임베디드 멀티코어 플랫폼을 위한 운영체제 연구

홍철호O 유혁
고려대학교 컴퓨터 전파통신공학과
{chhongO, hxy}@os.korea.ac.kr

Operating Systems Research for the Embedded Multi-core Platforms

Cheolho HongO, Chuck Yoo
Department of Computer Science and Engineering, Korea University

요 약

최근 무어의 법칙이 깨짐에 따라 멀티코어 프로세서의 활용이 늘어나고 있으며 이는 임베디드 환경에서도 보편화 되었다. 이러한 멀티코어 환경에 기존에 멀티프로세서 용으로 개발된 AMP 또는 SMP 구조의 운영체제를 적용시키게 된다면 멀티코어의 장점을 살리기 어렵다. 본 논문에서는 기존 운영체제 구조에 대한 분석을 통해 멀티코어용으로 적합한 운영체제 구조가 가상 머신 구조라는 것을 보이고 있으며 산업에 활용할 수 있는 멀티코어용 가상 머신 모니터의 설계를 제공하고 있다.

1. 서 론

무어의 법칙은 반도체의 집적 밀도가 18~24개월마다 2배로 증가한다는 것이다. 그러나 집적 밀도가 높아지고 트랜지스터의 크기가 점점 작아짐에 따라 전력 소모 및 열 발산 문제가 발생하였고 더 이상 트랜지스터의 크기를 줄이는 일이 불가능하게 되었다. 이렇게 무어의 법칙이 깨짐에 따라 복수 개의 코어를 하나의 칩 안에 집적하여 병렬적으로 성능을 높이는 멀티코어 기술이 최근 각광받고 있다.

인텔과 AMD사의 멀티코어가 데스크탑 환경에서 보편화되었고 인텔은 현재 80개의 코어를 내장한 CPU를 시제품으로 만들고 테스트를 진행하고 있는 상태이다. 또한 ARM과 IBM 같은 전통적인 프로세서 생산 업체에서도 멀티코어 프로세서를 제작하고 있다. 이러한 멀티코어 개발 동향은 최근 임베디드 환경에서도 보편화 되었다.

멀티코어 프로세서는 크게 두 가지 유형으로 나누어질 수 있다. 하나는 같은 유형의 코어를 하나의 칩 안에 패키징하여 기존의 멀티 프로세서와 같은 효과를 내는 동종 멀티코어 프로세서이다. 다른 하나는 이종의 서로 다른 코어를 하나의 칩 안에 내장하여 여러 코어가 갖는 특성을 극대화하는 이종 멀티코어 프로세서이다. 대표적인 동종 멀티 코어 프로세서의 예로는 인텔의 코어2듀오, ARM의 MPCore 프로세서가 있고 이종 멀티 코어 기술로는 IBM, Toshiba, Sony연합의 Cell 프로세서 등이 있다[4, 5].

운영체제의 관점에서 보면 멀티코어를 사용하는 플랫폼에 기존의 운영체제 설계 방식을 그대로 사용할 수 있다. 기존의 운영체제 방식이란 멀티 프로세서 환경을 위해 개발된 운영체제로서 AMP(asymmetric multiprocessing) 구조 또는 SMP(symmetric multiprocessing) 구조에 기반하고 있다.

그러나 기존의 운영체제 방식을 이종 혹은 동종 코어 플랫폼에 그대로 적용시키는 것은 몇 가지 문제점을 발생시킨다. 예를 들어 AMP 시스템의 경우 공유 자원을 효율적으로 관리하는 동기화 비용이 추가되어야 하며 SMP 시스템의 경우 두 개 이상의 운영체제를 지원할 수 없으며, 병렬성을 고려하여 프로그래밍 해야 하고 이종의 멀티코어 환경을 바로 지원할 수 없다는 문제가 있다.

이러한 AMP 및 SMP 구조의 문제점들은 가상 머신 구조를 사용하면 대부분 해결 될 수 있다. 가상 머신 구조의 경우 2개 이상의 운영체제를 지원하며 공유 자원을 중앙 집중적으로 관리할 수 있는 장점이 있다.

