실시간 내장형 시스템 가상화 연구 동향

고려대학교 ▌유시환* · 유 혁**

1. 실시간 내장형 시스템에서 가상화 연구의 필요성

가상화 기술은 다중 사용자를 위한 서버 활용도를 높이기 위한 노력에서 출발하여 네트워크 가상화, 스토리지 가상화 등으로 다양한 분야로 적용 범위를 넓혀가고 있다. 가상화란 실재하는 대상에 대하여 사용자에게 독립적인 환영(illusion)을 제공하는 기술로서 각 사용자들은 물리적인 대상의 상태나 위치와 무관하게 자신만의 독립적인 컨텍스트 정보를 가짐으로써 마치 개개의 사용자가 모두 자신만의 개체를 가진 것처럼 느끼게 해 주는 기술이다. 최근까지의 시스템 가상화 기술은 서버와 같이 고성능 시스템의 활용율을 높이는데 초점을 맞추어왔다.

최근 내장형 시스템의 발전은 대단히 주목할 만하다. 내장형 시스템들은 범용 시스템들과 달리 제한된 목적을 가지고, 주어진 응용 프로그램들만을 동작시킬 수 있도록 설계되었다. 하지만, 내장형 시스템에서 사용되는 하드웨어의 발전은 임베디드 시스템이 가진 자원 제약을 극복할 수 있는 계기를 마련하도록 하였다. 기존의 휴대전화, 가전제품 등에서 사용되던 내장형 소프트웨어들은 하드웨어 성능의 발전에 따라 더욱 많은 기능을 추가하게 되었고, 이 결과 점차로 새로운 형태의 내장형 소프트웨어 개발 방법론이 필요하게 되었다. 내장형 소프트웨어는 그 특성 상, 한번 시스템에 장착되어 실행되면, 수정과 디버깅 등이 매우 불편하며, 새로운 응용의 추가 등이 어려운 단점이 있다. 특히 최근의 컨버전스 추세로 인한 휴대형 내장형 기기들의 특징은 하드웨어 장치만으로는 그 특성을 규정하기 힘들만큼 다양한 형태로 발전하고 있다. 이러한 기능 확장은 내장형 시스템에서 사용되는 소프트웨어에도 큰 변화를 가져왔다.

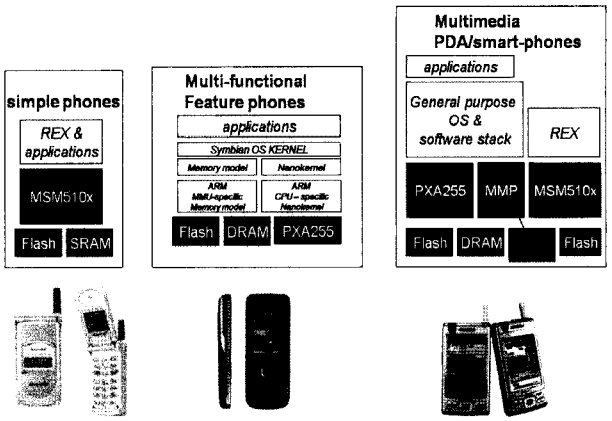

그림 1 휴대전화와 사용 운영체제의 변화

내장형 시스템에서 사용되는 운영체제들은 대개 매우 단순하여, 단순한 역할을 하는 응용 프로그램들에 큰 오버헤드를 주지 않도록 설계되었다. 하지만, 복잡한 응용 프로그램들이 동시에 동작하게 됨에 따라 응용 프로그램들을 관리해 줄 수 있는 운영체제 역시 범용 운영체제에 버금가는 강력한 운영체제의 필요성을 낳게 되었다. 휴대전화에서 사용되는 운영체제의 변화와 발전의 추이를 통해, 이러한 특성을 쉽게 알 수 있다.

그림에서 볼 수 있듯이 휴대전화에서 사용되는 운영체제는 매우 단순한 모뎀 컨트롤러 역할을 하는 RexOS와 같은 운영체제에서 출발하여, 심비안과 같이 UI를 지원하는 운영체제를 거쳐 최근에는 임베디드 리눅스나 Windows Mobile 등과 같은 다양한 기능을 제공하는 운영체제로 발전되어 왔다. 하지만, 현재까지 범용 운영체제는 내장형 시스템에서 사용되는 임베디드 운영체제와 다른 여러 가지 특징들을 가지고 있어 추가적인 하드웨어의 도움을 받거나 기능을 제한하는 등의 한계성을 가지고 있다.

대개 임베디드 운영체제는 정해진 응용의 특성만을 고려하여 설계, 제작되기 때문에 범용성은 떨어지지만, 해당 응용에 대하여 최적화되어있다. 반면 범용 운영체제는 다양한 응용 프로그램을 실행하기 위해 임

* 학생회원
** 종신회원

베디드 운영체제보다 일반적인 프레임워크나 구조를 가지고 있는 경우가 많다. 따라서, 응용 프로그램 입장에서 본 상대적 성능은 임베디드 운영체제를 사용한 경우와 범용 운영체제를 사용한 경우에 차이가 날 수 밖에 없다. 특히, 실시간 임베디드 운영체제의 경우 실시간성을 보장하기 위한 여러 가지 기법들을 가지고 있으며, 범용 운영체제가 가지지 않은 제약들을 가짐으로 인해서, 실시간성에서 아직까지도 큰 차이를 보이고 있다. 최근의 하드웨어 발전으로 인해, 별도의 DSP나 멀티-코어 구조 등을 이용하여 이러한 한계를 극복하려는 노력이 시도되고 있다.

