임베디드 시스템 가상화: 도전과 과제

유시환 유혁
고려대학교 컴퓨터학과
{shyoo, hxy}@os.korea.ac.kr

Embedded System Virtualization: Challenges and Problems

Seehwan Yoo Chuck Yoo
Department of Computer Science and Engineering, Korea University

요 약

시스템 가상화 기술은 현재까지 대형 서버 시스템의 관리의 편의성과 유지 비용의 최소화를 위해 널리 사용되어 오고 있다. 임베디드 환경에서도 가상화를 통해 유지, 관리의 편의성과 시스템의 신뢰성 확보 및 보안성 강화 등의 장점을 가질 수 있다. 본 논문에서는 임베디드 시스템의 가상화를 위한 도전 과제들과 구체적인 문제점들을 분석한다.

1. 서 론

가상화 기술은 물리 객체에 대한 가상화된 시각을 줌으로써 물리적인 객체의 한계점을 극복할 수 있도록 해 준다. 가상화 기법은 가상화 층위를 통해, 물리적 객체의 속성을 사용자가 필요한 형태로 가공하여 제공하므로, 실재하지 않는 객체의 구현이나, 추가적인 객체의 구현을 제공한다. 이처럼 가상화된 객체는 상위의 사용자들이 가상화 된 사실을 알지 못하도록 함으로써 사용자에게는 자신만이 독립적인 물리 객체를 가진 것과 같은 착각을 준다.

전통적인 가상화 기법은 서버를 기반으로 한 대형 시스템에서 널리 사용되어 왔다. 가상 머신 기반의 시스템 가상화 기법들을 통해 사용자는 하나의 서버 시스템 안에서 여러 종류의 서버들을 수용하여 동작시킴으로써 총 관리 비용(TCO, Total Cost of Ownership)의 절감을 이룰 수 있다. 이 밖에도 가상화를 통하여, 보안성의 확보, 시스템 환경의 이동, 아직 개발되지 않은 시스템의 테스트와 검증, 교육용 시스템의 제작 등에 활용될 수 있음이 알려져있다[1].

임베디드 시스템의 경우, 시스템 가상화는 현재까지 성능상의 오버헤드가 큰 것으로 지적되고 있기 때문에, 아직까지 이렇다 할 연구가 진행되지 않고 있다. 하지만, 하드웨어의 성능이 높아짐에 따라 성능상의 제약이 점차로 줄어들고 있으며, 가상화를 함으로써 얻을 수 있는 장점들로 인하여, 근시일 내에 임베디드 시스템의 가상화 연구도 진행될 것이라 본다.

본 논문에서는 임베디드 시스템의 가상화에서 고려해야 할 도전 과제들과, 대부분의 임베디드 시스템이 사용하고 있는 ARM 프로세서를 사용하여 시스템의 가상화를 구현하고자 할 때, 기존의 가상화 구현과 어떤 차이점이 있는지를 밝힌다.

가상 머신 모니터는 여러 개의 가상 머신을 동작시키기 위한 플랫폼이다. 가상 머신 모니터의 중요한 역할은 하드웨어 자원을 고립화하여, 각각의 가상 머신에서 독립적으로 동작할 수 있도록 분리해 주는 것이다.

임베디드 시스템에서 가상 머신 수준의 분리는 프로세스 수준의 분리와 비교했을 때 다음과 같은 차이점을 가진다. 첫째, 운영체제의 프로세스 수준의 분리는 시스템 디자인의 분리를 제공할 수 없다. 하지만, 가상 머신을 사용한 분리는 시스템을 디자인 단계에서부터 분리할 수 있도록 해 준다. 임베디드 시스템의 디자인은 시스템의 하드웨어와 응용들과 밀접하게 관련되어있어, 새로운 하드웨어 장치나 응용을 추가하는 경우, 새로운 시스템을 디자인 해야 하는 단점이 있다. 예를 들어, 기존의 핸드폰 제품에서 비디오 컨퍼런스를 지원하기 위해서는 카메라와 인코더 같은 하드웨어 장치를 추가하고, 해당 하드웨어 장치를 제어할 수 있는 시스템 소프트웨어를 추가해야 한다. 이를 위해서는 시스템을 재구성하거나 시스템을 새로 디자인 해야 한다. 왜냐하면, 응용 수준의 서비스 밑에 있는 복잡한 시스템 서비스들에 대한 의존성에 대한 문제는 해결 되지 않았기 때문이다. 따라서 개발자는 시스템의 개발 대상이 되는 부분 외에도 다른 부분을 함께 고려하여 개발해야 한다. 가상 머신 모니터를 사용하여 시스템을 분리한 경우, 기존의 시스템과 완전히 분리된 가상 머신 위에서 추가된 기능만을 구현할 수 있다. 이 경우, 플랫폼과 의존성이 있는 모든 문제에 대해 고려할 필요 없이 각각의 시스템은 각자 사용하는 응용에 대해서만 고려하면 된다. 즉, 각 시스템은 관리하기 쉬운 형태로 각각의 시스템 소프트웨어를 가지고 있게 된다.