본 논문에서는 임베디드 멀티코어를 위한 기존의 운영체제 구조를 살펴보고 각각의 장단점에 대해 논의한다. 장단점의 분석 결과로 임베디드 멀티코어 환경에서 가상 머신 구조를 사용해야 하는 이유에 대해 알아본다. 또한 실제 멀티 코어 구조에 맞는 가상 머신 모니터의 설계를 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 AMP시스템과 SMP 시스템, 가상 머신 구조에 대해 자세히 알아본다. 3장에서는 가상 머신 구조를 사용했을 때의 장점을 살펴보고 4장에서 가상 머신 구조의 설계를 제공한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 AMP 시스템

AMP방식은 멀티코어 기반 프로세서에서 각각의 코어 위에 서로 다른 운영체제의 이미지를 올려 여러 개의 운영체제를 동시에 동작시키는 방식을 말한다. 하나의 코어에 하나의 운영체제가 운영되므로 기존의 단일 프로세서 방식의 구조와 유사하다. AMP방식을 사용할 경우 단일 프로세서로 가정하고 프로그래밍을 하면 되기 때문에 애플리케이션의 디자인 및 개발이 쉽고 기존에 사용하던 프로그램의 포팅이 쉬어진다. 반면 한 코어에 작업이 집중 될 경우 다른 코어로 일을 분배하기가 어려우며 공유 자원이 많을 경우 공유 자원의 분배를 위해 동기화 메커니즘이 철저하게 사용되어야 한다[8].

이러한 AMP를 지원하는 대표적인 솔루션으로 Enea 사의 AMP 솔루션이 있다[6]. 이 솔루션에서 사용하는 운영체제는 Enea사가 개발한 RTOS인 Enea OSE와 리눅스이다. OSE는 32비트와 64비트를 위한 실시간 운영체제이며 DSP에서 동작하도록 컴팩트 버전인 OSEck도 개발되어 있다. 리눅스는 MontaVista Linux, Red Hat, SUSE 등의 버전을 지원한다.

Enea 사의 OSE 솔루션 같은 AMP 시스템에서는 통신(IPC) 프로토콜이 중요하게 여겨진다. 각 코어마다 올라가 있는 운영체제가 통신을 통해 서로 정보 공유를 하고 제어를 하기 때문이다.

2.2 SMP 시스템

SMP 구조는 하나의 운영체제가 시스템에 설치된 모든 프로세서 코어들을 관리하는 모델이다. 여기에는 여러 개의 멀티 코어들을 스케줄링 할 수 있는 운영체제가 동작하게 된다. 스케줄링은 프로세스 혹은 스레드 단위로 이루어지며, 여러 코어가 동적으로 할당되어 시스템의 전체 부하를 균등하게 조절할 수 있다. 윈도우즈, 솔라리스, 리눅스 등의 현대적인 운영체제들은 대부분 SMP 모델을 고려하여 설계되고 구현되었다. 하지만 SMP 시스템은 두 개 이상의 운영체제를 지원하지 못하는 단점이 있고 병렬성을 고려하여 프로그래밍하지 않을 경우 성능을 높일 수 없다.

2.3 가상 머신 구조

가상 머신 구조는 운영체제 아래에 가상화 레이어를 추가한 기법을 가리킨다. 가상화 레이어 위에는 복수 개의 운영체제가 돌아갈 수 있다. 가상화 레이어는 각각의 운영체제가 하드웨어를 독점하고 있다는 환상을 주는 역할을 맡게 되며 이 레이어의 구현물을 가상 머신 모니터 또는 VMM(virtual machine monitor)라고 부르게 된다. 가상 머신 모니터는 각각의 운영체제가 돌아갈 수 있는 가상 머신을 제공하게 된다.