범용 운영체제의 도입은 성능 상의 문제뿐만 아니라, 시스템의 재개발과 관련된 문제를 가져온다. 즉, 기존에 사용하던 내장형 시스템의 소프트웨어의 재개발이 필요하다. 특히 내장형 시스템에서는 소프트웨어의 디버깅과 개발이 불편한 점을 고려해 볼 때, 소프트웨어의 재개발은 큰 문제가 된다. 대개의 임베디드 소프트웨어들이 동작하는 플랫폼과 밀접하게 동작하여, 높은 의존성을 가지고 있으며 임베디드 운영체제의 종류에 따라 서로 제공하는 추상화 계층이나 기능들이 모두 다른 점을 고려해 볼 때, 소프트웨어를 범용 운영체제로 재개발하기 위한 노력에 필요한 비용이 매우 큼을 예상해 볼 수 있다.

더욱이 내장형 시스템에서는 하드웨어 구성이 매우 다양한 점을 고려해 볼 때, 각 하드웨어 플랫폼에 따라 각기 다른 장치 드라이버와 보드 지원 패키지(Board Support Package)를 개발하는 일은 매우 큰 어려움으로 다가온다. Windows mobile은 이러한 어려움을 플랫폼을 표준화함으로써 해결하려고 하였으나, 휴대형 내장형 시스템의 다양성과 특성들을 고려해 볼 때 이 역시 어려운 일임은 자명하다.

재사용의 문제는 소프트웨어 신뢰성의 문제와도 관련된다. 소프트웨어의 재개발은 개발에 필요한 시간뿐만 아니라 검증과 평가에 따른 시간을 필요로 함이 널리 알려져 있다. 특히, 써드-파티에 의한 장치 드라이버는 운영체제 영역에서 특권 레벨을 가지고 실행되기 때문에 시스템 전체의 신뢰성에 큰 영향을 끼치게 되며, 이를 검증/평가하기 위해서는 매우 많은 비용이 필요하다.

이러한 문제를 보다 쉽게 해결하는데 내장형 시스템의 가상화 연구가 큰 도움이 될 수 있다. 즉, 기존의 소프트웨어를 변경하지 않고 재사용 가능한 가상화 구조의 특성을 활용하여, 가상 머신 환경에서 서로 다른 종류의 응용들을 변경 없이 사용할 수 있는 큰 장점을 갖게 된다.

뿐만 아니라 가상머신 시스템이 가진 구조적인 장점을 통해 소프트웨어 플랫폼의 신뢰성을 획기적으로 향상시킬 수 있는 계기가 될 수 있으며, 서로 다른 내장형 응용 프로그램들을 하나의 시스템에서 공통적으로 사용할 수 있는 차세대 소프트웨어 플랫폼으로 고려해 볼 수 있다.

2. 배경지식 및 관련 연구 - 시스템 가상화

시스템 가상화 기술[1]은 컴퓨터 시스템의 물리적 하부구조를 가상화하여 제공하는 기술을 의미한다. 컴퓨터 시스템의 물리적 하부구조는 컴퓨터 시스템이 받아들일 수 있는 기계어 수준의 인터페이스인 명령어-셋-구조(ISA, Instruction Set Architecture)를 가상화 하는 작업에서부터 시작된다. 이는 상위의 소프트웨어들이 각각 개별적으로 동작하도록 하기 위해 하드웨어의 상태를 인지할 수 있는 모든 인터페이스를 차단해야 하기 때문이다.

가상화 기법은 에뮬레이션과 유사하지만, 일부 명령어들은 직접 하드웨어에서 동작한다는 점에서 에뮬레이션과 다르다. 에뮬레이션은 하드웨어적인 모든 동작을 소프트웨어로 구현하는 반면, 가상화는 일부 명령어에 대해서만 에뮬레이션을 적용하게 된다. 이를 위해 Goldberg/Popek은 가상화를 제공할 수 있는 하드웨어 구조에 대하여 정의하였다. 이들은 가상화-민감 명령어(virtualization-sensitive instruction)들을 다음과 같은 두 종류로 정의하였다.

- -. 제어-민감 명령어(control-sensitive instruction): 명령어 자체가 가상메모리의 기저 주소 등을 변경하는 등의 중앙처리장치(CPU)의 주요한 상태(또는 레지스터 값)를 변경하는 명령어. 예) 특권 레벨을 변경하거나 가상메모리의 기저 주소를 변경하는 명령어는 CPU의 상태를 변경하는 명령어이므로, 제어-민감 명령어이다.
- -. 동작-민감 명령어(behavior-sensitive instruction): CPU 상태에 따라 다른 동작을 하는 명령어. 예) 메모리 읽기/쓰기는 가상 주소 공간이 변경되면, 물리 메모리의 서로 다른 주소에 접근하게 되므로, 동작-민감 명령어이다.

Goldberg/Popek의 정의에 따르면[2], 가상화를 지원하는 하드웨어 구조는 모든 가상화-민감 명령어들이 특권 명령어로 정의되어있어야 한다. 특권 명령어는 특권레벨에서만 실행이 가능하며, 비특권 모드에서 실

행한 경우 트랩이 발생하는 명령어들이다. 즉, 비특권 모드에서 특권 명령어를 실행하면, 예외상황으로 인지하여 CPU의 제어를 바로 트랩 핸들러로 옮기게 된다. 가상화를 지원하는 가상머신 구조는 모든 가상화-민감 명령어들을 가상머신의 제어 하에 실행하도록 하기 위해 각 게스트 운영체제들을 비특권 모드에서 동작시키며, 트랩을 발생시킨 명령어는 가상머신 모니터가 하드웨어를 대신하여 간접 실행을 하게 된다.

이외의 명령어들은 일반 명령어(innocuous instruction)으로 분류하여 에뮬레이션 없이 하드웨어를 통해 직접 실행을 허용함으로써 성능 저하를 막는다. 에뮬레이션과 달리 가상화 기법은 일부 명령어에 대해서만 에뮬레이션을 함으로써, 각 가상머신의 CPU 상태를 독립적으로 유지하면서도 성능 저하를 최소화 할 수 있다.

