임베디드 시스템 가상화를 위한 가상 인터럽트 콘트롤러의 구현 및 성능 분석*

오수철, 안창원
한국전자통신연구원
e-mail : {ponylife, ahn}@etri.re.kr

Implementation and Performance Analysis of a Virtual Interrupt Controller for Embedded System Virtualization

Soo-Cheo Oh, Chang-Won Ahn
Electronics and Telecommunication Research Institute

요 약

본 논문은 ARM 기반 임베디드 시스템상에서 전가상화를 지원하는 가상 머신 모니터를 위한 가상 인터럽트 콘트롤러를 제안한다. 가상 인터럽트 콘트롤러는 실제 하드웨어 인터럽트 콘트롤러와 동일한 인터페이스를 가지며, 운영체제의 수정 없이 가상 인터럽트 콘트롤러를 하드웨어 인터럽트 콘트롤러와 동일한 방식으로 사용하는 환경을 제공한다. 본 논문에서는 가상 인터럽트 콘트롤러를 구현하고 성능을 측정하였다.

1. 서론

본 논문은 가상화된 임베디드 시스템에서 인터럽트 콘트롤러를 가상화(Virtualization)하는 메커니즘에 관한 것이다. 가상화 컴퓨터 시스템은 단일 하드웨어를 가진 컴퓨터 시스템에서 복수의 운영체제를 동시에 수행하는 것을 가능하게 한다. 이를 위해 컴퓨터 하드웨어 상에 가상 머신 모니터(VMM : Virtual Machine Monitor)가 탑재되고, 가상 머신 모니터상에 복수개의 운영체제가 탑재된다. 이를 위해 가상 머신 모니터는 가상의 하드웨어 환경을 운영체제에 제공한다. 컴퓨터 하드웨어 중 인터럽트 콘트롤러는 중앙처리장치와 입출력 장치에서 발생한 이벤트를 중앙처리장치에 전달하는 역할을 한다.

컴퓨터 시스템의 가상화 기법은 전가상화(Full Virtualization)[1]와 반가상화(Para-Virtualization)[1]로 구분될 수 있다. 전가상화는 가상 머신 모니터가 가상화에 방해가 되는 명령어들을 실행시간에 감지하여 에뮬레이션한다. 이를 통하여 운영체제의 수정을 필요로 하지 않는 장점이 있으나, CPU 자원을 많이 사용하는 단점이 있다. 반면 반가상화는 가상화에 방해가 되는 명령어들을 소스코드 수준에서 수정한다. 즉, 가상화에 방해가 되는 명령어들을 가상 머신 모니터에 대한 hypercall 로 대체하거나, 동일한 동작을 수행하는 다른 코드로 변환하는 것이다. 이를 통하여 전가상화에 비해서 빠른 동작 속도를 보여준다. 그러나, 게스트 OS 의 코드를 가상 머신 모니터에 맞게 수정해야 하는 문제점이 있다.

모바일 시스템에서 반가상화를 지원하는 가상 머신 모니터로는 VMware 사의 MVP[2], VirtualLogix 의 VLX[3], XenARM[4] 등이 있다. 전가상화를 지원하는 가상 머신 모니터로는 QEMU[5]를 들 수 있다.

본 논문에서는 전가상화 기반 가상 머신 모니터에서 임베디드 시스템의 인터럽트 콘트롤러를 가상화하여 각 운영체제에 제공하는 메커니즘을 제안한다. 가상 인터럽트 콘트롤러는 실제 하드웨어 인터럽트 콘트롤러와 동일한 인터페이스를 가지며, 운영체제의 수정없이 가상 인터럽트 콘트롤러를 하드웨어 인터럽트 콘트롤러와 동일한 방식으로 사용하는 환경을 제공한다.