가상 머신의 구현 방법은 완전 가상화와 부분 가상화로 나뉘게 된다. 완전 가상화는 가상 머신 모니터가 모든 하드웨어의 에뮬레이션을 제공하며 운영체제를 수정할 필요가 없는 방식이다. 부분 가상화는 모든 하드웨어의 가상화를 지원하지 않으며 필요한 경우 게스트 운영체제를 수정할 수 있다. 완전 가상화의 대표적인 예로 VMWare[7]가 있으며 부분 가상화의 예로 Xen[1, 2]이 있다. Xen위에 올라가는 부분 수정된 운영체제를 게스트 운영체제라 부른다.

3. 가상 머신 구조 사용시 장점

관련 연구에서 살펴보았듯이 AMP와 SMP시스템을 임베디드 멀티코어 플랫폼에 바로 적용시키는 것은 문제가 있다. AMP 시스템의 경우 동기화 비용이 추가되어야 하며 SMP 시스템의 경우 두 개 이상의 운영체제를 지원할 수 없으며, 이종의 멀티코어 환경을 바로 지원할 수 없다는 문제점이 있다.
한편 임베디드 멀티코어 환경에서 가상 머신 구조를 사용하면 다음과 같은 장점이 있다[3].

3.1 AMP 시스템 사용시 발생하는 IPC 오버헤드 감소

AMP 시스템에서는 각각의 운영체제가 통신(IPC) 프로토콜을 통해 서로 정보 공유를 하고 공유 자원에 대한 제어를 하게 된다. 가상 머신 구조에서는 공유 자원의 관리를 운영체제가 아닌 가상 머신 모니터가 담당하게 된다. 가상 머신 모니터 내부에 운영체제당 할당된 자원의 내역에 관한 테이블이나 버퍼를 두고 관리하며 게스트 운영체제는 AMP 시스템처럼 다른 운영체제가 어떤 장치나 메모리 영역을 사용하는지 관리하지 않아도 된다. 그러므로 AMP 시스템처럼 불필요한 IPC 오버헤드가 발생하지 않게 된다. 또한 가상 머신 모니터와 게스트 운영체제는 같은 주소 공간을 나누어서 사용하게 된다. 따라서 게스트 운영체제가 가상 머신 모니터에 자원의 할당 및 해제를 요청할 때는 주소 공간이 바뀌지 않은 상태에서 요청이 수행되므로 AMP 시스템보다는 적은 통신 오버헤드가 발생하게 된다.

3.2 안정성

가상 머신 환경하에서는 각각의 운영체제가 비특권 레벨에서 돌아가고 가상 머신 모니터만 특권레벨에서 돌아가게 된다. 이때 하나의 운영체제에서 문제를 일으키면 프로세서에서 트랩이 발생하고 그 트랩에 대한 처리를 특권레벨에서 돌아가고 있는 가상 머신 모니터가 처리할 수 있게 된다. 따라서 가상 머신 모니터에서 문제를 일으킨 운영체제의 자원을 회수한 후 재 시작하면 되므로 다른 운영체제 또한 영향을 받지 않고 실행을 계속 할 수 있다.

3.3 보조코어의 동기화 및 활용성

AMP 시스템의 경우 DSP, GPU 등 보조코어를 공유 자원으로 여기기 때문에 이에 대한 동기화 문제 및 로드 밸런싱의 문제가 발생할 수 있다. 그러나 가상 머신 구조에서는 각각의 운영체제가 보조 코어를 쓰고 싶다는 메시지를 인터페이스를 통해 가상 머신 모니터에게 전달하기만 하면 되며 그러한 요청을 받아서 가상 머신 모니터가 보조 코어에 대한 할당 및 반환을 진행하므로 동기화에 대한 문제를 해결할 수 있다.