가상머신 시스템의 구조는 그림 2와 같다.

가상머신은 독자적인 CPU 상태와 가상 I/O 장치들을 가지고 있으며, 물리머신은 가상화 계층에 의해 완전히 가려져 있어, 상위의 소프트웨어들은 자신이 가상머신에서 동작하는지 조차 알 수 없다. 이러한 구조에서는 가상머신 시스템이 제공해 주는 명령어-셋-구조에 의해 상위의 소프트웨어가 동작하기 때문에, 물리머신의 ISA와 상위 소프트웨어의 ISA가 완전히 분리가 가능하다.

또한, 동일한 ISA를 사용하더라도 서로 다른 가상머신들은 완벽히 분리되어, 서로 다른 하드웨어 상태 정보를 갖게 되므로, 이종 운영체제가 동작하더라도 서로 영향을 주지 않게 된다. 이를 위해서는 가상머신 모니터가 물리머신의 동작을 에뮬레이션 해 주어야 하며, 각 가상머신의 독립된 상태 정보를 유지/관리하는 역할을 해 주어야 한다. 이러한 특성은 가상머신 구조의 강한 고립성(strong isolation)을 보장해 주어, 상위의 응용 프로그램들에 대한 높은 보안성을 제공할 수 있는 기반을 제공한다.

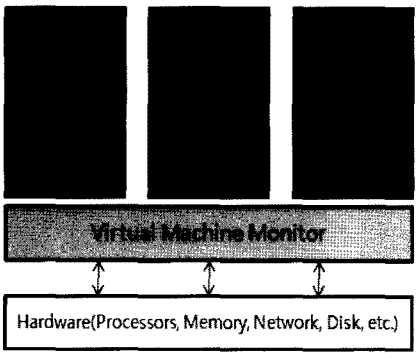

그림 2 가상머신 시스템의 구조

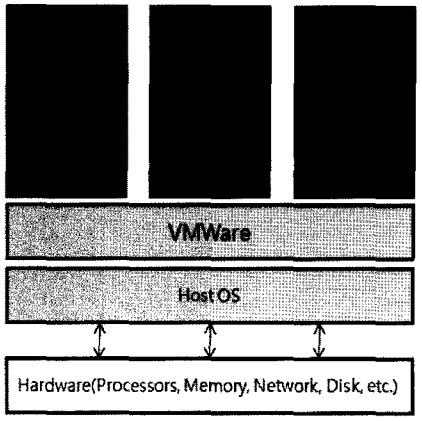

그림 3 VMWare 구조

가상머신의 동작을 감시하며, 가상머신들 간의 독립성을 보장하는 가상머신 모니터는 가상화 계층 혹은 가상화 레이어로도 불리며, 물리머신의 상태를 직접 제어한다.

가상머신 시스템의 구조 중에서 변형된 것으로는 그림 3과 같은 형태도 존재한다[3].

그림 3에 나타난 가상머신 시스템의 구조는 가상머신 모니터가 특정 호스트 운영체제 상에서 동작하며, 호스트 운영체제의 도움을 받아 가상화를 구현한다. 이러한 구조에서는 호스트 운영체제의 동작에 가상머신 상의 게스트 운영체제가 큰 영향을 받게 된다.

최근의 Xen 가상머신 모니터[4]는 다음 그림 4와 같은 구조를 가진다. 최근의 프로세서 구조들은 앞서 정의한 가상화 가능한 구조를 갖지 않는다. 즉, x86 계열의 프로세서들이나 ARM 등의 프로세서는 가상화-민감 명령어들을 비특권 모드에서 실행하더라도 트랩을 발생시키지 않는다. 이러한 프로세서 구조에서 가상화를 지원하기 위해서는 전체 명령어-셋-구조를 에뮬레이션 하거나 가상화 계념을 완화하여 가상머신을 제공한다.

Xen 가상머신 모니터는 가상화의 개념을 완화하여 가상화를 구현한다. 즉, 응용 프로그램의 수정을 필요

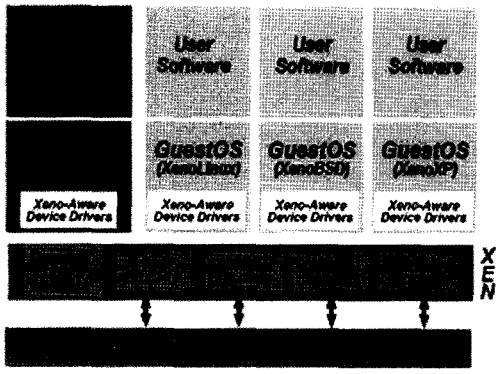

그림 4 Xen 구조

로 하지 않지만, 게스트 운영체제의 일부 코드를 수정하여 사용함으로써 기존의 가상머신 구조들이 가지고 있던 성능상의 문제점을 크게 해소하였다.

Xen 가상머신 모니터는 수백 대의 분리된 네트워크 서버들을 하나의 물리 서버 시스템에서 통합하여 동작시킬 수 있도록 한다. Xen 가상머신 모니터는 기존의 가상머신 구조들과 달리 물리 장치들에 대한 드라이버를 별도의 게스트 운영체제 상에 올려 두어 가상머신 크기를 줄이고, 신뢰성을 확보하였다. 드라이버 도메인이 분리되었기 때문에 일반 사용자 응용 프로그램을 위한 게스트 도메인과 백엔드 도메인 간에 통신을 위한 이벤트 채널과 데이터 전송을 위한 비동기 통신 기법을 도입하였다.

