임베디드 시스템 상에서의 가상화 기술 적용

김효수○ 신동군
성균관대학교 정보통신공학부
asdfppp3@naver.com, dongkun@skku.edu

Virtualization Technique On Embedded System

Hyosu Kim○ Dongkun Shin
School of Informatin & Communication Engineering, Sungkyunkwan University

요 약

최근 임베디드 시스템과 인터넷 기술과의 접목은 시스템의 신뢰성 측면을 크게 부각시키고 있다. 이에 대한 해결책으로서 임베디드 시스템상에서의 가상화 기술 적용에 대한 연구가 활발히 진행 중이다. 시스템 가상화 기술은 가상 머신 모니터의 통제 하에 각각의 가상 머신을 고립시킴으로서 시스템의 신뢰성을 크게 향상시킬 수 있다. 본 논문에서는 임베디드 시스템 가상화 솔루션의 하나로서 Secure Xen on ARM의 구조와 성능을 분석한다.

1. 서 론

최근 임베디드 시스템이 다양한 분야에서 널리 사용되기 시작함에 따라서 시스템 소프트웨어의 구조가 복잡해지고 다양한 기술들이 요구되고 있다. 특히 모바일 폰의 경우 인터넷 등의 기술과의 접목으로 인해 신뢰성의 문제가 대두되고 있으며, 이에 따라 임베디드 시스템 가상화 기술에 대한 필요성이 높아지고 있다.

시스템 가상화 기술은 하드웨어 자원을 필요에 따라 가상화 시킴으로서 각각의 가상 머신을 만들고, 사용자에게는 독립적인 물리 객체를 가진 것과 같은 착각을 준다. 시스템 가상화 기술에 있어 가장 핵심이 되는 부분은 가상 머신 모니터로서 이는 하드웨어를 고립화하여 각각의 가상 머신을 독립적으로 동작할 수 있도록 관리하는 역할을 한다.

가상 머신 모니터를 통한 시스템 가상화 기술은 전체 시스템의 신뢰성을 크게 향상 시킬 수 있다. 각각의 가상 머신을 서로 안전하게 고립시킴에 따라 하나의 게스트 도메인에서의 고장이 다른 게스트 도메인 혹은 전체 시스템으로 전이되는 것을 막을 수 있는 구조를 지니고 있어, 시스템의 신뢰성을 보장한다.

시스템 가상화 기술이 임베디드 시스템의 보안 문제에 대한 대안이 됨에 따라, 최근 임베디드 시스템 가상화 기술 및 가상화 솔루션에 대한 연구가 활발히 진행되고 있다. 삼성전자에서도 Xen 가상화 솔루션[1]을 ARM 보드에 적용하여 임베디드 시스템 상에서 복수 운영체제 구동을 가능케 하고, 전체 시스템의 신뢰성을 향상시킨Secure Xen on ARM 가상화 솔루션[2]을 개발하였다.

본 논문에서는 Secure Xen on ARM의 구조 및 기존 Xen 가상화 솔루션과의 차이점을 실제 코드 차원에서 분석하고, 간단한 실험을 통해서 성능을 분석하고자 한다. 실험에서는 실제 ARM보드가 아닌 goldfish(android emulator) 를, 가상화 솔루션으로서 Secure Xen on ARM을, 게스트 OS로서 mini-OS를 선택하였고, mioni-OS 스레드 수행 시 실제로 사용되는 명령어의 개수와 클럭 사이클을 통해 성능을 측정하였다.

2. 배경 및 관련 연구

2.1. 시스템 가상화 기술

물리적 하부구조의 가상화를 통해 사용자에게 실제 대상에 대한 환상을 제공하는 시스템 가상화 기술은 가상화 적용 대상에 따라 CPU 가상화, 메모리 가상화, I/O 가상화 3가지로 분류된다. 이 중 가장 중요한 가상화 대상 자원은 CPU로서 크게 전가상화와 반가상화 기법으로 구분된다[7].

CPU 전가상화 기법과 반가상화 기법은 주로 특권모드(privileged mode) 명령어를 처리하는 방법에 따라 구분 된다. 먼저 전가상화 기법은 게스트OS를 가상화 계층에 의해 밑에 있는 하드웨어로부터 완전히 추상화시키는 기술이다. 따라서 게스트 OS는 별도의 수정이 필요하지 않으며, 가상화에 대해서 전혀 인식하지 못한다. 전가상화 기법은 하드웨어적 지원을 통해 특권모드 명령어가 실행될 때 트랩(trap)을 발생시키고, 가상 머신 모니터에서 이를 파악하여 트랩 핸들러를 통해 해당 명령어를 간접적으로 처리한다.