또한 가상 머신 모니터에서 보조 코어를 강제적으로 할당하고 반환을 하게 된다면 한 운영체제가 보조 코어를 할당해서 놓지 않는 경우를 막고 보조코어를 필요한 운영체제에 바로 할당할 수 있어 활용도를 높일 수 있다. 즉 AMP 구조에서 비 선점형(non-preemptive)으로 보조 코어를 사용했다면 가상 머신 구조에서는 선점형(preemptive)으로 보조 코어를 사용하게 되는 것이다. 그리고 일반적인 프로세스 스케줄링 정책과 별도로 보조 코어를 위한 스케줄링 정책을 적용할 수 있게 되어 각 태스크의 응답성을 높일 수 있고 실시간 작업에 대비할 수 있다.

3.4 전원 관리(power management)

가상 머신 구조에서는 코어 및 하드웨어 자원이 한 운영체제에 정적으로 할당되는 것이 아니라 가상 머신 모니터에 의해 동적으로 할당되게 된다. 따라서 모든 하드웨어의 요청(I/O request) 및 요청에 대한 결과(I/O response)는 가상 머신 모니터를 통하게 되어 있으므로 가상 머신 모니터는 현재 어떤 코어가 실행되고 있지 않은 지와 어느 디바이스가 사용되고 있지 않은 지를 알 수 있다. 따라서 가상 머신 모니터에서 현재 사용하지 않는 코어나 디바이스의 전원을 오프시킴으로써 전원을 절약할 수 있다. 이러한 구조에서는 가상 머신 모니터에 의해 모든 장치의 온/오프가 제어되므로 최적의 절전 성능을 기대할 수 있다.

4. 멀티코어를 위한 가상 머신 설계

본 연구에서는 임베디드용 이종/동종 멀티 코어 플랫폼을 위한 가상 머신 모니터를 설계하기 위해 아래와 같은 목표를 설정하였다.

1. 실시간 운영체제를 포함한 복수개의 운영체제 환경을 지원함
2. 복수개의 운영체제가 수행되는 경우 운영체제 간의 고립화를 제공함
3. 보조 코어의 활용도를 높이며 확고한 관리기법을 제공함
4. 메모리 자원 등 공유 자원의 정적인 할당 및 해제를 제공함

위와 같은 목표를 세우고 코어 및 메모리, 보조 코어 서브 시스템에 대한 구체적인 설계를 다음과 같이 진행하였다.

4.1 CPU 가상화

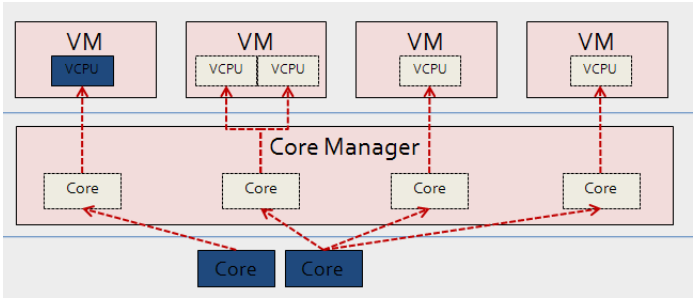

그림 1 코어 가상화

실제의 물리 코어를 코어 매니저에서 가상 코어로 추상화를 하며 추상화된 가상 코어를 게스트 운영체제에 필요한 만큼 나누어주게 된다. 이런 구조를 가짐으로써 CPU의 사용량이 많은 게스트 운영체제에 더 많은 가상 CPU를 할당함으로써 작업을 분배할 수 있는 로드 밸런싱의 기능을 구현할 수 있다. 또한 실시간 게스트 운영체제가 올라갈 경우 물리적인 CPU를 하나 이상 정적으로 할당하여 실시간 작업을 처리하게 하고 있다.

4.2 메모리 가상화

메모리 가상화는 Xen과 같이 게스트 운영체제가 갖고 있는 페이지 테이블을 하드웨어의 MMU에 직접 매핑하여 성능을 높이도록 한다. 게스트 운영체제의 페이지 테이블에 쓰기 방지 옵션을 활성화시켜 게스트 운영체제가 페이지 테이블을 수정할 때마다 다른 게스트 운영체제의 페이지를 할당하는지 가상 머신 모니터에 의해 검증을 받게 한다.