Xen 가상머신 모니터는 게스트 운영체제를 수정해야 하기 때문에, 실제 구현에는 많은 어려움이 따른다. Xen의 논문에서는 3,000~6,000여 라인의 수정을 통해 부정 가상화(para- virtualization)를 구현하였다고 하였으나, 실제 수정되는 코드의 위치가 운영체제의 여러 곳에 흩어져 있으며 하드웨어 추상화 계층(HAL, Hardware Abstract Layer)과 같은 구조가 잘 정의되어 있지 않으면 수정해야 할 부분을 찾기가 쉽지 않다. 이러한 문제는 내장형 시스템과 같이 다양한 하드웨어 구조를 가진 시스템에서 더욱 크다. 이러한 문제점을 해결하기 위하여 Xen의 구조 중에서 가상화에 따라 변경이 필요한 부분만을 독립적으로 개발할 수 있도록 가상화 지원 루틴을 게스트 운영체제 코드에서 분리한 접근 방식이 있다. 선행 가상화(Pre-virtualization) 기법은 Xen과 같은 부정가상화 기법을 사용한 게스트 운영체제의 개발상의 용이성을 제공한다.

시스템 가상화와 마이크로 커널 구조에 관련된 비교 연구[5]에서는 마이크로 커널 기반의 시스템과 가상머신 시스템의 구조적 차이점을 파악하였고 장/단점을 분석하였다. 마이크로 커널은 비교적 작은 크기의 코드만을 특권레벨에서 동작시키고, 나머지 운영체제 요소들은 특권 레벨을 제거한 형태로 사용자 레벨의 서버로 동작시킴으로써, 운영체제 구성 요소들 간의 의존성을 낮춤으로써 유연성을 향상하였으며, 특권 레벨을 가진 코드들의 신뢰성을 향상 시키도록 하였다. 하지만, 마이크로 커널의 사용자 레벨 서버들이 다른 서버나 커널에 대하여 역의존 관계를 가지는 경우가 생길 수 있으며, 이러한 역의존 관계는 오히려 구조적인 불균형을 낳거나 신뢰성을 무너뜨리는 결과를 가져올 수 있다. 또한 사용자 레벨의 서버들간의 통신으로 인한 IPC 성능이 전체 시스템의 성능 저하를 가져올 수 있기 때문에 IPC로 인한 성능 저하를 최소화할 수 있는 구조를 반드시 가져야만 한다.

이와 반대로 가상머신 시스템은 전체 소프트웨어 시스템을 하나의 가상머신 안에서 구현하고 있으므로 자체를 하나의 소프트웨어 컴포넌트화하여 서브 시스템을 구성할 수 있는 형태로 유지하는 것이 중요하다. 두 구조의 차이는 마이크로 커널이 여러 종류의 사용자 레벨 서버들이 모여 하나의 시스템을 구성하는 반면, 가상머신 시스템에서는 각각의 가상머신이 독립적인 시스템을 구성하고 있으며 따라서 의존성이나 IPC 성능과 같은 문제는 없다. 하지만, 가상화에 필요한 성능적인 오버헤드를 최소화하면서도 가상머신의 특징을 살릴 수 있는 기법이 필요하다. 특히 게스트 운영체제나 게스트 운영체제에서 사용되던 응용 프로그램들을 변경 없이 사용할 수 있도록 지원하는 것이 매우 큰 매력이기 때문에, 게스트 운영체제를 변경하더라도 최소한의 범위에서 성능상의 문제를 해결할 수 있는 범위 내에서 수정할 수 있도록 하는 것이 바람직하다.

가상머신의 특성을 활용하여 시스템의 신뢰성을 높이기 위한 여러 가지 노력들이 최근까지 진행되어왔다[6,7]. 최근의 관련 연구[7]에서는 가상머신을 활용하여 버그가 있거나 오동작하고 있는 운영체제의 오류로부터 극복할 수 있는 방법을 제시하고 있다. 기존의 운영체제에서는 장치 드라이버와 같은 서브시스템들이 커다란 운영체제의 일부로서 특권레벨을 가지고 수행되기 때문에 일부 서브시스템의 실패가 전체 운영체제나 시스템 실패로 이어질 수 있다. 가상머신을 이용한 경우 물리머신과 가상머신은 완전히 분리되어 있으며, 모든 물리머신에 대한 직접적인 접근은 가상머신 모니터를 통해서만 이루어지기 때문에, 가상머신 모니터에서 추가적인 보호 메커니즘을 통하여 시스템을 안전한 상태로 복원시킬 수 있다. 이는 일부 서브 시스템의 실패가 전체 시스템의 실패로 이어지는 문제점을 극복하며, 고신뢰성이나 고가용성이 필요한 시스템에 매우 효과적으로 사용이 가능하다. 최근들어 휴대형 내장 시스템에도 개인정보나 금융 정보가 탑재됨에 따라 정보 보안을 위하여 가상머신을 사용하려는 시도가 활발하게 진행되고 있다[8,9].

3. 기존 가상머신들의 한계점

실시간 내장형 시스템에서는 Xen과 같은 기존 가상머신 시스템을 그대로 수용하기 어려운 점이 있다. 첫째, 기존의 실시간 시스템에서 사용하던 실시간 특성

을 제공하기 힘들다. 가상화 계층은 실제 물리머신에서 발생한 이벤트를 모두 가리고 있어, 정확한 물리시간을 측정할 수 없다. 뿐만 아니라 게스트 운영체제는 가상시간을 기반으로 동작하기 때문에, 다른 게스트 운영체제와 같이 동작하기 위해서는 가상 I/O 장치의 동작을 예측하기 힘든 단점이 있다. 둘째, 가상머신 시스템의 구조에 따라 응답시간 등의 요구사항을 만족시키기 힘든 경우가 있다. 특히 Xen 가상머신 모니터의 경우, 드라이버 도메인의 분리와 비동기 통신으로 인해 성능 예측이 매우 힘들고, 응답시간을 보장할 수 있는 기법이 없기 때문에 실시간 요구사항을 만족시킬 수 없다. 셋째, 기존 가상머신들의 경우 대규모 네트워크 서버를 통합할 목적으로 구성되었으나, 내장형 시스템은 규모성(scalability)보다는 서로 다른 이종 시스템들을 동시에 사용할 수 있는 가상머신 간의 이질성(heterogeneity)을 제공하는 것이 중요하다. 하지만, 현재까지 연구된 가상머신 모니터들은 이러한 특성을 충분히 고려하고 있지 않다. 이 섹션에서는 보다 구체적인 실시간 내장형 시스템을 가상화 하기 위한 기존 가상머신들의 한계점에 대해 살펴본다.