두번째 차이점은 임베디드 시스템의 특성 상, 프로세스 수준의 분리가 완전한 분리를 구현하고 있지 못하다는 것이다. 임베디드 시스템에서 사용되는 운영체제들은 응용과 시스템 소프트웨어의 구분이 명확하지 않은 경우가 많으며, 응용 프로그램의 오동작은 전체 시스템 실패로 이어지는 경우가 많다. 임베디드 시스템에서 사용하는 운영체제는 대개 주소 공간 관리를 단순화 하여, 운영체제에 의해 전역으로 사용되는 공간이 응용 프로그램의 비정상적인 동작에 의해 영향을 받을 수 있는 여지가 있다. 반면, 가상 머신 모니터를 사용한 경우, 하드웨어 수준의 분리를 구현하므로, 시스템

*이 논문은 2004년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (No. R01-2004-000-10588-0).

전체의 안정성 또한 높일 수 있다.

세번째로, 프로세스 수준의 분리는 시스템 실패나 장치 드라이버 실패와 같은 문제를 해결할 수 없다는 단점을 가지고 있다. 새로운 하드웨어의 등장은 새로운 응용들을 등장 시키고, 이러한 하드웨어에 대한 디바이스 드라이버들은 대개 시스템 모드 혹은 특권 모드에서 동작한다. 응용 수준의 분리가 운영체제에 의해 지켜지더라도 장치 드라이버를 고치지 않고, 응용을 분리하여 실행을 지속하는 것은 매우 어렵다. 가상 머신 모니터의 경우, 각 가상 머신의 설정에 따라 불필요한 하드웨어 장치를 제거하고, 필요한 가상 머신에 한해서만 장치 드라이버를 사용할 수 있다. 따라서, 시스템 실패의 가장 큰 요인인 불안정한 드라이버로부터의 시스템 보호를 구현할 수 있다.

2. 배경 및 관련 연구

2.1 전통적인 가상화와 부정 가상화

전통적인 가상화 연구들[2,3,4,5,6,7]에서 가상 머신 모니터는 하드웨어를 에뮬레이션 하는 플랫폼을 제공했다. 초기 컴퓨터 시스템에서 소프트웨어 개발은 하드웨어 개발에 이어 진행이 되었기 때문에, 운영체제와 컴파일러를 비롯한 소프트웨어는 동작하는 하드웨어에 매우 의존적이었다. 소프트웨어의 하드웨어 의존성은 하드웨어가 빠른 속도로 발전함에 따라 소프트웨어의 호환성이라는 문제가 등장하게 되었다. 따라서, 변화하는 하드웨어와 기존의 소프트웨어를 분리할 수 있는 레이어가 필요하게 되었다.

하드웨어와 소프트웨어를 구분하는 인터페이스는 인스트럭션 셋 구조(ISA)이다. ISA가 소프트웨어 구현과 구체적인 하드웨어의 구현을 분리하였으나, 여전히 하나의 물리 시스템에 한정되어있었다. 가상 머신 연구의 중요한 목적은 물리적인 하드웨어에 종속적이지 않은 매우 유연한 소프트웨어를 개발할 수 있도록 하는 것이다. 가상 머신들은 물리 시스템을 에뮬레이션하여 여러 개의 다양한 플랫폼에 대한 뷰를 제공해 준다. 하나의 가상 머신은 독자적인 운영체제나 명령어 셋을 가지며, 이들은 실제 하드웨어와 다를 수 있도록 한다.

가상화 연구에 대한 서베이 논문들[1,8,9,10,11]은 여러 종류의 다양한 가상화 기법들을 소개하고 있다.

기존의 가상 머신 연구들이 가상 머신 위에서 동작하는 게스트 운영체제의 완벽한 호환을 추구했던데 비해, 최근의 가상 머신 연구들은 게스트 운영체제의 수정을 통하여 가상 머신의 확장성(Scalability)과 성능 (Performance) 상의 약점을 극복하고 있다.

Washington 대학의 Denali[12] 는 수만 개의 네트워크 서버를 하나의 물리 머신에서 동작시키는 것을 지향한다. 실행 시간의 오버헤드를 줄이기 위해서 Denali는 부정가상화(para-virtualization)를 도입하였다. Denali는 모든 하드웨어를 가상화 하지 않고, 네트워크 인터페이스와 디스크만을 가상화하여 대량의 가상 네트워크 서버가 동작할 수 있도록 하였다. 서로 다른 가상 머신은 서로 다른 이름 공간(name space)을 사용하여 상위의 소프트웨어 컴포넌트가 가상화된 하드웨어 자원의 접근을 할 수 없도록 하였다. Denali의 단점은 운영체제와 함께 응용 프로그램까지 수정을 필요로 한다는 것이다. Denali위에서 동작하는 응용 프로그램들은 Denali 위에서 동작하는 운영체제를 지원하기 위해서 수정해야만 한다.

