이지가 희생될 페이지로 선정되지 않도록 보호한다. 기본적으로 IPP는 페이지 교체 메커니즘 수행 시 동작되며 추가로 Lumpy 회수 메커니즘이 수반되는 경우 추가 작업을 수행한다. IPP는 페이지 교체 메커니즘에서 inactive_list 내 희생될 페이지 선정 시, 해당 페이지의 구조체에 기록된 ID와 FTI에 기록된 ID를 비교하여 해당 페이지가 대화형 태스크의 페이지이면 희생될 페이지 선정에서 제외시킨다. 그림 1에서 inactive_list에서 희생될 페이지로 선정된 페이지들은 shrink_list()에서 페이지 타입 및 참조 비트를 검사하여 최종 희생될 페이지들을 선정 한 후 해당 페이지들을 페이지캐시에서 제거한다. 따라서 대화형 태스크의 페이지를 보호하기 위해 구현된 함수를 shrink_list()에서 호출한다. 해당 함수는 FTI에서 제공하는 대화형 태스크 ID 제공 함수를 호출하여 대화형 태스크 ID를 획득하여 해당 페이지를 판별한다. 그리고 shrink_list()에서 대화형 태스크의 페이지일 경우의 case 문을 추가하여 해당 페이지가 메모리에 유지될 수 있도록 한다. 또, 대화형 태스크 페이지의 참조 비트를 1로 설정하여 active_list로 포함될 수 있도록 한다.

Lumpy 회수 메커니즘으로 선정된 희생될 페이지들에 관해서 IPP는 비대화형 태스크의 메모리 할당 요청 시 reclamation 영역의 대화형 페이지의 존재 여부를

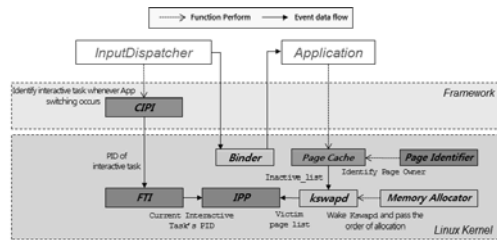


그림 5 프레임워크 지원 선별적 페이지 보호 기법
Fig 5. Framework assisted selective page protection technique

판단한다. 대화형 페이지가 존재할 경우 이 페이지를 제외한 연속된 페이지들을 희생될 페이지들로 선정한다. Lumpy 회수 메커니즘의 경우 태그 페이지를 중심으로 페이지를 검사하는데, 참조 비트를 고려하지 않는다. 따라서 이 메커니즘에서 페이지 검사를 하는 과정에서 대화형 페이지를 보호하기 위해 구현된 함수를 호출하여 대화형 태스크의 페이지를 식별한다. 그리고 Lumpy 회수 메커니즘에서 대화형 태스크의 페이지가 희생될 페이지로 식별된 경우의 희생될 페이지에서 제외된다.

시간이 지남에 따라 대화형 태스크와 관련된 페이지들이 IPP의 보호로 대부분의 페이지캐시 공간을 차지한다. 하지만 페이지 보호로 페이지캐시에 대화형 태스크의 페이지만 존재하게 된다면, 페이지캐시의 여유공간이 부족함에도 불구하고 페이지 회수를 할 수 없어 비대화형 태스크를 비롯한 대화형 태스크가 요청하는 페이지가 메모리에 적재되지 못하는 상황이 발생한다. 따라서 IPP는 비대화형 태스크의 페이지가 페이지캐시에서 모두 제거되고, watermark를 체크하여 여유 메모리공간이 부족하다고 판단될 경우 LRU-2 정책으로 대화형 태스크의 페이지를 제거한다.

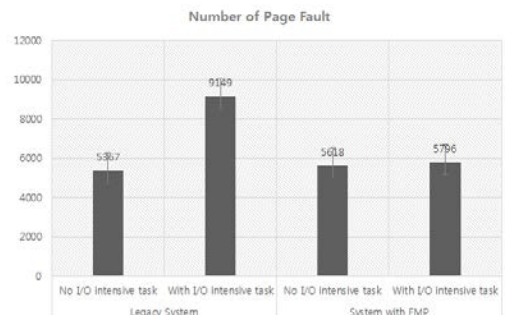


그림 6 기존 시스템과 FMP 기법을 적용한 페이지 폴트 수 측정 결과
Fig 6. Page fault count in legacy system and FMP system