3.1 실시간 서비스 측면

전통적인 실시간 시스템에서는 스케줄 가능성 분석[10](schedulability analysis)을 통해 시스템의 예측성(predictability)과 확정성(determinism)을 제공한다. 즉, 새로운 태스크가 시스템에 유입되는 순간 해당 태스크가 자신의 임계 시간 이내에 실행을 완료할 수 있는지 판단한다. 이러한 예측을 기반으로 하여 실시간 태스크들의 수락 제어(admission control)를 수행하며, 이러한 기법은 실시간 시스템 내의 모든 태스크들이 임계 시간 이내에 실행을 완료할 수 있음을 결정적으로 제공한다.

전통적인 실시간 시스템에서 예측성과 확정성을 제공하는 근본적인 시스템 분석 기법은 스케줄 가능성 분석에서 출발한다. 스케줄 가능성 분석은 주어진 스케줄러에 대해서 모든 실시간 태스크들이 자신의 임계시간 이내에 모든 실행을 마칠 수 있는지를 확정적으로 설명한다. 이를 위해서 태스크 모델은 실행 주기를 가진 주기 태스크 모델(*주기, 실행시간, 임계시간*)으로 구성되며, 스케줄러는 선점형으로 구현되어 스케줄러가 원하는 시점에 항상 선점할 수 있어야 한다.

스케줄 가능성 분석은 시스템 내의 모든 태스크들이 *임계시간* 이내에 *실행시간* 이상의 수행 시간을 보장 받도록 한다. 실행 시간의 보장을 위해서는 인터럽트와 같이 예측하기 힘든 태스크 실행모델도 같이 고려해야 한다. 현재까지의 실시간 연구를 통해 살펴본 결과에 따르면[11,12], 인터럽트 자체가 문제되기 보다는 인터럽트 처리기 내에서 실행하는 시간을 짧게 줄이고, 쓰레딩이나 지연 실행(Delayed Procedure Call) 등을 통해 인터럽트 처리로 인한 영향을 최소화 할 수 있다.

가상머신 환경에서의 스케줄 가능성 분석을 위해서는 각 가상머신에게 주어지는 가상시간을 확정적으로 제공해 줄 수 있어야 할 뿐만 아니라, 물리적인 임계시간 조건을 만족할 수 있도록 게스트 도메인의 처리 시간을 제한해 줄 수 있는 특별한 기법이 필요하다. 이를 위해서는 가상머신 모니터가 선점형 스케줄러를 통하여 스케줄링 시점을 결정할 수 있어야 한다.

가상화로 인한 오버헤드 역시 극복해야 할 큰 문제 중의 하나이다. 잦은 트랩은 일반적으로 성능에 부정적인 영향을 끼칠 수 있다. 그 이유는 트랩이 발생하는 경우 실행은 불규칙 점프로 변경되며, 파이프라인 구조에서 스톨을 발생시키기 때문이다. 프로세서의 마이크로-구조에서 뿐만 아니라 I/O 장치 가상화는 더욱 큰 오버헤드를 가진다. I/O 장치 가상화를 위해서 기존의 가상머신 모니터는 가상 I/O 장치를 에뮬레이션 한다. 즉, 물리머신의 I/O 장치에서 인터럽트가 발생하거나 접근이 필요한 경우, 가상머신 모니터의 에뮬레이션을 거쳐야만 처리가 가능하기 때문에, 가상머신 모니터에서 실행되는 시간에 대한 이해가 없이는 실시간성의 보장을 제공할 수 없다.

현재까지 제시된 가상머신 모니터들에서는 이러한 요구사항을 충분히 반영하지 못할 뿐만 아니라, 가상화로 인한 오버헤드를 정량적으로 제한할 수 있는 방법 또한 아직까지 제시되지 못하고 있다. 따라서 정확한 시스템 분석과 결정적 실행을 요구하는 실시간 시스템이 가상머신 구조에서 동작하는 데는 큰 무리가 있다.

3.2 Xen의 구조적 측면

Xen 가상머신 모니터는 기존의 가상머신 모니터와 다른 특징을 가지고 있다. 기존의 가상머신 모니터가 가상 장치에 대한 에뮬레이션을 가상머신 모니터에서 직접 하는 반면, Xen 가상머신 모니터는 어떠한 가상 장치에 대한 추상화 개념도 제공하지 않는다. Xen 가상머신 모니터는 드라이버 도메인을 분리하여 제공함으로써 장치 드라이버를 가상머신 모니터에서 완전히 분리하였다. 드라이버 도메인은 가상 장치에 대한 추상화 인터페이스를 제공하고, 가상머신 모니터

는 두 도메인 간의 통신을 제공한다. 분리 드라이버 모델(Split-driver model)은 가상머신 모니터를 장치에 대해 비의존적으로 변경하였다. 즉, 가상머신 모니터가 제공하는 가상 장치에 대한 드라이버들은 드라이버 도메인에 자유롭게 탑재가 가능하기 때문에, 기존 운영체제에서 사용하던 드라이버 모델이나 장치 의존적인 동작을 매우 쉽게 구현할 수 있다.