이에 반해 반가상화 기법은 특권모드 명령어들을 처리하기 위해 사전에 하위 가상 머신 모니터 내의 함수를 호출하는 하이퍼 콜로 변환한 후 가상 머신 위에서 실행하는 방법이다. 이는 특권모드 명령어를 하이퍼 콜로의 변환이 필요하므로 OS 커널에 대한 직접적인 수정이 필요하다. 하지만 수정 소요가 반가상화 기법을 통한 성능 향상에 비해 매우 적은 양이므로 큰 문제가 되지 않는다[2]. 이런 반가상화의 대표적인 예로서는Xen 가상화 솔루션이 있다.

시스템 가상화 기술을 사용하는 이유에는 여러 가지가 있다[3,4]. 기존 소프트웨어의 재사용이 가능하다는 점과 복수 운영체제를 하나의 하드웨어 상에서 동시에 구동가능하다는 점도 그 중 하나이다. 하지만, 앞서 말했듯이 서버, 임베디드 시스템 분야에서 시스템 가상화 기술이 많은 주목을 받고 있는 이유는 기존 시스템의 보안 문제를 해결할

수 있기 때문이다[7]. 가상 머신 시스템은 각각의 게스트OS들을 독립적으로 고립시키고, 이를 가상 머신 모니터에서 관리함에 따라, 하나의 게스트 OS에서의 오류가 다른 게스트 OS 혹은 전체 시스템으로 확장되는 것을 막을 수 있어, 전체 시스템의 신뢰성이 크게 향상 된다.

2.2. Xen 가상화 솔루션

최근의 프로세서들의 경우 특권 모드 명령어를 비특권 모드에서 사용해도 트랩을 발생시키지 않음에 따라, Xen 가상화 솔루션에서는 사전에 특권모드 명령어를 하어퍼 콜로 변환하는 반가상화 기법을 사용하여 가상화를 구현하였고, 이를 위해 기존 OS 커널의 약 3,000 ~ 6,000여 라인을 수정하여 Xeno Linux라는 Xen용 게스트 OS를 만들었다[1, 6].

Xen 가상화 솔루션은 구조적 측면에 있어 기존 가상화 솔루션과 커다란 차이를 지니고 있다. 기존의 가상화 솔루션들은 가상 장치에 대해 직접적으로 에뮬레이션을 하는 반면, Xen 가상화 솔루션은 분리 드라이버 모델로서 장치 드라이버를 가상 머신 모니터에서 완전히 분리하여 가상 머신 모니터의 크기를 줄였다. 이에 따라 드라이버 도메인에서 가상 장치에 대한 추상화 인터페이스 제공 및 에뮬레이션을 수행하고, 가상머신 모니터는 단지 두 도메인간의 통신을 제공하는 역할만 하게 되었다.

또한 드라이버 도메인이 분리되었기 때문에 게스트 도메인과 드라이버 도메인 간 통신을 위한 이벤트 채널과 데이터 전송을 위한 I/O 링이 도입되었다. 즉, 데이터 전달을 위한 요청과 응답은 I/O 링을 통해 비동기적으로 교환하고, 실제 데이터의 교환은 공유 메모리를 통해 이루어지게 된다. 이때, 요청과 응답이 어떤 도메인에 쌓여있는지를 알려주기 위한 방법으로서 이벤트 채널을 두었다.

3. Secure Xen on ARM 가상화 솔루션 분석

Secure Xen on ARM 가상화 솔루션의 구조[8]는 그림2와 같다.

그림1에서 볼 수 있듯이, 기존 Xen 가상화 솔루션과 전체적인 구조면에서는 대부분 유사한 반면, 가상화를 지원하는 하드웨어에 있어 큰 차이를 보이고 있다. 기존 Xen 가상화 솔루션의 경우 x86 프로세서를 기반으로 하여 가상화를 구현한 반면, Secure Xen on ARM 가상화 솔루션은 가상화를 적용할 타겟 보드로서 ARM 프로세서를 선택하여 이를 구현하였다. 이에 게스트 OS 또한 Xen 가상화 솔루션을 위한 Xeno Linux 대신, ARM Linux와 실시간 OS인 μCOS의 커널을 수정하여 Secure Xen on ARM 가상화 솔루션 위에 포팅하였다.