2. 하드웨어 인터럽트 콘트롤러 구조

본 논문은 ARM 기반 임베디드 시스템을 대상으로 하고 있으며, ARM 사의 주요 코어에 포함되는 VIC (Vectored Interrupt Controller)를 대상으로 한다. 본 VIC 의 이름은 PL192[6]이다.

하드웨어로 제공되는 VIC 를 제어하는 방식은 프로세서의 주소공간으로 매핑된 인터럽트 콘트롤러의 내부 레지스터들에 대한 읽기/쓰기 동작을 수행함으로써 이루어진다. 인터럽트 콘트롤러에서 제공되는 주요 레지스터에 대한 설명은 <표 1>과 같다.

* 본 연구는 지식경제부의 IT 성장동력기술개발사업의 일환으로 수행하였음.
[KI002088, 공개 SW 기반 가상화 기술 개발]

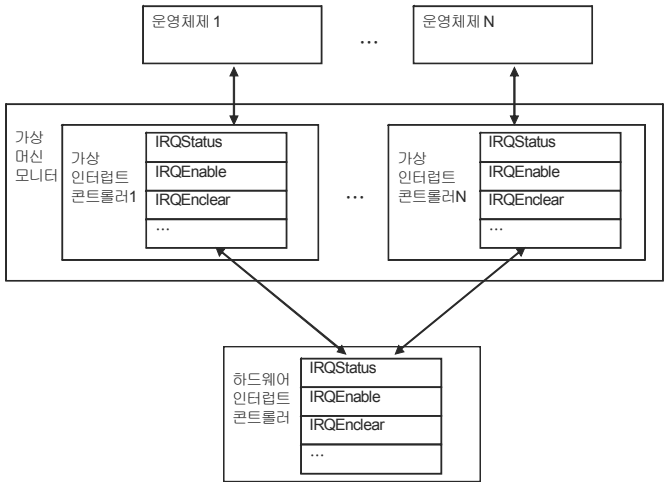
3. 가상 인터럽트 콘트롤러 구조

본 논문에서 제안하는 가상 인터럽트 콘트롤러의 구조는 <그림 1>과 같다. 임베디드 시스템에는 하드웨어 인터럽트 콘트롤러가 있으며, 가상 머신 모니터는 하드웨어 인터럽트 콘트롤러를 가상 인터럽트 콘트롤러로 만들어서 각 게스트 OS 에 제공한다. 가상 머신 모니터는 1 개의 게스트 OS 마다 1 개의 가상 인터럽트 콘트롤러를 제공한다. 따라서, 가상화된 시스템에서 수행되는 게스트 OS 는 하드웨어 인터럽트 콘트롤러가 아닌 가상 인터럽트 콘트롤러를 사용한다.

<표 1> PL192 의 주요 레지스터

레지스터	설명
IRQStatus	각 bit 들은 해당 인터럽트가 발생했는지 여부를 표시한다.
IRQEnable	각 bit 들은 해당 인터럽트가 unmask 되어 있는 상황을 보여준다. 또한 해당 bit 에 '1'을 write 함으로써 해당 인터럽트를 unmask 한다.
IRQEnclear	해당 bit 에 1 을 write 함으로써 해당 인터럽트를 mask 한다.
VectAddr	각 인터럽트를 위한 인터럽트 핸들러의 주소를 설정한다.
Address	인터럽트 발생시, 인터럽트 번호에 해당하는 VectAddr 를 표시한다.

하드웨어로 제공되는 인터럽트 콘트롤러를 제어하는 방식은 2 절에서 설명한 바와 같이 프로세서의 주소공간으로 매핑된 인터럽트 콘트롤러의 내부 레지스터들에 대한 읽기/쓰기 동작을 수행함으로써 이루어진다. 따라서, 인터럽트 콘트롤러를 가상화하기 위해서는 두가지 작업이 진행되어야 한다. 첫째, 운영체제에서 발생한 하드웨어 인터럽트 콘트롤러에 대한 접근을 가상 머신 모니터가 가로채야 한다. 둘째, 하드웨어 인터럽트 콘트롤러 내부의 레지스터 자원을 가상화하고 이를 사용하여 가로챈 인터럽트 콘트롤러에 대한 접근을 처리해야 한다.