하지만, Xen 가상머신 모니터는 분리 드라이버 모델을 사용함으로 인해서 성능상의 문제점을 안고 있다. 드라이버 도메인이 물리장치에 대한 드라이버를 가지고 있으며, 가상 장치의 에뮬레이션을 동시에 수행한다. 이러한 분리 드라이버 모델에서 동기적I/O 기법을 사용하는 경우, 성능 상에 큰 문제가 발생하게 된다. 즉, I/O를 요청한 가상머신이 동기적으로 I/O를 처리하는 경우, I/O가 완료될 때까지 처리를 멈추고 대기하여야 한다.

Xen 가상머신 모니터는 가상 장치의 동기적 처리로 인한 처리 시간 지연을 줄이기 위해 도메인 간 비동기 통신 기법을 사용한다. 데이터 전달을 위한 비동기 통신은 링버퍼를 통해 이루어진다. 즉, 데이터 전달을 위한 요청과 응답을 링버퍼를 통해 교환하고, 실제 데이터의 교환은 공유 메모리를 통해 이루어진다. 이때, 요청과 응답이 어떤 도메인에 쌓여있는지를 알려주기 위한 방법으로 이벤트 채널을 둔다. 이벤트 채널은 요청이 발생하면 비록 자신이 실행되고 있지 않더라도 이벤트 채널을 통해 알 수 있으며, 요청을 처리한 후 요청했던 게스트 도메인으로 이벤트를 통해 응답을 전송함으로써 해당 게스트가 실행될 때 바로 데이터를 받아 사용할 수 있도록 한다. 또한 요청과 응답은 한꺼번에 묶어서 처리가 가능하여, 처리량을 높이도록 하였다. 그 결과 Xen 가상머신 모니터의 네트워크 처리 성능은 처리량을 기준으로 기존 일반 리눅스 시스템 대비 3% 정도의 오버헤드를 가짐을 보였다. Xen의 도메인 간 비동기 통신 기법은 처리량을 높이는데 큰 기여를 한다. 특히 수백 대의 네트워크 서버 통합을 구현하는 경우, 이와 같은 네트워크 성능과 처리 지연 시간의 개선은 매우 중요한 의미를 가진다. 하지만, 현재 Xen의 구조는 실시간 처리 측면에서 부족한 점이 있다.

첫째, 드라이버 도메인 역시 Xen 가상머신 모니터의 스케줄을 받아 실행되므로, 드라이버 도메인의 스케줄이 빨리 되어야 I/O 장치의 높은 성능을 기대할 수 있다. 드라이버 도메인은 모든 가상장치와 물리장치의 접근을 제어하므로, 매우 높은 점유율을 할당받아야 한다[13]. 동시에 실시간 태스크들이 실행되

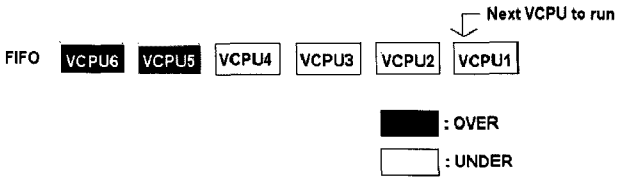

그림 5 크레딧 기반 스케줄러의 동작

는 게스트 도메인은 낮은 응답시간과 일정 시간 이상의 수행 시간을 보장받아야 하기 때문에 높은 우선순위를 가져야 한다.

Xen의 기본 스케줄러인 크레딧 기반 스케줄러는 VCPU를 기준으로 스케줄링하는 방식을 사용하며, 각 VCPU들의 상태는 그림 5와 같이 주어진 할당량과 사용한 시간에 따라 UNDER, OVER 두 가지 상태를 가진다. UNDER 상태의 VCPU는 스케줄 대상이며, 할당량을 모두 사용하면, OVER 상태로 변경된다. 크레딧 기반 스케줄러는 모든 UNDER 상태의 VCPU들이 없어질 때까지 UNDER 상태의 VCPU들을 스케줄링한다. 즉 드라이버 도메인이 자신에게 할당된 크레딧을 모두 사용하고 나면 다른 게스트 도메인들이 모두 자신의 할당량을 소진할 때까지 기다려야만 한다.

이러한 단점을 극복하기 위해서 최근의 크레딧 스케줄러[14]는 I/O가 요청된 게스트 도메인의 우선순위를 일시적으로 올려주는 기법을 도입하였다. 즉 VCPU의 상태에 UNDER, OVER 이외에 BOOST를 추가로 두어 I/O 처리를 위한 게스트 도메인의 처리 시간을 확보해 주었다. 이 경우 드라이버 도메인의 할당량이 고갈되어 I/O 처리 지연이 생기는 문제는 해결할 수 있으나, 모든 도메인들이 I/O 처리를 요청한 경우 모든 도메인이 BOOST 되는 문제가 다시 발생할 수 있는 단점이 있다.

I/O 처리를 요청한 도메인의 처리 시간을 제한할 수 없는 단점이 생기며 이러한 문제는 높은 실시간성을 요구하는 응용에게는 치명적인 문제를 가져온다. 또한, 처리량을 높이기 위해 사용되는 Xen 가상머신 모니터의 비동기 통신 기법은 응답 시간과 처리량 측면에서는 큰 장점이 있으나, 성능의 예측 가능성과 분석의 측면에서 근본적으로 한계가 있다. 다른 실시간 게스트 도메인에 주는 영향을 예측하거나 한계치를 분석할 수 없기 때문에, 실시간 게스트 도메인을 바로 사용하기에는 무리가 따른다.