3.1. ARM 프로세서에서의 Xen 가상화 기술 고려사항

앞서 보았듯이 구조적 측면에서 볼 경우 가상화를 지원하는 하드웨어의 변경으로 인한 게스트OS의 변화만이 눈에 띄나, 사실 내부적인 측면에서 볼 때 임베디드 시스템, 특히 ARM 프로세서로의 Xen 가상화 솔루션 적용 시 고려하여야 할 사항들이 많이 존재한다[2].

첫째, 임베디드 장치의 경우 데스크탑이나 서버와 비교하였을 때, 하드웨어 자원이 제한되어 있다. 특히 제한된

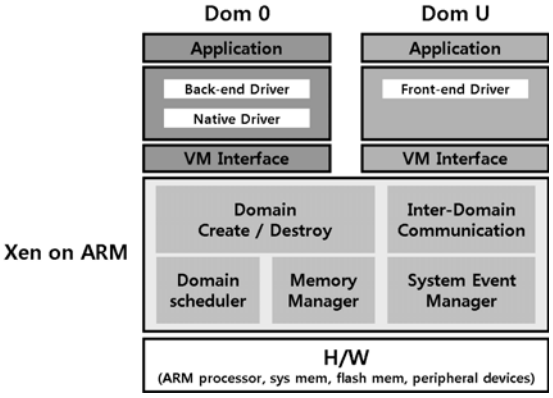

그림 1 Secure Xen on ARM 구조

전력으로 인해 임베디드 시스템에서의 효율성은 매우 중요한 이슈가 되고 있다. 임베디드 시스템에 가상화 기술을 적용할 시에도 이 효율성 측면은 간과할 수 없는데, 실시간적으로 특권모드 명령어에 대해 이진변환[7]을 요구하는 전가상화 기법에 비해 사전에 OS커널을 수정하여 특권모드 명령어를 처리하는 반가상화 기법이 실제 동작시 더 높은 효율성을 지니게 된다. 그러므로 반가상화 기법을 사용하는 Xen 가상화 솔루션의 경우, 효율성 측면에 있어 전가상화 기법을 사용하는 다른 가상화 솔루션에 비해 임베디드 시스템에 더 적합하다고 볼 수 있다.

둘째, ARM 프로세서의 경우 7가지의 CPU 동작 모드를 지원하나, 특권 모드의 개념에서 봤을 때, USR 모드만이 비특권 모드이고, 이를 제외한 나머지 6가지 모드 모두 특권 모드이다. 가상화 개념에서 볼 때, 하드웨어 자원 관리 및 접근에 대한 독점적인 권리를 가지는 가상 머신 모니터가 특권 모드에서 동작을 하고, 게스트 OS와 사용자 프로세스는 그 윗 단인 비특권 모드에서 동작을 하여야 한다. 하지만, ARM 프로세서의 경우 1개의 비특권 모드만이 지원되므로 게스트 OS와 사용자 프로세스가 같은 모드에서 동작을 하게 된다. 이로 인해, 가상 머신 모니터 영역을 게스트 OS 커널로부터 보호할 때, 특권 모드, 비특권 모드가 구분되므로 기존 Xen 솔루션에서와 마찬가지의 방법으로 보호가 가능하나, 사용자 프로세스로부터 게스트OS 커널을 보호할 때에는 문제가 발생하게 된다.

셋째, 사용자 프로세스와 게스트OS 커널의 같은 모드에서의 실행은 시스템 콜 구현 시에도 문제를 야기한다[5]. 같은 모드에서 동작하므로, 사용자 프로세스에서 게스트OS 커널에 대한 접근을 제한하기 위해 페이지 테이블에는 커널 주소 영역에 대한 정보를 보유하지 않게 된다. 이로 인해 시스템 콜 등을 사용하여 커널에 접근할 때, 커널 모드 권한 획득 및 접근은 기존 Xen에서의 시스템 콜 처리 방식과는 달리 간접적으로 가상 머신 모니터를 통해서만 처리가 가능해진다.

마지막으로, ARM 프로세서의 경우 x86 프로세서의 PIPT (Physically Indexed Pysically Tagged) 캐시와는 다르게 VIVT(Virtually Indexed Virtually Tagged) 캐시를 지원하므로, 하나의 가상 메모리 주소에 대해서 여러 프로세스에서 사용할 수 있고, 이로 인해 앨리어싱 문제가 발생할 수 있다. 이러한 문제를 해결하기 위해서는 주소 변환시마다 캐시를 flush 해주어야 하는데, 이 때 상당한 오버헤드가 발

생하게 된다. 가상화 기술을 적용하기 위해 단순히 게스트 OS, 사용자 프로세스의 메모리 영역을 나누어 저장관리하는 기법을 사용하게 될 경우, 게스트 OS 커널이나 사용자 프로세스 메모리 등으로의 주소 변환 시 캐시 전체를 flush 해야 하는 문제가 빈번히 일어나고 이로 인해 시스템의 효율성이 현저하게 떨어지게 되는 문제가 발생한다.