<그림 1> 가상 인터럽트 콘트롤러 구조

첫째, 가상 머신 모니터가 운영체제의 하드웨어 인터럽트 접근을 가로채는 방법은 다음과 같다. 가상 머신 모니터상에서 수행되는 게스트 OS 는 가상-물리 주소 매핑을 가지고 있으며, 하드웨어 인터럽트 콘트롤러에 대한 매핑 정보도 포함하고 있다. 가상 머신 모니터는 부팅시에 게스트 OS 의 주소 매핑을 수정하여 하드웨어 인터럽트 콘트롤러에 대한 매핑 정보를 삭제한다. 이렇게 되면 게스트 OS 에서 하드웨어 인터럽트 콘트롤러에 대한 읽기/쓰기 동작을 할 때마다, ARM 프로세서의 data abort exception 이 발생한다.

가상화가 적용되지 않은 시스템에서는 exception vector table 을 OS 가 관리하지만, 가상화된 시스템에서는 가상 머신 모니터가 exception vector table 을 관리한다. 따라서, 발생한 data abort exception 은 가상 머시 모니터에 의해서 처리가 가능하다.

둘째, 가상 인터럽트 콘트롤러는 하드웨어 인터럽트 콘트롤러 내부의 레지스터와 동일한 구성을 가지는 레지스터들을 가지고 있다. 이러한 레지스터들은 가상 머신 모니터가 관리하는 메모리상에 위치하며, 하드웨어 인터럽트 콘트롤러와의 일관성을 유지시켜주는 작업을 수행한다. 이러한 작업은 <그림 2>에 있는 access controller 에서 수행된다.

4. 인터럽트 처리 과정

가상 인터럽트 콘트롤러를 사용하여 인터럽트를 처리하는 과정은 <그림 2>와 같다.

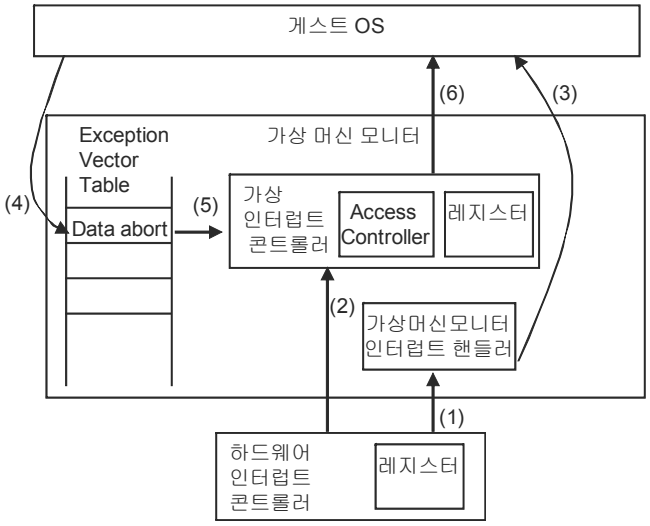
하드웨어에서 인터럽트가 발생하면 하드웨어 인터럽트 콘트롤러를 통하여 CPU 에 전달되고 가상 머신 모니터의 인터럽트 핸들러가 수행된다 <그림 2-(1)>. 가상 머신 모니터의 인터럽트 핸들러는 인터럽트 발생정보를 가지고 있는 하드웨어 인터럽트 콘트롤러의 IRQStatus 레지스터를 가상 인터럽트 콘트롤러의 IRQStatus 로 복사함으로써, 인터럽트 발생정보를 가상 인터럽트 콘트롤러로 전달한다<그림 2-(2)>. 이때, 가상 인터럽트 콘트롤러의 IRQEnable 레지스터를 조사하여 게스트 OS 가 mask 시키지 않은 인터럽트만 게스트 OS 로 전달한다. 이후, 가상 머신 모니터는 게스트 OS 의 인터럽트 핸들러를 수행하여 인터럽트를 처리한다<그림 2-(3)>.