Xen 가상머신 모니터는 거의 비슷한 구조와 기능을 가진 네트워크 서버의 통합을 목적으로 설계되었기 때문에, 다양한 하드웨어 조합을 가진 내장형 시스템에서 효율적으로 사용되기 힘들다. 서로 다른 가상머신이라고 하더라도 네트워크 서버들은 거의 비슷한

설정과 하드웨어를 가정하고 있다. 이에 반해, 내장형 시스템에서 사용되는 하드웨어들은 서로 제각기 다른 설정과 하드웨어 구조를 가지고 있다. 시스템을 구성하는 컴포넌트의 하나로 각 가상머신을 생각해 볼 때, 각 가상머신들은 동일한 역할이나 기능을 제공할 필요가 없으며, 오히려 서로 다른 가상머신들이 독립적인 기능을 제공하는 구성이 바람직하다. 즉, 서로 다른 가상머신에 대해 각기 다른 하드웨어/소프트웨어 구성을 가지는 형태로 구성이 가능해야 한다. 현재 Xen 가상머신 모니터에서 제공하는 가상머신의 설정(configuration)은 매우 단순한 형태이며, 가상 장치들의 모델도 매우 범용적인 설정만을 제공한다. 서로 다른 가상머신을 정의하기 위해서는 하드웨어 추상화 계층에 대한 명확한 정의가 필요하며, Xen에서는 이러한 설정을 손쉽게 변경하기 힘들다.

4. 실시간 내장형 시스템 가상화를 위한 요구사항

위에서 언급한 바와 같이 현재 Xen 가상머신 모니터는 실시간 내장형 시스템의 가상화를 위해 사용하기에 부족한 면들이 있다. 이번 절에서는 이러한 문제를 해결하기 위해 필요한 연구들의 방향과 몇가지 해결책들을 제시한다.

4.1 가상머신 구조에서 실시간성 지원을 위한 연구

가상머신 시스템에서 실시간성을 지원하기 위해서는 범용 운영체제가 실시간 지원을 위해 노력했던 연구들을 검토해 볼 필요가 있다. 범용 시스템에서는 상위의 다양한 응용 프로그램들을 지원하면서도 실시간 태스크들에 일정량 이상의 처리 시간을 제공해야 하며, 이를 위해서 다음과 같은 몇 가지의 기법들이 제시되었다.

첫째, 실시간 스케줄 가능성 분석을 위한 기법이 필요하다. 실시간 시스템에서 예측 가능성과 결정성을 제공하기 위해서 필요한 분석 기법인 스케줄 가능성 분석은 현재까지 가상머신 시스템에서 이루어지지 않고 있다. 기존의 계층형 스케줄러 등과 같은 연구를 통해 각 서브시스템들의 자원 할당과 예약을 통하여 결정성을 제공하고, 이를 통해 실시간 게스트 도메인의 시간 제약 조건을 만족할 수 있음을 보여야 한다.

특히 가상 I/O 장치를 사용하는 태스크들에 대한 실시간성 보장을 어떻게 제공할 것인지에 대한 논의가 추가적으로 필요하다. 왜냐하면, 가상 I/O 장치들의 경우 동작과 처리 시간에 대한 분석이 가상머신 상에서 이루어질 수 없으며, 가상머신 모니터 내 혹은 드라이버 도메인에서 이루어지기 때문이다.

둘째, 불가피하게 발생하는 예측 불가능한 이벤트들(인터럽트 등)에 대한 처리를 어떻게 결정적으로 제공할 수 있을 것인지에 대한 연구가 필요하다. 특히 I/O 모델과 관련하여 실시간을 고려하지 않은 범용 시스템의 I/O 모델과 실시간 시스템의 I/O 모델을 어떻게 동시에 사용할 수 있을 것인지에 대한 연구가 필요하다.

동시에 기존의 운영체제에서 사용되던 드라이버 모델을 어떻게 하면 재활용하면서도 가상머신 시스템의 I/O 모델에 맞게 사용할 수 있는지에 대한 연구도 필요하다. 이러한 연구는 장치 드라이버의 신뢰성과 나아가 가상머신 모니터 자체의 신뢰성과도 직결되는 문제가 될 수 있다.

즉, 실시간 스케줄러와 관련하여 1) 스케줄 가능성 분석을 위한 연구, 2) 가상머신 환경에서 결정적인 I/O 처리 모델 등에 대한 연구가 필요하다.

4.2 가상머신의 인터페이스 정의

내장형 시스템의 특성 상, 다양한 하드웨어 구성과 구조는 표준화된 개인용 PC 기반의 가상머신들과 큰 차이를 가진다. 하드웨어 추상화 계층을 운영체제가 가지고 있는 경우, 가상머신과 게스트 운영체제의 하드웨어 추상화 계층 간의 인터페이스를 정해둠으로써 이러한 문제는 쉽게 완화될 수 있다.

하드웨어 추상화 계층을 가지고 있지 않은 운영체제들에 대해서도 가상머신의 인터페이스를 통해 접근하기 위해서, 범용적이면서도 하드웨어 구조에 따라 유연하게 정의 가능한 형태의 가상머신 인터페이스를 정의해야 한다.

현재까지 가상머신 인터페이스는 x86 기반의 개인용 PC 구조를 바탕으로 정의되어 있다. 이는 Toyomo Linux[15]와 VMWare[16]를 통해 공개되어 있다. 하지만, 다양한 종류의 임베디드 프로세서 구조와 하드웨어 특성을 반영하기 위해서 I/O 모델을 포함하는 가상머신 인터페이스에 대한 정의가 필요하다.

특히 부정 가상화와 같이 게스트 운영체제의 수정이 필요한 접근 방식들에서는 이러한 정의를 통해 부정 가상화의 어려운 점들이 많이 해소될 수 있을 것으로 예상된다.