3.2. 기존 Xen 가상화 솔루션과의 차이점

ARM 프로세서에서 Xen 가상화 기술을 바로 적용할 경우 생기는 문제점들을 해결하기 위해 Secure Xen on ARM 가상화 솔루션은 약 22,000 여 라인을 수정 및 추가하였고, 8개의 새로운 하이퍼 콜을 추가하여 이를 구현하였다[9].

3.2.1. 가상 특권 모드

Secure Xen on ARM 가상화 솔루션에서는 ARM 프로세서에서의 제한된 수의 비특권 모드로 인한 게스트 OS 커널의 사용자 프로세스로부터의 비보호 문제를 해결하기 위해 가상 특권 모드를 새롭게 생성하였다. 기존 비특권 모드(USR 모드)를 각각 커널 모드와 사용자 모드로 분할하고 이를 가상 머신 모니터에서 관리함에 따라, 게스트 OS 커널은 자신이 특권 모드에서 동작중이라는 착각을 가질 수 있게 되었으며, 게스트 OS 커널의 추가적인 수정을 최소화 할 수 있게 되었다.

하지만 기존의 페이징 메커니즘으로는 논리적으로 분할된 가상 커널 모드와 사용자 모드간의 보호 문제를 해결하기에는 한계가 있다. 실제 게스트 OS 커널과 사용자 프로세스가 사용하는 메모리 공간의 주소를 정확히 알 수 없을 뿐만 아니라, 설사 안다고 하더라도, 굉장히 비효율적이기 때문이다. Xen on ARM 가상화 솔루션에서는 이를 해결하기 위해서 ARM 프로세서의 도메인 보호 메커니즘을 사용하였다. 가상 머신 모니터, 게스트 OS 커널, 사용자 프로세스 영역을 각각 다른 도메인에 할당하고, 타 도메인에 대한 접근 권한을 도메인 접근 제어 레지스터(DACR)을 통해 제어함에 따라서 게스트 OS 커널을 사용자 프로세스로부터 보호하였다.

가상 특권 모드와 도메인 메커니즘 사용에 있어 Secure Xen on ARM 논문[2]과 실제 구현상에는 약간의 차이점이 존재하였다. 우선 가장 큰 차이점은 기존 가상 머신 모니터, 게스트 OS, 사용자 프로세스 도메인 영역 외에 I/O 도메인 영역이 추가되었다. 이로 인하여 D0, D1, D2에 각각 할당 되었던 가상 머신 모니터, 게스트 OS, 사용자 프로세스 도메인 영역들이 D0, D1, D3에 할당되었고, I/O 도메인이 D2에 새롭게 할당되었다. 도메인 간 접근 권한 또한 I/O 도메인에 대한 접근 권한이 새롭게 추가되었으며, 기존 도메인간의 접근권한 또한 약간 변경되었다. 실제 구현상에서의 접근권한은 표 1과 같다.

특권 모드에서 작동하는 가상 머신 모니터의 경우 모든 영역에 대해 client 권한을 가지게 되었다. 비록 해당 페이지 테이블의 접근 권한을 살펴보아야 하는 client 권한이지만, 특권 모드에서 비특권 모드로의 접근이므로 접근에 대한 아무런 제약이 따르지 않게 된다. 하지만, 그 반대의 경우, 커널 모드와 사용자 모드도 가상 머신 모니터 도메인에 대해 client의 권한을 가지지만, 비특권 모드에서 특권 모드로의 직접적인 접근이 제한되므로 가상 머신 모니터 도메인 영역을 커널 모드와 사용자 모드로부터 보호할 수 있다. 또한 가상 머신 모니터 도메인 영역을 제외한 나머지 영역에 대해 커널 모드는 manager권한을 가지는데, 이는 커널 모드가 사용자 프로세스, I/O 등에 대해 직접적인 제어 권한을 지닌다는 것을 의미한다. 반면 사용자 모드의 경우 모든 도메인 영역에 대해 client 권한을 가지나, 다른 도메인 영역들에 대해 페이지 테이블의 접근권한(AP)이 no access로 설정되어 있어 접근이 불가능하다. 이에 대한 자세한 구현 사항은 xen/include/asm-arm/cpu-domain.h에 나와 있다.