Xen[13]은 최근의 가상 머신 모니터와 관련된 가장 활발한 연구 중 하나이다. Denali가 목표했던 것보다는 작은 수(수천 개)의 가상 네트워크 서버 지원을 목표로 하였다. Xen의 주요한 디자인 원칙은 1) 응용 프로그램 바이너리의 수정없는 지원, 2) 실제 다중 응용 프로세스 운영체제 지원, 3) 성능을 위한 부정 가상화와 비협조적 머신 구조에 대한 자원 분리, 4) 자원 가상화에 대한 영향의 완벽한 은닉으로 요약된다.

Xen은 Denali가 응용 프로그램의 수정을 필요로 하는 단점을 보완하여, 완벽한 응용 바이너리 인터페이스 (Application Binary Interface) 호환을 추구하였다. 또한, Denali가 인터넷 서버만을 위해 운영체제의 기능을 변형한 데 비해, 다중 응용 프로세스를 지원하는 범용 게스트 운영체제를 고려하여 디자인되었다. Xen은 게스트 운영체제가 스스로 페이징과 디스크 할당과 같은 고유의 기능을 수행할 수 있도록 하였으며, Denali가 가진 이름 공간을 없애고, 안전한 접근 제어를 통해서도 이를 구현할 수 있음을 보였다. Xen에서 사용된 부정가상화 기법은 운영체제의 수정만을 통하여 가상 머신 모니터의 통제하에 게스트 운영체제가 하드웨어 상에서 직접 동작할 수 있도록 하였다.

하지만, Xen은 게스트 운영체제를 가상화를 위해 변경함으로써, 다른 플랫폼으로 포팅하기 위해서는 기존의 가상화 기법들보다 훨씬 높은 난이도와 비용을 필요로 한다. 즉, Xen 형태의 가상화를 다른 플랫폼에서 구현하기 위해서는 가상화를 구현하기 위해 컴퓨터 구조의 세부적인 구현뿐만 아니라, 대상 게스트 운영체제의 하드웨어 관련 구조까지도 자세히 알고 있어야 하는 단점을 안고 있다. 이러한 단점을 극복하기 위해 기계적으로 기존 운영 체제의 민감한 명령어를 찾아내어, 가상화에 적합한 형태로 변경을 도와주는 선행가상화(pre-virtualization)가 소개되었다[14]. 선행가상화는 부정가상화의 난점인 다른 플랫폼으로의 이식성(portability)을 높여줄 수 있지만, 부정가상화의 틀 안에서 이루어진다는 점에서 한계점을 안고 있다.

임베디드 환경에서는 프로그래밍 언어 기반의 가상 머신이 널리 사용되고 있다[15,16]. 이들의 장점은 표준화된 개발 환경을 통해서 응용 프로그램의 기능을 확장할 수 있다는 것이다. 특정 가상 머신에서 동작하는 응용 프로그램이 일단 작성된 후에는 작은 가상 머신 위에서도 동작할 수 있기 때문에, 손쉽게 배포와 실행이 가능하다. 하지만, 이러한 가상 머신 형태의 접근 방식들은 성능상의 문제를 안고 있으며, 시스템의 저수준 기능에 직접 접근할 수 없다는 단점이 있다. 최근에 소개된 센서 네트워크 플랫폼 상에서 동작하는 가상 머신의 경우[16], mote와 같은 매우 제한된 자원 환경에서도 실행이 가능하며 쉽게 배포할 수 있는 장점이 있다. 하지만, 센서 플랫폼에 매우 밀접하게 의존되어있어, 하드웨어의 변화에 대응할 수 없고, 가상

머신을 위해 제안된 특별한 언어를 사용해야만 하는 단점이 있다.

2.2 ARM 프로세서의 구조와 특징

ARM 프로세서는 RISC 프로세서 계열의 하나로 임베디드 시스템을 타깃으로 설계되었다[17]. ARM 프로세서는 다음과 같은 7개의 실행 모드가 있으며, 매 순간 프로세서는 하나의 실행 모드에서 동작한다.

1. User mode: 사용자 응용이 동작하는 모드이며, 비특권 모드이다. 다른 모드가 선점 가능하다
2. Supervisor mode: 관리자 모드로서, 특권 모드이다. 다른 모드에 의해 선점되지 않는다.
3. Interrupt mode: 외부 인터럽트가 발생하면, 인터럽트 시그널에 의해 전이되는 모드이다. FIQ 모드를 제외한 다른 모드에 의해 선점되지 않는다. 특권 모드이다.
4. Fast interrupt mode: 빠른 인터럽트 모드로서, 인터럽트 모드를 선점하여 실행이 가능하다. 특권 모드이다.
5. Abort mode: 내부의 오류에 의해 전이되는 모드이다. MMU에 의한 abort와 divide by 0 등과 같은 에러에 의해 전이 된다. 특권 모드이다.
6. Undefined mode: 인스트럭션이 비정상적인 경우 전이되는 모드이다. 특권 모드이다.
7. System mode: 시스템 소프트웨어 중 특권 모드에서 선점 가능한 소프트웨어 수행을 위해 별도로 만든 모드이다. 다른 모드에 의해 선점이 가능하다. 특권 모드이다.