5. 실험 및 검증

본 장에서는 제안된 기법을 검증하기 위한 실험과 그 결과를 보인다. 우리는 제안된 기법을 안드로이드 4.4 KitKat 과 커널 3.4 가 탑재된 구글 넥서스 5 스마트폰 상에서 구현하고 I/O 작업을 발생시키는 응용인 갤러리 응용의 응답시간과 페이지 폴트 횟수를 측정하였다. 응답시간 측정은 커널 레벨 프로파일링 도구인 KernelShark[7]를 사용하였고, 페이지 폴트 횟수 측정은 proc 파일 시스템[8]을 이용하여 측정하였다. 그리고 제안된 기법이 적용된 시스템의 전체적인 성능을 측정하기 위해 Antutu[9], PassMark[10] 벤치마크 응용을 사용하였다. 각 벤치마크는 CPU, RAM, 스토리지, I/O 등과 같은 성능을 측정하여 수치로 제공한다.

본 실험에서는 안드로이드에서 제공하는 기본 응용인 갤러리 응용과 I/O 집약적 태스크를 동시에 실행하였다. I/O 집약적 태스크는 지속적으로 I/O 를 발생시키는 태스크로서 본 실험에서는 USB 를 통해 스마트폰으로 대용량 파일을 전송하도록 요청하며 I/O 를 발생시킨다.

그림 6 은 기존 시스템과 제안된 기법인 framework-assisted selective memory protection(FMP)를 적용한 시스템에서 I/O 집약적 태스크를 수행시키기 전과 후의 페이지폴트 발생 횟수를 측정한 결과이다. 기존 시스템에서는 I/O 집약적 태스크가 있는 경우 페이지폴트 수가 약 41% 증가한다. FMP 기법을 적용시킨 시스템에서는 기존 시스템에서 I/O 집약적 태스크가 있는 경우에 발생하는 페이지 폴트 수보다 약 37% 감소하였다. 그림 7 은 기존 시스템과 FMP 가 적용된 시스템에서의 응답시간 측정 결과를 나타낸다. I/O 집약적인 태스크가 수행중인 경우 기존 시스템에 비해 FMP 가 적용된 시스템에서는 응답시간이 약 11% 단축되었다.

그림 8 은 Antutu, PassMark 벤치마크를 통해 기존

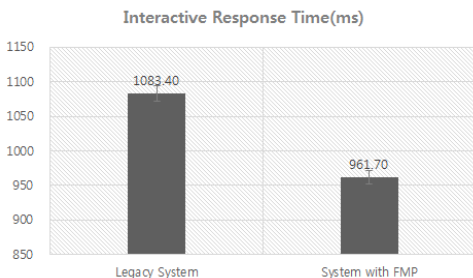


그림 7 기존 시스템과 FMP 기법을 적용한 응답시간 측정 결과

Fig 7. Interactive response time in legacy system and FMP system

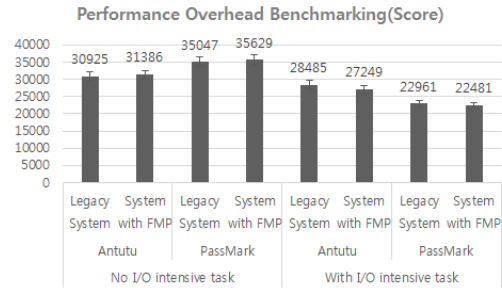


그림 8 벤치마크를 통한 성능 오버헤드 측정 결과
Fig 8. Performance overhead measured via benchmark

시스템과 제안된 기법이 적용된 시스템의 성능을 측정한 결과값이다. I/O 집약적 태스크가 없는 경우 성능 오버헤드는 각 시스템이 비슷한 성능을 나타낸다. I/O 집약적 태스크가 수행될 경우에는 FMP 기법이 적용된 시스템이 기존 시스템보다 3% 정도의 오버헤드를 가진다.

6. 관련 연구

모바일 환경에서의 사용자 응답성을 높이기 위한 선행연구가 활발히 진행되어 왔다. 이러한 기존 접근 방법들은 커널 레벨과 프레임워크 레벨로 나눌 수 있다. 커널 레벨에서는 I/O 발생 횟수를 최소화하고 I/O 스케줄러를 수정하여 사용자 응답성을 향상시키려 노력했다. 리눅스에서는 페이지 폴트로 인한 I/O 발생을 최소화하기 위해 readahead 메커니즘을 제공한다. 이 메커니즘을 이용하여 안드로이드에서 사용자 응답성 태스크의 페이지를 readahead 하여 페이지 폴트를 최소화하는 기법이 제안되었다[11]. 이 기법은 대화형 태스크를 커널에서 런타임으로 구분한다. 그리고 식별된 태스크의 페이지를 readahead 할 때 다른 태스크보다 더 많은 페이지를 메모리에 선포이징하여 대화형 태스크의 페이지 캐시 적중률을 높인다. 하지만 이 기법은 대화형 태스크를 찾기 위한 런타임 오버헤드가 발생하며, 페이지 교체 정책에서 대화형 태스크를 구분하지 않아 해당 페이지가 페이지캐시에서 제거될 수 있다.