표 1 실제 구현된 도메인 간 접근권한

	VMM (D0)	커널 (D1)	사용자 (D3)
VMM (D0)	client	client	client
커널 (D1)	client	manager	client
I/O (D2)	client	manager	client
사용자 (D3)	client	manager	client

3.2.2. ARM 가상 캐시 최적화

VIVT 캐시로 인해 주소 변환시마다 캐시 flush 오버헤드가 발생하는 문제를 해결하기 위한 방안으로 Secure Xen on ARM 가상화 솔루션에서는 가상 머신 모니터, 게스트 OS 커널, 사용자 프로세스 각각에게 고정된 가상 메모리 주소를 할당하였다. 가상 메모리 주소가 고정됨에 따라 가상 머신 모니터로의 접근, 가상 머신 모니터로부터의 리턴 시에 별도의 주소 변환이 필요하지 않게 되었고 이에 따라 캐시 flush또한 필요하지 않게 되었다. 결국 가상 머신 모니터가 현재 처리중인 게스트 도메인을 바꾸거나, 게스트 OS가 프로세스를 바꿀 때에만 주소 변환 및 캐시flush가 발생하므로 그 발생 빈도 및 오버헤드가 현저하게 줄어들게 되었다.

소스 코드에서 실제로 구현된 총 4GB의 가상 주소 구조는 그림 2와 같다.

할당된 가상 주소 공간을 자세히 살펴보면, 먼저 가상 주소의 상위 64MB가 하이퍼바이저, 가상 머신 모니터를 위해 할당되었다. 하이퍼바이저의 올바른 동작을 위해 내부적으로도 가상 주소가 각각 용도에 따라 분할되었는데 동적으로 가상 주소를 물리 주소로 맵핑해주는 ioremap()을 위한 영역이 하이퍼바이저의 가장 상위 4MB에 할당되었다. 그 다음 12MB는 DIRECTMAP영역으로 전체적인 동작을 관리하는 모니터 등의 물리 주소와 맵핑되어 있다. DIRECTMAP영역의 하위 8MB는 PERDOMAIN 영역에 할당되었는데, 이는 게스트 도메인을 위한 공간으로서 게스트 도메인만이 사용가능하기 때문에 하이퍼바이저에서는 이 영역을 비워두었다. 또한 그림 2에서 볼 수 있는 것처럼, PERDOMAIN 영역내의 상위 4MB에는 MAPCACHE라는 별도의 영역이 존재한다. 이는 게스트 도메인에 할당된 물리 페이지를 임시적으로 가상 머신 모니터의 주소 공간과 맵핑시키기 위한 용도로 사용된다. 4번째와 5번째 영역인 SH_LINEAR_PT와 LINEAR_PT영역의 경우 각각 4MB의 가상 주소가 할당되었는데, 이는 선형 페이지 테이블과의 맵핑 즉, 하이퍼바이저의 L1 페이지 테이블에 접근을 위해 사용된다. RDW

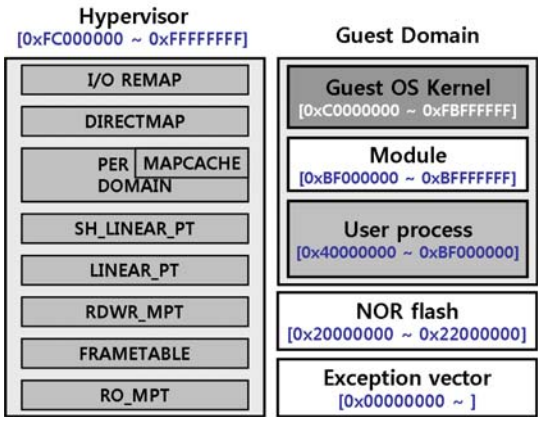

그림 2 Secure Xen on ARM 가상주소

R_MPT 영역이 다음 하위 4MB에 할당되었고, 이는 Read-Write Machine-to-Physical 맵핑 테이블로서 맵핑된 물리 주소의 경우 특권 모드에서는 읽기 쓰기 권한을 모두 가지지만, 비특권 모드에서는 읽기 쓰기 권한 모두 없는 특징을 지닌다. 그 다음 영역은 하이퍼바이저에서 가장 큰 영역으로서 24MB가 실제 물리 메모리의 페이지 프레임 관리를 위한 프레임 테이블에 할당되었다. 마지막으로 하위 4MB가 RO_MPT영역에 할당되었는데, RDWR_MPT와는 달리 테이블에 맵핑된 물리 주소 영역에 대해 비특권 모드에게도 읽기 권한이 주어진다.