4.3 기존 Xen 가상머신 모니터 시스템의 실시간 지원 필요

기존 Xen 가상머신 모니터는 가상화의 성능에 대한 우려를 부정 가상화를 통해 해결하였다. 실시간 내장형 시스템에서는 단순히 높은 성능에 대한 결과보다

는 예측 가능성과 결정성에 대한 보다 뚜렷한 해결책을 제시해 주어야 한다. 즉, Xen 가상머신 모니터에서 실제 실시간 운영체제를 사용한 경우의 성능 평가를 기반으로 하여 응용 프로그램들에게 얼마나 높은 예측 가능성을 제시할 수 있는가에 대한 대답을 제시할 수 있어야 한다. 이를 위해서는 앞서 언급한 가상 장치의 I/O 모델이나 스케줄 가능성 검사를 통한 가상머신 생성/스케줄 등에 필요한 서비스 수준 관리(Quality of Service)에 대한 연구가 추가적으로 필요하다.

5. 결 론

본 논문에서는 내장형 시스템 가상화 연구의 동향과 함께 Xen 가상머신 모니터 등의 현재 제시된 가상머신 시스템이 실시간 내장형 시스템에서 사용되기 위해서 어떠한 추가 연구가 필요한지에 대하여 살펴보았다. 가상머신 시스템에서는 물리머신의 이벤트가 모두 가상화되어 게스트 운영체제에 전달되므로 실시간 응용이 직접적으로 사용되기 힘들다. 특히 시스템의 예측 가능성과 결정성을 제공하는 스케줄 가능성 분석이 불가능하다. 따라서 실시간 운영체제를 가상머신 구조에서 사용하기 위해서는 스케줄 가능성 분석을 위한 기법과 이를 지원할 수 있는 적절한 I/O 모델에 대한 연구가 필요하다. 또한, 다양한 하드웨어 구성을 가지는 임베디드 시스템의 특성을 고려하여 임베디드 시스템에서도 사용할 수 있는 범용적인 가상머신 인터페이스에 대한 연구가 필요하다.

참고문헌

[1] J. Smith and R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes(The Morgan Kaufmann Series in Computer Architecture and Design). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[2] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," Commun. ACM, vol. 17, no. 7, pp. 412-421, 1974.

[3] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in Proceedings of the General Track: 2002 USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, pp. 1-14, 2001.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in SOSP'03: Proceedings of the nineteenth ACM symposium on Operating systems principles. New York, NY, USA: ACM, pp. 164-177, 2003.

[5] G. Heiser, V. Uhlig, and J. LeVasseur, "Are virtual-machine monitors microkernels done right?" SIGOPS Oper. Syst. Rev., Vol. 40, No. 1, pp. 95-99, 2006.

[6] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz, "Unmodified device driver reuse and improved system dependability via virtual machines," in OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. Berkeley, CA, USA: USENIX Association, pp. 2-2, 2004.

[7] F. M. David, J. C. Carlyle, E. M. Chan, P. A. Reames, and R. H. Campbell, "Improving dependability by revisiting operating system design," in HotDep'07: Proceedings of the 3rd workshop on Hot Topics in System Dependability. Berkeley, CA, USA: USENIX Association, p. 1, 2007.

[8] C. J. Fidge, "Real-time schedulability tests for preemptive multitasking," in WPDRTS: Selected papers from the 4th workshop on Parallel and distributed real-time systems. Norwell, MA, USA: Kluwer Academic Publishers, pp. 61-93, 1998.

[9] L. P. Cox and P. M. Chen, "Pocket hypervisors: Opportunities and challenges," in HOTMOBILE'07: Proceedings of the Eighth IEEE Workshop on Mobile Computing Systems and Applications. Washington, DC, USA: IEEE Computer Society, pp. 46-50, 2007.

[10] C. J. Fidge, "Real-time schedulability tests for preemptive multitasking," in WPDRTS: Selected papers from the 4th workshop on Parallel and distributed real-time systems. Norwell, MA, USA: Kluwer Academic Publishers, pp. 61-93, 1998.

[11] M. B. Jones and J. Regehr, "The problems you're having may not be the problems you think you're having: Results from a latency study of windows nt," in HOTOS'99: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems. Washington, DC, USA: IEEE Computer Society, p. 96, 1999.

[12] M. B. Jones and S. Saroiu, "Predictability requirements of a soft modem," SIGMETRICS Perform. Eval. Rev., Bol. 29, No. 1, pp. 37-49, 2001.

[13] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," SIGMETRICS Perform. Eval. Rev., Vol. 35, No. 2, pp. 42-51, 2007.

[14] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling i/o in virtual machine monitors," in VEE'08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. New York, NY, USA: ACM, pp. 1-10, 2008.

[15] Toyomo Linux paravirtualization Interface http://tomoyo.sourceforge.jp/cgi-bin/lxr/source/include/asm-x86/paravirt.h

[16] VMware, Inc. vmi_spec.txt: Paravirtualization API Version 2.5. http://www.vmware.com/pdf/vmi_specs.pdf, February 2006.

유 시 환

2002 고려대 컴퓨터학과 학사
2004 고려대 컴퓨터학과 석사
현재 고려대학교 컴퓨터학과 박사과정 재학 중
관심분야 : Real-time OS, 내장형 시스템, 시스템 가상화
E-mail : shyoo@os.korea.ackr

유 혁

1982 서울대 전자공학과 학사
1984 서울대 전자공학과 석사
1986 University of Michigan 컴퓨터 공학 석사
1990 University of Michigan 컴퓨터 공학 박사
현재 고려대학교 컴퓨터학과 교수
관심분야 : Real-time OS. 임베디드 소프트웨어, 네트워크, 미디어 처리
E-mail : hxy@os.korea.ac.kr