사용자 모드를 제외한 모드들은 특권 모드로서, 명령어의 실행에 있어 특권 레벨의 제한을 받지 않는다. 그리고, 메모리 접근 시 페이지 테이블과 시스템 상태 레지스터의 값에 따라 비 특권 모드에서 특권 모드 영역에 대한 접근을 시도하거나, 허락되지 않은 쓰기 명령을 수행하려고 할 경우, 익셉션이 발생한다.

ARM 프로세서의 레지스터는 범용 레지스터와 특수 레지스터로 구분된다. ARM 프로세서의 범용 레지스터는 모드의 구분 없이 사용되는 12개의 일반 레지스터와 FIQ 모드를 위한 4개의 뱅크 레지스터 및 스택 포인터, 링크 레지스터, 프로그램 카운터로 구성된다. 특수 레지스터는 현재 시스템의 상태를 나타내는 CPSR (Current Program Status Register)와 이전 모드로 복귀할 때, 이전 CPSR을 복구하기 위한 SPSR(Stored Program Status Register)로 구성된다. 이 중, 스택 포인터, 링크 레지스터, 프로그램 카운터, SPSR은 모드별로 존재한다. 모드가 전이될 때는 모드별로 존재하는 레지스터의 값이 복구된다.

최근의 ARM 프로세서들은 페이징을 통한 가상 메모리를 지원한다. 페이징을 위해서는 별도의 프로세서인 시스템 제어 코프로세서를 사용하여 페이지 테이블을 지정해 주어야 한다. 시스템 제어 코프로세서의 접근은 특권 모드에서만 가능하도록 하여 시스템을 보호한다. ARM 프로세서는 페이징을 통한 메모리 보호 이외에도, 도메인을 이용한 메모리 보호를 지원한다. 1단계 페이지 테이블의 엔트리에는 해당 메모리 영역이 어떤 도메인에 속하는지를 기술하는 도메인 번호 비트가 할당되어있다. 4비트의 도메인 번호는 총 16개까지의 도메인을 설정할 수 있음을 의미한다. 시스템 제어 코프로세서의 레지스터인 도메인 접근 제어 레지스터의 설정 값에 따라 0번부터 15번 도메인에 대해 메모리 접근 시, 메모리 보호 검사 여부를 미리 결정할 수 있다.

ARM 프로세서는 메모리-매핑 I/O를 지원한다. I/O 장치의 접근을 지시하는 직접적인 명령어는 없으며, 하드웨어 장치의 레지스터들이 메모리 영역의 일부에 매핑되어 있다. 이 경우, 장치에 접근하는 명령어들이 특권 명령어가 아니라, 일반 명령어이기 때문에 보호에 좀 더 각별한 신경을 써야 한다.

3. 임베디드 시스템 가상화의 장점

임베디드 시스템 가상화의 장점은 다음과 같은 네가지로 정리할 수 있다.

a) 시스템 통합의 신뢰성 확보
b) 복잡성 완화
c) 높은 보안성 확보
d) 고가용 시스템 구현
e) 테스트와 탑재의 편이성

임베디드 시스템은 특성 상 구조와 구성 하드웨어의 종류가 매우 다양하다. 또한, 범용 시스템과 달리 시스템이 특정 목적을 위해서만 제작되어 제한된 응용 프로그램과 UI를 가지고 있다. 따라서, 일단 시스템이 빌드 된 후에는 디버깅과 복구가 어려운 단점이 있다. 하지만 최근의 임베디드 시스템들은 다양한 하드웨어 장치들을 하나의 시스템에 흡수하면서, 복합화되고 있다. 따라서 시스템의 통합을 하는 과정에서 신뢰성 검사 등을 통해 안정화된 소프트웨어들만 수용하도록 하고 있다. 하지만, 개개의 소프트웨어가 안정화되어 있더라도 통합을 하면서 문제가 확대되거나, 전체적인 신뢰성이 깨질 수 있기 때문에 시스템 통합시의 신뢰성 확보는 매우 중요한 문제가 된다. 가상 머신 모니터를 활용하는 경우 개별적인 응용 시스템이 독립된 가상 머신 환경에서 동작하기 때문에, 시스템의 통합시의 복잡한 의존성에 대한 문제가 해결된다. 또한 개별 응용 시스템의 실패가 고립되어, 전체 시스템의 실패로 이어지는 것을 막을 수 있기 때문에 전체 시스템의 신뢰성 확보 측면에서도 중요하다.

가상 머신은 하드웨어 수준의 독립성을 지원하므로, 시스템 소프트웨어의 취약점을 이용한 공격에 대해서도 기존의 보안 시스템보다 높은 수준의 보안성을 제공할 수 있다. 특히, 휴대폰 등의 시스템에서는 개인화가 중요한 주제가 됨에 따라서, 개인적으로 민감한 데이터를 이동하는 경우, 높은 수준의 보안을 제공하기 위해서 가상 머신을 활용할 수 있다. 뿐만 아니라 소프트웨어적인 동작을 가상 머신을 통해 제공함으로써, 접근을 차단하여 소프트웨어 자체의 권리에 대한 보호를 제공할 수 있다.