게스트 OS 의 인터럽트 핸들러가 실행되면 하드웨어 인터럽트 콘트롤러에 접근을 수행한다. 이때 가상 머신 모니터는 3 절에서 설명한 바와 같이 data abort exception 을 사용하여 게스트 OS 의 하드웨어 인터럽트 콘트롤러에 대한 접근을 가로챈다<그림 2-(4)>.

Data abort exception 핸들러는 data abort 가 발생한 주소를 조사하여 하드웨어 인터럽트 콘트롤러에 대한 접근이면 가상 인터럽트 콘트롤러의 access controller 를 수행한다<그림 2-(5)>. Data abort 가 인터럽트 콘트롤러를 위한 것이 아니라면 예외를 현재 수행 중인 OS 로 전달한다.

Access Controller 는 data abort 의 유형을 조사한다. 그리고 가상 인터럽트 콘트롤러에 있는 해당 레지스터에 대한 읽기/쓰기 작업을 수행하고, 필요한 경우 하드웨어 인터럽트 콘트롤러와의 일관성 유지 작업을

수행한다. 이후, data abort 를 발생시킨 guest OS 로 복귀한다 <그림 2-(6)>.

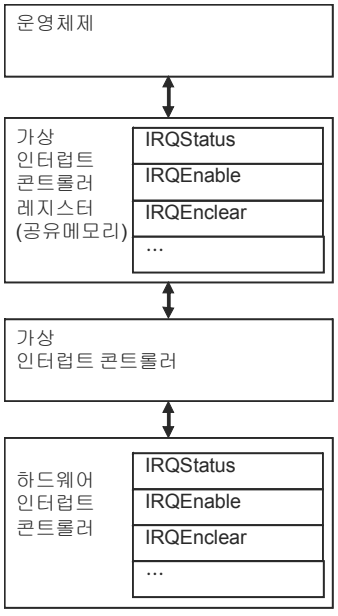


<그림 2> 인터럽트 처리

5. 개선된 인터럽트 콘트롤러 접근 제어

3 절에서는 data abort exception 에 기반해서 하드웨어 인터럽트 콘트롤러에 대한 접근을 가로채는 방식에 대해서 설명했다. 이 방식은 읽기/쓰기 모두 data abort exception 을 발생시킴으로 비용이 많이 발생하는 문제가 있다.

가상 인터럽트 콘트롤러와 하드웨어 인터럽트 콘트롤러와의 일관성을 유지시켜주기 위해서는 게스트 OS 에서 발생한 인터럽트 콘트롤러에 대한 쓰기 동작을 가로채는 것이 필요하며, 읽기 동작은 가로채지 않아도 일관성을 유지시켜줄 수 있다. 따라서 본 절에서는 이를 개선하여 쓰기에만 data abort exception 을 사용하는 방식을 제안한다.



<그림 3> 개선된 가상 인터럽트 콘트롤러 구조

본 방식에서는 하드웨어 인터럽트 콘트롤러에 대한 주소 매핑을 제거하는 대신에 read-only 매핑을 사용한다. 즉 게스트 OS 의 하드웨어 인터럽트 콘트롤러에 대한 매핑을 가상 머신 모니터 및 운영체제가 공유하는 메모리 영역으로 대체한다<그림 3>. 그리고 공유 메모리 영역에 가상 인터럽트 콘트롤러의 레지스터들을 저장한다. 또한, 쓰기에만 data abort exception 을 발생시키기 위해서 매핑의 속성은 read-only 로 수정한다.