하이퍼바이저의 하위 영역은 게스트 OS 커널과 사용자 프로세스를 위해 각각 0xC0000000 ~ 0xFBFFFFFF, 0x40000000 ~ 0xBEFFFFFF의 가상 주소가 할당되었고, 이 둘의 사이인 0xBF000000 ~ 0xBFFFFFFF가 XIP 커널을 위한 모듈의 가상 주소로서 사용되었다. 이 외에도 exception 벡터를 위한 가상주소가 0x00000000부터 0번 exception부터 차례차례 할당되었으며, NOR 플래시에 대한 가상 주소로서 0x20000000 ~ 0x22000000이 할당되었다.

이러한 구조에 대한 정의 및 구현 중 하이퍼바이저와 게스트 도메인 구분은 xen/include/asm-arm/config.h에 게스트 도메인내 게스트 OS 커널과 사용자 프로세스에 대해서는 linux-sparse/include/asm-arm/memory.h에 정의되어 있다.

3.2.3. Exception 처리

x86 프로세서에서 ARM 프로세서로 타겟 보드가 변경됨에에 따라서 exception을 처리하는 메커니즘에도 변화가 발생했다. 실제 구현된 exception 중 소프트웨어 인터럽트(swi)의 처리 메커니즘을 살펴보면 다음과 같다.

가장 먼저 커널 모드에서 특권 모드 명령어 사용을 위해 하이퍼 콜을 사용하거나, 사용자 모드에서 커널 모드로의 접근 등을 위해 사용하는 시스템 콜을 통해 소프트웨어 인터럽트가 발생하게 되면 vector_swi() 어셈블리 함수를 통해 현재 SP, SPSR, PC, LR 등을 스택에 저장한다. 특히 SP의 경우, 현재 exception을 발생시킨 곳이 커널 혹은 사용자 모드인지 파악하여 각각을 해당 모드의 VSP에 저장하고, SPSR도 현재 모드에 맞게 값을 재설정하여 저장한다. 또한 현재 발생한 exception이 하이퍼 콜인지 아닌지 여부를 판단하여 하이퍼 콜일 경우 process_hypercalls()를, 반대의 경우 do_upcall()을 호출한다.

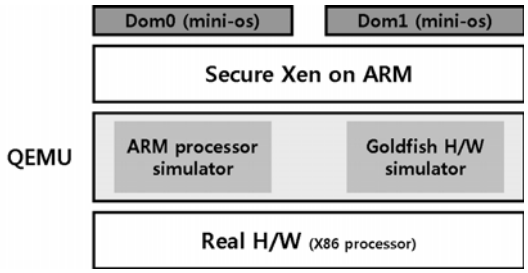

그림 3 Goldfish상에서의 Secure Xen on ARM 구조

process_hypercalls()를 통해 하이퍼 콜을 호출한 경우, 하이퍼 콜 테이블을 참조하여 해당 요청에 대한 처리를 완료한 후 ret_from_hypercall(), ret_to_user(), restore()의 과정을 거쳐 가상 머신 모니터에서 다시 본래의 모드로 돌아오게 된다.

반면 시스템 콜의 경우 하이퍼 콜과는 다르게 사용자 모드에서 요청한 사항을 커널 모드에 간접적으로 알려주기 위해서 업 콜이 새롭게 추가된다. 이를 위한 do_upcall() 어셈블리 함수를 살펴보면, 가상 머신 모니터에서 커널 모드로 이동하는 것이므로, VKSP, SPSR을 커널 모드에서 사용할 레지스터에 각각 저장한다. 특히 커널 모드에서 직접적으로 FAR(Fault Address Register), FSR(Fault Status Register)에 대한 접근이 불가능하므로, 이를 위해 가상으로 둔 VFAR, VFSR을 레지스터에 저장하여 이를 간접적으로 사용한다. 커널 모드에서 업 콜로 요청된 사항을 모두 처리하게 되면, 커널은 하이퍼 콜을 발생시켜 시스템 콜 요청 시 저장했던 정보들을 복원하고, 다시 사용자 모드로 돌아가게 된다.