그림 2 정적인 메모리 할당 기법

한편 물리 메모리 할당 정책은 정적인 할당 정책을 택한다. Xen과 같은 서버 가상화용 가상 머신 구조에서는 활성화된 운영체제의 목표를 100개 정도로 잡기 때문에 동적으로 물리 메모리를 할당하고 해제하는 알고리즘이 필요하지만 임베디드 환경에서는 운영체제가 많아야 세 개 정도 올라가게 된다. 이러한 환경에서는 물리 메모리를 정적으로 할당하는 것이 비용을 줄일 수 있다. 그림2는 세 개의 운영체제가 물리 메모리를 정적으로 할당하여 사용하고 있는 모습을 그리고 있다.

4.3 보조 코어 가상화

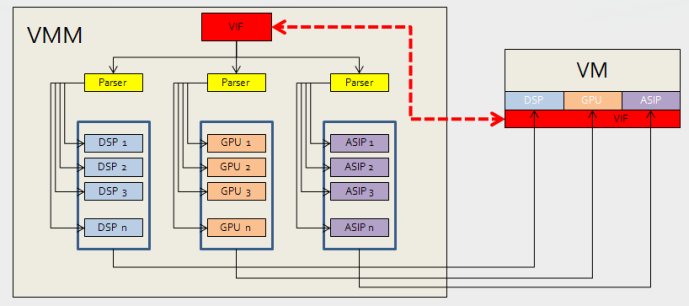

그림 3 보조 코어 인터페이스

각 게스트 운영체제는 보조 코어를 디바이스처럼 생각하고 가상 머신 모니터에 I/O작업을 요청하게 된다. 이때 가상 머신 모니터는 각각의 게스트 운영체제로부터 들어온 I/O작업을 우선 순위별로 스케줄링 하여 실제 물리적인 보조 코어에 작업을 할당하게 된다. 게스트 운영체제는 비슷한 특성의 서로 다른 DSP, GPU, ASIP 등이 있는 상황에서 공통적인 인터페이스를 사용하여 VMM에 요청을 하게 되며 VMM에서는 이를 파싱하여 특정한 물리 보조 코어에 작업을 할당하게 된다.

5. 결론

본 논문에서는 임베디드 멀티코어 환경에서의 운영체제를 연구하였고, AMP, SMP 그리고 가상 머신 구조 중에 가상 머신 구조가 임베디드 멀티코어 환경에 가장 적합하다고 결론을 내렸다. 가상 머신 구조를 사용하면 복수개의 운영체제를 지원할 수 있고 안정성을 얻을 수 있다. 또한 보조코어의 동기화를 지원하며 보조코어의 활용성을 높일 수 있다. 그리고 전원 관리를 효율적으로 지원하여 최대의 절전 성능을 얻을 수 있다.

이러한 가상 머신 구조의 장점을 바탕으로 본 논문에서는 실제 산업 현장으로의 적용을 위해 멀티코어, 메모리, 디바이스에 대한 가상화 기법을 제안하였다.

참고문헌

[1] B. Dragovic et al. Xen and the Art of Virtualization. In Proc. of SOSP, 2003.
[2] I. Pratt et al. Xen 3.0 and the Art of Virtualization. In Proc. of the Ottawa Linux Symposium, 2005.
[3] Greg Diamos, Ada Gavrilovska, Vishakha Gupta, Sanjay Kumar, Himanshu Raj, Karsten Schwan, Sudhakar Yalamanchili, "Virtualizing Heterogeneous Many-core Platforms", EuroSys 2007 Poster, Lisbon, Portugal, Mar. 2007.
[4] J. Kahle et al., "Introduction to the Cell Multiprocessor," IBM J. Research and Development, Sept. 2005, pp. 589-604.
[5] M. Gschwind et al., "Synergistic Processing in Cell's Multicore Architecture," IEEE Micro, Mar./Apr. 2006, pp. 10-24.
[6] ENEA AMP Solution - http://www.multicore-expo.com
[7] VMWare Solution - http://www.vmware.com/
[8] MultiCore Expo 2007