OS 가 하드웨어 인터럽트 콘트롤러에 대한 읽기를 시도할 경우, 공유 메모리로 설정된 가상 인터럽트 콘트롤러 레지스터에서 data abort exception 없이 바로 읽어갈 수 있다. 반면 쓰기의 경우, 4 절과 동일하게 data abort exception 을 사용한다.

6. 구현

본 논문에서 제안한 가상 인터럽트 콘트롤러는 휴인스[7]에서 개발된 ARM11-6410SYS 임베디드 시스템에 구현되었다. 본 시스템은 ARM1176JZF-S 코어를 사용한 삼성의 S3C6410 SoC 를 기반으로 개발되었으며, CPU 의 동작 속도는 533MHz 이다. 자세한 스펙은 다음과 같다.

- 128MB DDR SDRAM
- 128MB NAND flash
- 1MB NOR flash
- 4.3 inch wide color TFT LCD, 해상도 480x272
- 리눅스 커널 : 2.6.21

7. 실험

7.1 가상 인터럽트 콘트롤러 접근 시간

본 실험에서는 게스트 OS 에서 가상 인터럽트 콘트롤러의 레지스터를 읽기/쓰기하는데 소요되는 시간을 측정하였으며, 그 결과는 <표 2>와 같다. 게스트 OS 에서 발생하지 않는 읽기/쓰기는 측정하지 않았다. 예를 들어 IRQStatus 에 대한 쓰기는 게스트 OS 에서 발생하지 않는다.

Data abort 는 3, 4 절에서 설명한 방식이며, read-only 는 5 절에서 설명한 방식이다. 또한, native 는 가상 인터럽트 콘트롤러를 사용하지 않는 방식이다. 결과를 보면 가상 인터럽트 콘트롤러를 사용할 경우, native 에 비해서 접근 시간이 매우 높은 것을 알 수 있다. 이것은 native 의 경우 1 개의 읽기/쓰기 명령어로 처리되는 반면에 data abort 및 read-only 는 복잡한 처리 과정을 거치기 때문이다.

Read-only 와 data abort 를 비교할 경우, 읽기는 read-only 가 data abort 에 비해서 많이 감소한 것을 알 수 있다. 반면에 쓰기는 오히려 증가한 것을 알 수 있다. Read-only 방식의 경우, 공유 메모리를 사용하여 가상 인터럽트 콘트롤러를 구현하였다. 이때 가상 인터럽트 콘트롤러와 게스트 OS 에 의해서 공유되는 메모리의 일관성을 유지 시키기 위해서 공유메모리 영역은 CPU 의 캐쉬에 의해서 캐쉬되지 않도록 설정하였다.

따라서 쓰기의 경우 read-only 가 data abort 보다 시간이 오래 소요된다.

<표 2> 가상 인터럽트 콘트롤러 접근 시간

(단위 : us)

		읽기	쓰기	합계
Data abort	IRQStatus	1.32		12.09
	IRQEnable		1.37	
	IRQEnclear		1.29	
	Address	1.37	1.41	
Read-only	IRQStatus	0.4		9.51
	IRQEnable		2.47	
	IRQEnclear		2.12	
	Address	0.39	2.54	
Native	IRQStatus	0.11		0.9
	IRQEnable		0.08	
	IRQEnclear		0.08	
	Address	0.11	0.08	

<표 2>의 합계는 한 개의 인터럽트를 처리하기 위해서 소요되는 인터럽트 콘트롤러 접근 시간의 합을 측정한 것이다. 리눅스 커널에서는 한 개의 인터럽트를 처리하기 위해서 총 9 번의 인터럽트 콘트롤러 접근이 발생하며 자세한 내용은 <표 3>과 같다.

<표 3> 인터럽트 콘트롤러 접근 회수

	읽기	쓰기
IRQStatus	4	
IRQEnable		1
IRQEnclear		1
Address	2	1

<표 2>의 합계를 보면 인터럽트 콘트롤러 접근시간은 read-only 가 9.51us 를 소비하여 12.09us 를 소비하는 data abort 보다 21% 시간이 향상된 것을 알 수 있다.