이러한 exception 처리와 관련하여 xen/arch/arm/xen/entry.S에서는 업 콜, IRQ 등의 인터럽트 처리에 대해서 구현되어있고, hypercall.S에서는 소프트웨어 인터럽트 발생과 context 저장 및 복원 등에 대해서 자세히 구현되어 있다.

4. 실험 결과

Secure Xen on ARM 가상화 솔루션의 경우 Freescale imx21 보드에 맞게 특화되어 있다[2]. 이로 인해 지원하는 보드가 아닌 타 보드에서는 포팅 및 실험 측정이 힘들다는 단점이 있다. 하지만, 최근 릴리즈된 버전에서 goldfish라는 QEMU 0.82[10]기반의 ARM926T 프로세서 안드로이드 에뮬레이터를 새롭게 지원함에 따라, 실제 하드웨어가 지원하는 보드가 아니더라도 가상으로 이를 시뮬레이션을 할 수 있게 되었다. 이러한 구조는 그림 3과 같다.

이번 실험에서는 이러한 goldfish의 구조적 장점을 이용하여, goldfish 상에 Secure Xen on ARM 가상화 솔루션을 포팅하여 시뮬레이션을 진행하였다. 우선 게스트 OS인 mini-os에서 수행되는 하나의 스레드의 시작과 종료를 단위 태스크로 정하였고, 각각의 게스트 도메인에서 4개의 스레드를 번갈아 실행하면서, 그 동안 수행된 명령어의 개수와 클럭 사이클을 trace로 출력하여 성능을 측정하였다. Goldfish가 일종의 가상 머신이기 때문에, 추가적인 오버헤드가 발생하여 시간적으로는 원하는 결과 값의 측정이 힘들다고 생각하여, 수행된 명령어의 개수와 클럭 사이클을 성능 측정 지표로서 선택하였다. 또한, 실제로 측정해야하는 구간

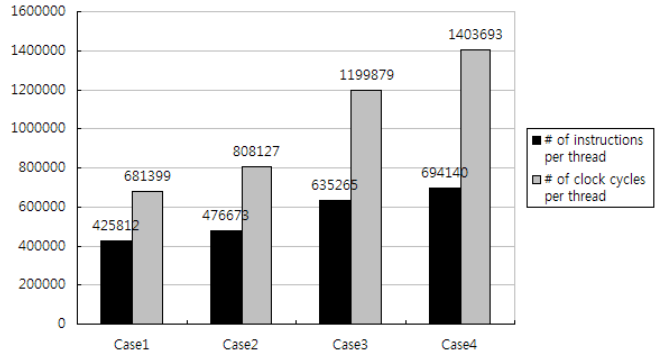

그림 4 측정된 명령어의 개수와 클럭 사이클 평균

을 정확히 구분하고자, 단위 태스크 종료 후 소프트웨어 인터럽트를 발생시켜 trace에 추가적으로 표시하였다. 실제 측정되는 단위 태스크, 스레드의 동작은 다음과 같다.

```
printk("%d thread\n", thread.id);
for(i=0; i<100000; i++);
prink("After %d thread \n", thread.id);
for(i=0; i<100000; i++);
__asm__volatile__("swi #0xffffab");
// a : domain id, b : thread id
schedule();
```

위에서 볼 수 있듯이 스레드는 시작 메세지 출력- 100,000회 루프 - 종료 메세지 출력 - 100,000회 루프 - 소프트웨어 인터럽트 발생- 대기 중인 다음 스레드 실행의 라이프 사이클을 가진다. 특히, 다음 스레드를 실행하기 직전에 소프트웨어 인터럽트를 발생시킴으로서 각각의 스레드를 구분할 수 있게 하였고, 이를 기반으로 하여 전체 스레드의 명령어의 개수, 클럭 사이클 평균을 계산하였다.

이렇게 측정한 성능 값을 비교 분석하기 위해실제 실험에서는 4가지 경우에 대해서 시뮬레이션을 수행하였다. 첫째, goldfish상에 직접 게스트 OS를 포팅한 경우, 둘째, Secure Xen on ARM을 포팅하고, 게스트 도메인을 도메인0만 포팅한 경우, 셋째, 도메인 0와 도메인 1을 포팅한 경우, 마지막으로 도메인 2까지 포팅한 경우 이렇게 4가지 시뮬레이션을 통해 실험 결과를 측정 비교하였다. 이러한 실험 측정 결과는 그림 4와 같다.