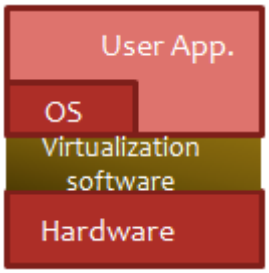

그림 1. 일반적인 시스템 가상화의 구조

가상 머신 모니터는 하나의 물리 시스템에 여러 가상 머신을 동작 시킬 수 있기 때문에, 고가용 시스템에 활용하여 서비스의 지속성을 제공할 수 있다. 한 가상 머신의 시스템 실패는 대상 가상 머신으로 고립되기 때문에 해당 가상 머신을 재시작하는 동안 다른 가상 머신 상의 백업 시스템을 통하여 서비스 지속성을 확보할 수 있다.

가상 머신 구조는 시스템의 빠른 개발과 쉬운 탑재에도 도움을 준다. 최근의 임베디드 시스템의 복합화 경향은 높은 소프트웨어 재사용성을 필요로 한다. 가상 머신 위에서 기존에 사용하던 시스템을 그대로 사용함으로써 빠른 개발 시간을 확보할 수 있으며, 기 개발된 시스템을 재사용함으로써 개발의 비용을 줄일 수 있다.

4. 임베디드 시스템 가상화와 서버 시스템 가상화의 차이점

4.1 임베디드 시스템의 취약성

임베디드 시스템에 사용되는 시스템 소프트웨어들은 제한된 하드웨어 자원의 문제로 인해, 대형 서버 시스템보다 낮은 신뢰성을 가지며, 최적화로 인한 구조상의 결점을 가진다. 임베디드 시스템의 가상화는 이처럼 제한된 자원과 지원 소프트웨어의 문제를 극복해야 한다. 서버시스템에서 사용하는 운영체제와 하드웨어와 비교하여 볼 때, 임베디드 시스템에서 주로 사용되는 운영체제와 하드웨어가 가진 단점은 시스템이 매우 간단하고 견고하게 구성되어야 함을 알 수 있다. 특히, 임베디드 시스템의 경우, 디버깅과 테스트가 어렵기 때문에, 높은 수준의 견고성이 필요하다.

4.2 다양한 하드웨어의 지원

임베디드 시스템은 동일한 종류의 장치라고 하더라도 실제 구성이 각기 다를 수 있으며, 이러한 다양한 하드웨어 구성을 관리하기 위한 기법이 특별히 필요하다. 서버시스템은 이에 비해, 사용되는 하드웨어의 종류와 구성이 비교적 표준화 되어있으며, 안정성이 높은 장치를 사용하기 때문에, 하드웨어 다양성에 대한 고려가 특별히 필요하지 않다. 더욱이 임베디드 시스템에서 사용되는 시스템 소프트웨어는 구성된 하드웨어에 의존적인 경우가 많기 때문에, 범용 시스템의 가상화보다 어려운 점이 많다.

5. 임베디드 시스템 가상화의 도전 과제

5.1 게스트 운영체제의 특권 레벨 조정

일반적인 가상 머신의 구조는 그림1과 같다. Goldberg는 가상화 가능한 시스템의 구조를 정의하였다 [5,6]. 가상화를 지원하기 위해서는 시스템의 상태를 변경하는 민감한 명령어들이 특권 명령어로 정의되어야 한다. 즉, 민감한 명령어들을 수행하는 경우 가상 머신 모니터가 트랩을 통해 제어권을 넘겨받아, 해당 가상 머신에 대해 필요한 연산을 대신해 준다. 이를 위해서, 게스트 운영체제는 기존의 운영체제와 달리 특권 모드가 아닌 비 특권 모드에서 동작해야 한다.

표 1. 메모리와 프로세서의 특권 모드에 따른 게스트 운영체제의 세 가지 동작 모드

(Memory, Processor)	P: Privileged mode N: Non-privileged mode		
User App	(N,N)	(N,N)	(N,N)
Guest OS	(N,N)	(N,P)	(P,N)
VMM	(P,P)	(P,P)	(P,P)

기존의 시스템이 운영체제를 특권 모드에서, 사용자 응용 프로그램을 비 특권 모드에서 실행하였던 것과 달리, 가상 머신 모니터가 특권모드에서 동작하도록 하기 위해서는 최소한 세 계층을 보호할 수 있는 보호 메커니즘이 필요하다. 즉, 가상 머신 모니터와 게스트 운영체제간, 게스트 운영체제와 사용자 응용 프로그램간의 보호를 지원해야 한다.

하지만, 대개의 임베디드 시스템에 사용되는 프로세서는 두 개의 특권 레벨만 지원한다. 따라서, 두 개의 하드웨어 특권 레벨을 사용하여 세 계층을 구분해 주기 위해서는 특별한 기법이 필요하다.