커널에서 I/O 스케줄러는 I/O 의 특성에 따라 우선순위를 두어 요청을 처리한다. I/O 의 종류는 동기적(synchronous), 비동기적(asynchronous) I/O 가 있다. 동기적 I/O 는 해당 I/O 요청을 발생시킨 태스크가 I/O 요청이 수행되어 완료될 때까지 프로그램을 수행하지 않고 기다리는 I/O 이다. 비동기적 I/O 는 태스크가 I/O 요청을 발생시키고 I/O 요청의 완료와 상관없이

프로그램을 수행될 수 있는 I/O 이다. 비동기적 I/O 는 태스크의 수행에 영향을 주지 않으므로 커널에서는 동기적 I/O 요청에 우선순위를 주어 비동기적 I/O 보다 빠르게 처리한다. 하지만 안드로이드에서 비동기적 I/O 로 분류된 요청임에도 해당 I/O 가 완료될 때까지 응용이 블락당하여 응답시간이 지연되는 경우가 발생한다. 따라서 비동기적 I/O 로 분류된 요청과 응용 태스크의 의존도를 체크하여 해당 비동기적 I/O 의 우선순위를 높여 사용자 응답성을 높이는 기법이 제안되었다[12]. 이 기법은 비동기적 I/O 로 분류되었지만 이를 기다리는 응용태스크가 있는 경우, 해당 I/O 의 우선순위를 높여줌으로써 응답 시간이 지연되는 것을 막는다. 하지만 이 기법은 지연될 수 있는 상황을 방지하지만 대화형 태스크를 커널이 식별하지 않아 사용자 응답성을 보장하기에는 한계가 있다.

프레임워크 레벨에서도 빠른 사용자 응답성을 제공하기 위해 다양한 기법을 사용하였다. 대표적으로 안드로이드는 응용태스크를 빠르게 실행시켜 사용자 응답성을 개선시켰다[12]. Zygote 프로세스라는 프로세스를 생성하여 응용이 자주 사용하는 라이브러리 클래스들을 미리 초기화 한다. 그리고 각 응용들이 생성될 때 미리 초기화된 Zygote 프로세스를 통해 생성됨으로써 응용이 생성되는 시간을 단축한다[13].

안드로이드에서는 터치화면에서의 드로잉시간 단축을 위해 분리된 surface 라는 분리된 구역으로 나누어 렌더링하는 작업을 수행한다[14]. 여러 개의 surface 구역으로 나누어 갱신된 surface 만 새롭게 렌더링한다. 안드로이드에서 제공하는 서비스인 SurfaceFlinger 는 여러 개의 surface 들을 조합하여 프레임버퍼장치로 보냄으로써 전체 렌더링 시간이 줄어들어 사용자에게 향상된 응답성을 제공한다.

프레임워크의 지원을 받아 대화형 태스크를 커널에서 구분하여 대화형 태스크의 우선순위를 조정함으로써 사용자 응답성을 개선하는 기법도 제안되었다[15]. 사용자 응답 처리 과정에서 태스크 경쟁에 의해 대화형 태스크의 수행이 지연될 수 있다. 따라서 대화형 태스크의 우선순위를 높여주어 사용자 응답 처리 중 CPU 자원을 우선적으로 할당해줌으로써 사용자 응답성을 개선한다. 하지만 메모리에서 응답성 관련 태스크의 페이지를 보호하지 못하기 때문에 사용자 응답 처리 중 페이지폴트에 의해 지연되는 문제를 해결하지 못한다.

7. 결론

이 논문은 모바일 환경에서의 사용자 응답성이 저조해지는 상황을 파악하고 이를 해결하기 위해 프레임워크 지원 선별적 페이지 보호 기법을 제안하였다. 이를 위해 먼저 사용자 응답성과 태스크 체인을 정의하고, 이 과정에서 페이지 폴트가 응답시간을 지연시킨다는 것을 밝혔다. 제안된 기법은 프레임워크에서 사용자 입력 처리 태스크를 식별하고 이를 커널로 전달하여 해당 대화형 페이지를 보호함으로써 사용자 입력 처리 중 페이지 폴트 발생을 최소화한다.