그림 4에서 볼 수 있듯이, Secure Xen on ARM 가상화 솔루션을 사용할 때(case1)와 사용하지 않을 때(case0)에 있어 약 12%의 추가적인 명령어가 사용된 것을 볼 수 있다. Secure Xen on ARM 가상화 솔루션을 통해, 게스트 OS가 하드웨어 자원을 사용하고 실행되므로 추가적인 오버헤드가 발생한 것이다. 또한, 포팅된 가상화 머신의 개수 측면에서 분석해보면, 가상 머신의 개수가 많아질수록 요구되는 명령어의 개수도 증가하는 것을 볼 수 있다. 이는 가상 머신 개수의 증가로 인한context switch 횟수의 증가가 영향을 끼쳤다고 볼 수 있다. 즉, 가상 머신 모니터에서 n개의 가상 머신에게 1개의 하드웨어 자원을 1/n개로 나누어 균등하게 배분해야 되므로, 가상 머신의 개수(n)의 증가로 인해 게스트 도메인 변경이 더욱 빈번하게 발생되는 것이다.

클럭 사이클 또한 명령어 개수와 비슷하게 Secure Xen on ARM 사용 유무에 따라, 가상 머신 개수에 따라 증가하는 것을 볼 수 있다. 게다가 CPI(Clock cycle per Instruction) 측면에서 볼 경우 각각 case1 : 1.6, case2 : 1.7, case3 : 1.89, case4 : 2.02로 증가하는 것을 볼 수 있다. 즉, 가상화 솔루션의 사용 혹은 가상 머신 개수의 증가로 인한 오버헤드, 특히 context switch 등이 더욱 빈번하게 발생하여, 이로 인해 성능 저하가 발생하는 것이다.

5. 결 론

본 논문에서는 임베디드 시스템의 신뢰성 향상을 위한 가상화 솔루션으로서 Secure Xen on ARM을 살펴보았다. 특히 기존의 Xen 가상화 솔루션을 ARM 프로세서에서 포팅하기 위해 실제로 구현한 사항들과goldfish를 이용한 시뮬레이션 실험을 통해Secure Xen on ARM 가상화 솔루션의 구조와 성능을 분석하였다.

비록 가상화 솔루션의 적용으로 인해 오버헤드가 발생하고, 성능이 저하된다는 단점이 있지만 앞서 말했듯이 가상화 솔루션을 통한 신뢰성의 강화와 다양한 OS의 동시 구동, 소프트웨어 재사용의 장점은 이를 충분히 보완할 수 있을 것이라고 생각된다. 하지만, 실험을 하면서 느꼈듯이, 임베디드 시스템 각각이 가진 하드웨어적 특성이 상이하여 임베디드 시스템 가상화 솔루션의 범용화 및 이식성의 문제가 아직 남아있다. 또한 제한된 하드웨어 자원을 효율적으로 사용하기 위해 가상화 솔루션 오버헤드도 더욱 감소시킬 수 있는 방안을 찾아야 할 것이다.

참고문헌

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, Xen and the Art of Virtualization, Proceedings of the ACM Symposium on Operating Systems Principles, pp. 164-177, 2003.

[2] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, Chul-Ryun Kim, Xen on ARM:System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones, IEEE CCNC, pp. 257-261, 2008.

[3] Gernot Heiser, The Role of Virtualization in Embedded Systems, First Workshop on Isolation and Integration in Embedded Systems, 2008.

[4] Yuki Kinebuchi, Hidenari Koshimae, Shuichi Oikawa, Tatsuo Nakajima, Virtualization Techniques for Embedded Systems, 2006.

[5] Daniel R. Ferstay, Fast Secure Virtualization for the ARM Platform, the faculty of graduate studies at the university of British columbia, 2006.

[6] 유시환, 유혁, 실시간 내장형 시스템 가상화 연구 동향, 정보과학회지, 제26권, 제10호, 41-49쪽, 2008. 10.

[7] 홍대영, 고원석, 임성수, 보안과 신뢰성있는 컴퓨팅을 위한 가상화 기술, 정보과학회지, 제26권, 제10호, 50-57쪽, 2008. 10.

[8] Sang-Bum Suh, Secure Architecure and Implementation of Xen on the ARM 9 for mobile devices, http://www.xen.org/files/xensummit_4/Secure_Xen_ARM_xen-summit-04_07_Suh.pdf, Xen Summit April, 2007.

[9] Sang-Bum Suh, Secure Xen on ARM, http://www.xen.org/files/xensummit_fall07/14_SangBumSuh.pdf, Xen Summit November, 2007.

[10] F. Bellard. QEMU, a fast and portable dynamic translator. Proceedings of the USENIX Annual Technical Conference,2005.