게스트 운영체제의 비 특권 모드 동작을 위해서는 ①게스트 운영체제의 실행 영역을 비 특권 영역으로 지정하거나, ②비 특권 상태에서 실행을 하도록 지정해 주어야 한다. 운영체제의 실행 영역을 비 특권 영역으로 지정한 경우, 매 인스트럭션 실행에 대해 트랩을 발생하게 되므로 실행 시간에 수행되는 운영체제의 코드에 대해 가상 머신이 디코딩과 해석을 담당해야 한다. 이것은 오버헤드가 너무 크기 때문에, 특권을 필요로 하는 명령에 대해서만 트랩을 발생하도록 하거나, 마이크로 커널과 같이 대부분의 운영체제 역할을 사용자 레벨로 올리는 등의 기법을 사용해야 한다. 메모리와 프로세서의 모드에 따른 구성은 표 1과 같은 세 가지 조합이 가능하다.

게스트 운영체제의 실행과 실행 영역이 모두 비 특권 상태로 지정된 경우, 게스트 운영체제가 비 특권 모드에서 동작하므로, 민감한 명령에 대해서 가상 머신 모니터가 자연스럽게 트랩을 발생할 수 있다. 하지만, 게스트 운영체제를 사용자 응용으로부터 보호할 수 있는 기법이 추가적으로 필요하다. 게스트 운영체제가 특권 모드 영역에서 동작하기 때문에, 불법적인 사용자 응용의 접근이 트랩을 발생하지 않는다. 따라서, 이러한 접근을 막을 수 있는 메모리 보호 기법이 필요하다.

게스트 운영체제가 특권 모드에서 동작하는 경우에는 어떠한 명령어도 직접적인 수행이 가능하므로, 트랩을

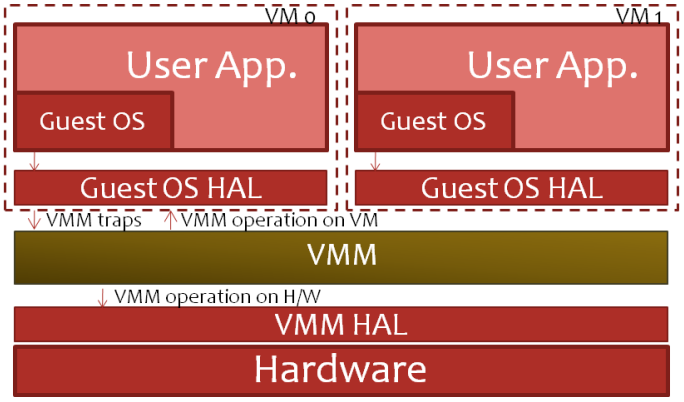

그림 2. 가상화 하드웨어 추상 계층

발생시키기 위해서 특별한 기법이 필요하다. 즉, 이 경우에는 게스트 운영체제를 일부 수정하여 가상 머신 모니터가 수행에 직접적으로 관여할 수 있도록 해 주어야 한다. 그러나 운영체제의 메모리 영역은 비 특권 모드로 설정되어 가상 머신 모니터 영역에 대한 접근은 트랩을 발생시킨다. 게스트 운영체제의 민감한 명령의 실행을 막을 수 있는 방법이 없기 때문에, 위험한 방법이다. 또한 게스트 운영체제의 영역이 사용자 수준에서 접근 가능하므로, 불법적인 접근을 막을 수 있는 추가 기법이 필요하다. 또한 게스트 운영체제의 영역이 사용자 수준에서 접근 가능하므로, 불법적인 접근을 막을 수 있는 추가 기법이 필요하다.

프로세서의 상태가 비 특권 모드이고, 실행 영역이 특권 모드를 요구하는 경우, 실행을 위한 모든 메모리 억세스 명령이 트랩을 발생한다. 즉, 특권 모드를 필요로 하지 않는 명령어에 대해서도 트랩이 발생한다. 따라서, 이 경우에는 트랩이 너무 빈번하게 발생하는 문제점이 있다.

마이크로 커널 구조에서는 특권 영역이 최소화 되어있고, 많은 운영체제의 기능들이 사용자 수준의 서버로 구현되기 때문에, 서로 다른 주소 공간을 갖게 되어, 실행영역과 프로세서의 모드가 모두 비 특권 모드에서 동작하더라도 비교적 적은 오버헤드로 게스트 운영체제를 보호할 수 있다.

5.2 하드웨어 인터페이스의 범용화

임베디드 시스템은 그 특성 상 구성할 수 있는 하드웨어의 종류가 많다. 이는 구성의 편이성은 제공하지만, 일반화 하기 힘든 단점이 있다. 또한, 메모리-매핑 I/O를 사용하는 많은 시스템들은 I/O 접근 명령이 특권 모드를 필요로 하지 않기 때문에, I/O 접근 시 추가적인 보호 기법을 지원해야 한다. 따라서, 메모리-매핑 I/O를 구조에서 I/O 장치의 접근에 대한 트랩을 발생하기 위해서는 장치 I/O 영역의 메모리를 특권 영역에서 접근 가능하도록 하는 등의 보호 기법이 필요하다.