선별적 페이지 보호는 리눅스의 페이지캐시에서 채택하는 페이지 교체정책과 Lumpy 회수를 모두 고려하여 보호한다. 각 상황에서 회수 되기 위해 희생될 페이지로 선정되는 페이지 중에서 대화형 태스크의 페이지를 식별하고 희생될 페이지 선정에서 제외시켜 해당 태스크의 페이지를 보호한다. 실험 결과 제안된 기법이 기존 시스템 대비 페이지 폴트 횟수는 약 37%, 응답시간은 11% 감소하였다.

참고 문헌

- [1] Klick Health 2013. [Online]. Available: <https://www.klick.com/health/news/blog/mhealth/mobile-apps-what-consumers-really-want>
- [2] BGR 2013. [Online]. Available: <http://bgr.com/2013/09/20/iphone-android-touch-screen-responsiveness>
- [3] W. Mauerer, Professional Linux Kernel Architecture, 1st Ed., pp. 970. Wolfgang-Mauerer, Canada, 2008.
- [4] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," Proc. VLDB Conf., pp. 297-306, 1994.
- [5] LWN 2011. [Online]. Available: <https://lwn.net/Articles/422291/>
- [6] Y. Son, S. Huh, J. Yoo and S. Hong, "Framework-assisted Priority boosting for Improving Interactivity of Android Smartphones," Journal of KIISE: Computer Systems and Theory, Vol. 39. No.6, pp. 380-386, 2012
- [7] eLinux 2011. [Online]. Available: http://elinux.org/images/6/64/Elc2011_rostedt.pdf
- [8] Wikipedia 2010. [Online]. <http://en.wikipedia.org/wiki/Procsfs>
- [9] Antutu 2015. [Online]. Available: <http://www.antutu.com/en/index.shtml>
- [10] PassMark software. 2015. [Online]. Available: <http://www.androidbenchmark.net/>
- [11] S. Bae, H. Song, C. Min, J. Kim, and Y. Eom, "EIMOS: enhancing interactivity in mobile operating systems," Computational Science and Its Applications-ICCSA, Springer Berlin Heidelberg, pp. 238-247, 2012.
- [12] D. Jeong, Y. Lee, and J. Kim, "Boosting quasi-asynchronous I/O for better responsiveness in mobile devices," Proceedings of the 13th USENIX Conference on File and Storage Technologies, USENIX Association, pp. 191-202, 2015.
- [13] D. Bornstein 2008. [Online]. Available: <https://sites.google.com/site/io/dalvik-vm-internals>
- [14] Android Open Source Project. [Online]. Available:

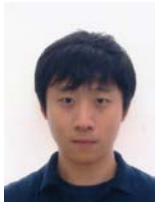
<https://source.android.com/devices/graphics/>

- [15] S. Huh, J. Yoo, and S. Hong, "Cross-layer resource control and scheduling for improving interactivity in Android," *Softw. Pract. Exper.*, doi: 10.1002/spe.2285. 2014.



김 승 준

2014년 숭실대학교 컴퓨터학부 졸업(학사). 2014년~현재 서울대학교 융합과학기술대학원 지능형융합시스템학과 석사과정. 관심분야는 내장형 시스템 소프트웨어, 소프트웨어 플랫폼, 실시간 운영체제



김 정 호

2010년 포항공과대학교 전기전자공학과 졸업(학사). 2013년 서울대학교 전기컴퓨터공학부 졸업(석사). 2013년~현재 서울대학교 융합과학기술대학원 지능형융합시스템학과 박사과정. 관심분야는 내장형 시스템 소프트웨어, 빅데이터 분산 처리 프레임워크



홍 성 수

1986년 서울대학교 컴퓨터공학과 (학사). 1988년 서울대학교 컴퓨터공학과 (공학석사). 1994년 University of Maryland, Department of Computer Science (공학박사). 1995년~현재 서울대학교 전기정보공학부 교수. 2008년~현재 가현신도리코 재단 석좌교수. 2009년~현재 스마트시스템연구소 연구소장. 2013년~현재 한국자동차공학회 사업이사. 2012년~2014년 서울대학교 융합과학기술대학원 부원장. 2012년~2014년 서울대학교 융합과학기술대학원 융합과학부장. 2006년~2012년 서울대학교 융합과학기술대학원 지능형융합시스템학과장. 2004년~2006년 서울대학교 내장형시스템연구센터 센터장. 관심분야는 내장형 실시간시스템 설계, 실시간 운영체제, 내장형 미들웨어, 실시간 시스템 설계 방법론, 소프트웨어 공학, 컴포넌트 기반 소프트웨어 설계