7.2 가상 인터럽트 콘트롤러 오버헤드

인터럽트가 발생하여 해당 인터럽트 핸들러가 수행되는데 소요되는 시간을 측정하였으며, data-abort 와 read-only 의 경우 261us 및 258us 가 소요되었다. 본 시간은 가상 인터럽트 콘트롤러 접근 시간을 포함하며, 본 실험에서는 타이머 인터럽트만 발생하였다.

본 실험 결과에 따르면, 가상 인터럽트 콘트롤러 접근 시간(12.09us : data abort, 9.51us : read-only)이 전체 인터럽트 처리에서 차지하는 비중은 4.6%(data abort)와 3.7%(read-only)로 그리 크지 않음을 알 수 있다. 즉, 가상 인터럽트 콘트롤러의 경우, 인터럽트 콘트롤러 접근 시간 자체는 native 에 비해서 많이 크지만 전체 인터럽트 처리 시간을 고려할 경우, 오버헤드가 크지 않음을 알 수 있다.

가상 인터럽트 콘트롤러 접근시간이 전체 시스템의 성능에서 차지하는 비중을 측정하기 위해서 화면에 문자열을 계속 출력하는 프로그램을 수행하여 인터럽트를 연속적으로 발생시켰다. 이때 인터럽트는 초당 600 회 발생하였으며, 이중 200 회는 timer 인터럽트, 400 회는 UART 인터럽트였다. 이때 read-only 방식의 경우 1 초당 가상 인터럽트 콘트롤러를 접근하는데 소요되는 시간은 5.7ms (=9.51us * 600)이다. 이것은 1 초의 0.57%로 전체 시스템의 성능에 영향을 거의 미치지 않는 다는 것을 알 수 있다.

인터럽트 부하를 좀더 증가시켜 초당 1000 회가 발생한다고 가정하더라도 1 초당 가상 인터럽트 콘트롤러를 접근하는데 소요되는 시간은 9.51ms (=9.51us * 1000)이다. 이것은 1 초의 0.95%로 역시 전체 시스템의 성능에 영향을 거의 미치지 않는 다는 것을 알 수 있다.

8. 결론

본 논문에서는 ARM 기반 임베디드 시스템상에서 전가상화를 지원하는 가상 머신 모니터를 위한 가상 인터럽트 콘트롤러를 제안하였다. 가상 인터럽트 콘트롤러는 실제 하드웨어 인터럽트 콘트롤러와 동일한 인터페이스를 가지며, 운영체제의 수정 없이 가상 인터럽트 콘트롤러를 하드웨어 인터럽트 콘트롤러와 동일한 방식으로 사용하는 환경을 제공한다. 실험을 통하여 성능을 측정한 결과, 전체 시스템의 성능에서 가상 인터럽트 콘트롤러를 적용한 오버헤드는 거의 무시할 수준인 것으로 판단되었다.

참고문헌

[1] “Understanding Full Virtualization, Paravirtualization, and Hardware Assist”, White Paper, pp.4-5, VMware, 2007.

[2] “VMware MVP”, http://www.vmware.com/products/mobile/index.html, VMware, 2010.

[3] “Meeting the Challenges of Connected Device Design”, White Paper, VirtualLogix, pp. 8-10, 2006.

[4] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, “Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones”, IEEE 5th Consumer Communications and Networking Conference, pp. 257-261, 2008.

[5] Fabrice Bellard, “QEMU, a Fast and Portable Dynamic Translator”, USENIX 2005 Annual Technical Conference, pp. 41-45, 2005.

[6] “ARM PrimeCell Vectored Interrupt Controller (PL192), Technical Reference Manual”, ARM, 2002.

[7] http://www.huins.com, HUINS, 2010.