다양한 하드웨어의 지원을 위해, 기존의 운영체제가 가지고 있는 하드웨어 추상 계층을 가상화 하는 것도 대안이 될 수 있다. 하드웨어 추상 계층은, 운영체제가 하드웨어 장치에 대한 다양한 접근 방식을 통일시켜 줌으로써, 손쉽게 하드웨어의 다양성을 수용할 수 있도록 해 준다. 가상 머신 구조에서도 그림 2와 같이 하드웨어 추상 계층을 정의하면, 물리 시스템의 하드웨어 접근 인터페이스를 가상 머신 상의 하드웨어 인터페이스와 매핑할 수 있으므로 게스트 운영체제는 가상 머신 상의 하드웨어 추상 계층을 이용하며, 이 인터페이스는 가상 머신 모니터에 트랩을 발생시켜, 가상 머신 모니터가 효율적으로 시스템의 제어를 할 수 있도록 돕는다.

이 외에도 가상화 하드웨어 추상 계층은 가상 머신 간 이질적인 하드웨어 자원을 가진 가상 머신을 제작하는데 도움을 준다.

5.3 실시간성 지원

실시간 시스템은 시간에 대한 제약 조건이 매우 큰 시스템으로 임베디드 시스템들 중 높은 비중을 차지하고 있다. 실시간 시스템의 가상화를 위해서는 다음의 두 가지 문제를 해결해야 한다. 1) 물리 시간의 가상화, 2) 동기화 객체 획득 지연 시간의 최소화

가상 머신 상에서는 물리 자원의 접근이 불가능하며, 물리적인 실제 속성이 모두 가려져 있어, 가상 머신 환경에서 실제 시간과 관련된 정보를 얻기는 힘들다.

가상 머신 환경에서 정확한 물리 시간을 기반으로 한 타이머 서비스를 제공하기 위해서는 그림 3 a)와 같이 가상 머신 모니터가 물리 시간에 관련된 정보를 가상 머신에 직접 알려주어야 한다. 가상 머신 모니터는 실시간성을 지원하기 위해 우선 순위 기반 선점형 실시간 스케줄러를 구현한다. 하드웨어에서 발생한 타이머 인터럽트는 가상 머신 모니터가 직접 받아, 필요한 가상 머신에게 전달하고, 이를 전달받은 게스트 운영체제는 타이머 인터럽트를 처리하고, 스케줄러를 호출한다.

실시간 시스템의 구현에서 고려해야 하는 추가적인 오버헤드는 동기화로 인한 지연이다. 낮은 우선 순위의 쓰레드가 동기화 객체를 사용 중인 경우, 선점형 스케줄러의 우선 순위에 의한 역전 현상이 발생할 수 있다. 이 경우, 동기화를 맞추기 위하여 우선 순위를 바꾸어 주거나, 올려줌으로써 동기화 객체를 가진 쓰레드가 가능한 한 빨리 작업을 수행할 수 있도록 한다. 이 경우, 동기화 객체의 사용이 완료 될 때까지 높은 우선 순위의 쓰레드는 실행을 멈추고 기다린다.

가상 머신 환경에서 동기화로 인한 오버헤드는 훨씬 심각하다. 다른 가상 머신이 자원 접근을 동기화하기 위해 동기화 객체를 사용하고 있다면, 실시간 시스템은 실행을 멈추고 기다려야 한다. 이 때, 대기 시간을 제한할 수 없기 때문에(unbound), 실시간 시스템의 성능은 큰 영향을 받을 수 밖에 없다. 따라서, 동기화로 인한 실시간 시스템의 대기 시간을 최소화 하기 위한 기법이 필요하다.

대기 시간을 최소화 하기 위해서, 실시간 시스템의 요청에 대해, 그림 3 b)와 같이 가상 머신 모니터는 우선 순위가 있는 트랩을 발생시키고, 선점 당한 게스트 운영체제에게는 적절한 통지를 보내어, 해당 게스트 운영체제에게 동기화 객체가 선점되어 사용할 수 없음을 알리는 방법을 생각해 볼 수 있다.

5.4 자원 제한의 극복

임베디드 시스템의 하드웨어 자원은 제작 비용과 직접적으로 관련되어있다. 제한된 자원의 문제는 특히 메모리

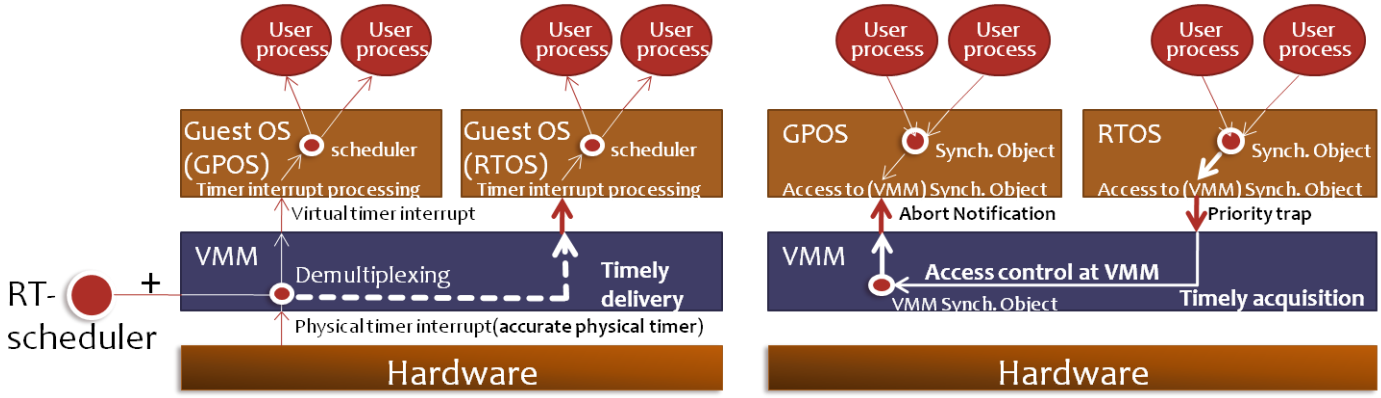

그림 3. 실시간 가상화 시스템을 위한 a)타이머 서비스와 b)우선 동기화

부분이 심각한데, 이는 운영체제의 최적화를 통해 해결할 수 있다. 범용 운영체제와 달리 임베디드 시스템에 사용되는 많은 운영체제는 필요한 시스템의 요구사항에 맞게 최소로 구성할 수 있다. 운영체제의 재구성을 통해 커스터마이즈 된 시스템은 동시에 여러 운영체제를 가짐으로서 발생하는 중복성 (redundancy)을 최소화 할 수 있다.

가상화로 인한 성능 상의 문제를 해결하기 위한 노력들도 필요하다. 예를 들면, 서로 다른 가상 머신 간의 스위칭 시 필요한 캐시의 플러시를 최소화 할 수 있는 기법이나 TLB와 관련된 기법들에 관한 연구가 추가적으로 필요하다.

이 외에도, 최근의 ARM 프로세서 구조에서 제안된 Trust Zone이나 다중 페이지 테이블 기반 레지스터 같은 기술은 임베디드 시스템에서도 하드웨어를 통한 가상화에 적극적으로 활용할 수 있다.

6. 결론

임베디드 시스템 가상화 기술은 가상 머신을 기반으로 하여, 임베디드 시스템의 신뢰성 있는 확장을 가능하게 해준다. 뿐만 아니라, 시스템의 복잡도를 낮추고 강화된 보안을 제공할 수 있다. 하지만 아직까지 성능상의 문제로 인해 널리 활용되고 있지 않은 실정이다. 본 논문에서는 임베디드 시스템의 가상화가 가져올 수 있는 긍정적인 측면들과 이를 구현하기 위해 직면한 과제들에 대해 분석해 보았다. 임베디드 시스템의 가상화를 위해서 1) 제한된 특권 레벨을 활용한 게스트 운영체제의 보호, 2) 다양한 하드웨어 지원을 위한 인터페이스의 범용화, 3) 실시간 시스템의 가상화 지원, 4) 디자인의 중복성을 제거한 디자인 고려와 같은 문제를 해결해야 하며, 각각에 대한 간단한 해결책을 제시함으로써 이들에 추가적인 연구가 필요함을 보였다.

참고 문헌

1. R. P. Goldberg, "Survey of Virtual Machine Research," IEEE Computer Magazine, vol. 7, p. 34-45, 1974.
2. R. P. Goldberg, "Architecture of virtual machines," In Proceedings of the workshop on virtual computer systems 1973.
3. H. C. Lauer and D. Wyeth, "A recursive virtual machine architecture," In Proceedings of the workshop on virtual computer systems 1973.
4. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," IBM Systems Journal, vol. 11, p. 99, 1972.
5. G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," In Proceedings of the Fourth Symposium on Operating System Principles, 1973.
6. G. Belpaire and N.-T. Hsu, "Formal properties of recursive Virtual Machine architectures," In Proceedings of the fifth ACM symposium on Operating systems principles 1975.
7. G. Belpaire and N.-T. Hsu, "Hardware architecture for recursive Virtual Machines," In Proceedings of the annual conference 1975.
8. R.J. Creasy, "The Origin of the VM/370 Time-Sharing System," IBM Journal of Research and Development, p. 483-490, 1981.
9. J. E. Smith, Ravi Nair, "Virtual machines: Architectures, implementations and applications, " Morgan Kaufmann Publishers, 2004.
10. J. E. Smith, "A unified view of virtualization," In Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments 2005.
11. S. Crosby and D. Brown, "The virtualization reality," Queue vol. 4, p. 34-41, 2006.
12. A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and performance in the Denali isolation kernel, " In Proceedings of the 5th Symposium on Operating Systems Design and Implementation
13. P. Barham, B. Dragovic, et al., "Xen and the art of virtualization," In Proceedings of the 19th Symposium on Operating Systems Principles, p. 164-177.
14. J. LeVasseur, V. Uhlig, B. Leslie, M. Chapman, and G. Heiser, "Pre-Virtualization: Uniting Two Worlds," In Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005.
15. Java Platform, Micro Edition (Java ME), http://java.sun.com/javame/technology/index.jsp
16. P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," In Proceedings of the 10th international conference on Architectural support for programming languages and operating systems 2002.
17. ARM Architecture Reference Manual, Second Edition, edited by David Seal : Addison-Wesley : ISBN-10: 0201737191, ISBN-13: 978-0201737196