韓國 컴퓨터情報學會 論文誌
第14卷 第1號, 2009. 1.

2009-14-1-1-7

임베디드 시스템을 위한 가상 머신 모니터의 설계와 구현

손 성 훈*, 이 재 현**

Design and Implementation of Virtual Machine Monitor for Embedded Systems

Sunghoon Son*, Jaehyeon Lee**

요 약

가상화는 최근 활발한 연구가 진행되고 있는 시스템 소프트웨어 분야 중 하나이다. 범용 컴퓨터 시스템을 위한 가상화 분야에는 이미 상용화 제품들도 다수 존재하는 반면, 임베디드 시스템의 특성을 고려한 가상화에 대한 연구는 상대적으로 미미한 실정이다. 본 논문에서는 임베디드 시스템의 각 하드웨어 자원을 여러 논리적인 하드웨어 자원들로 나누고, 이를 다수의 가상 머신들로 재구성하는 가상 머신 모니터를 설계하고 구현하였다. 제안된 가상 머신 모니터는 하나의 임베디드 시스템 상에 다수의 실시간 운영체제들을 동시 수행하는 것을 가능하게 한다. 실제의 임베디드 시스템 하드웨어 상에서 구현된 가상 머신 모니터에 대해 다양한 성능 측정 실험을 수행하였다. 성능 평가 결과를 통해 제안된 가상 머신 모니터가 실제의 임베디드 시스템 응용 분야에 적용이 가능함을 보였다.

Abstract

Recently virtualization has been one of the most popular research topics in system software area. While there already exist many commercial virtualization products for general-purpose computer system, little efforts are made to virtualize embedded system. In this paper, we design and implement a virtual machine monitor which divides each physical hardware resource of a embedded system into logical ones and reorganizes them into many virtual machines, which result in running several real-time operating systems concurrently on a single embedded system. We measure various performance metrics of the virtual machine monitor developed on a real embedded system. The results of the study show that our virtual machine monitor has enough potentiality of its application to real-world embedded systems.

▸ Keyword : 임베디드 시스템(Embedded system), 가상화(Virtualization), 가상 머신(Virtual machine), 가상 머신 모니터(Virtual machine monitor)

• 제1저자 : 손성훈
• 투고일 : 2008. 11. 18, 심사일 : 2008. 11. 19, 게재확정일 : 2008. 12. 24.
* 상명대학교 컴퓨터과학부 조교수 ** 상명대학교 일반대학원 컴퓨터과학과 석사 과정
※ 이 논문은 2008년도 상명대학교 일반연구기관 선발과제의 연구비를 지원 받았음

Ⅰ. 서 론

가상 머신 모니터(Virtual Machine Monitor)는 하나의 시스템 상에서 다수의 운영체제를 동시에 수행할 수 있도록 하는 소프트웨어 계층이다. 시스템을 가상화하면 다양한 이득을 얻을 수 있다. 우선 물리적인 머신 수를 줄여 부피와 무게를 최소화할 수 있고, 시스템의 유휴 자원을 효율적으로 활용할 수 있다. 또한 시스템의 오동작으로 인한 영향을 특정 영역으로 국한시켜 안정성을 높일 수도 있다.

초기의 임베디드 시스템은 휴대성을 위해 작은 크기의 저성능 하드웨어를 사용함으로써 충분한 하드웨어 성능을 필요로 하는 가상화 기술의 적용이 어려웠다. 하지만 하드웨어 성능이 개선됨에 따라 최근 임베디드 시스템은 작은 크기의 고성능 하드웨어로 진화하였고, 이를 활용한 임베디드 시스템의 가상화가 가능해졌다. 하지만 임베디드 시스템은 범용 시스템에 비해 많은 제약을 가지기 때문에 범용 시스템을 대상으로 한 기존 가상 머신 모니터와는 다른 설계가 요구된다.

본 논문에서는 임베디드 시스템에 적합한 가상 머신 모니터를 설계하고 구현한다. 또한 제안된 가상 머신 모니터를 적용한 시스템의 수행 시간 등을 측정하고, 그 결과를 기반으로 제안된 가상 머신 모니터의 성능을 평가한다. 더불어 가상 머신 모니터를 효과적으로 수행하기 위한 최적의 타이머 발생 주기나 스케줄링 파라미터 등을 제시한다.

본 논문은 다음과 같이 구성된다. 1장에 이어, 2장에서는 임베디드 시스템의 가상화에 대한 관련 연구를 언급한다. 3장에서는 본 논문의 기반이 되는 가상 머신 모니터의 설계 및 구현 내용을 소개하고, 4장에서는 실제 임베디드 시스템에 가상 머신 모니터를 적용한 성능 측정 결과를 제시한다. 마지막으로 5장에서는 결론을 도출하며 향후 연구 방향을 모색한다.

Ⅱ. 관련 연구

1. 범용 시스템의 가상화

파라 가상화(Para virtualization)[1][2]는 가상 머신 모니터(Virtual Machine Monitor)라고 불리는 소프트웨어 계층을 통해 하나의 시스템에서 다수의 운영체제를 동시에 수행한다. 가상 머신 모니터는 하나의 물리 자원을 여러 개의 논리 자원들로 가상화하며, 가상화된 논리 자원들로 가상 머신(Virtual Machine)을 구성한다. 가상 머신 상에서 수행되는 게스트 운영체제(Guest Operating System)는 가상 머신에 할당된 논리 자원을 물리적인 자원으로 인식하여 동작한다.

범용 머신을 위한 파라 가상화 시스템으로 Xen[2], VMware ESX Server[3] 등이 상용화된 반면, 임베디드 시스템을 위한 가상화 연구는 아직 미미한 수준이다. 이는 기존 임베디드 시스템이 가상화를 감당할만한 충분한 하드웨어 성능을 지니지 못한 것으로 여겨졌기 때문이다. 하지만 점차 향상되고 있는 임베디드 시스템의 성능으로 볼 때, 향후 임베디드 시스템의 가상화로 얻는 이점이 가상화로 인한 오버헤드보다 클 것으로 예상된다[4,5].

2. 임베디드 시스템의 가상화

임베디드 시스템을 가상화하면 다음과 같은 이점을 얻을 수 있다[6]. 첫 번째로 복잡한 시스템을 분리할 수 있다. 두 번째로 하나의 기계에서 다수의 운영체제를 동시에 수행시킬 수 있다. 세 번째로 보안성과 안정성을 향상시킬 수 있다. 네 번째로 멀티코어 CPU로의 확장이 용이하다. 다섯 번째로 라이선스를 분리할 수 있다.

최근 임베디드 시스템을 가상화하기 위한 다양한 시도가 있다. [7]은 한 임베디드 시스템에서 CPU 시간을 균등하게 나누어 다수의 μC/OS-II를 번갈아 수행하는 방법을 제안하였다. 이 연구는 인터럽트 가상화의 기본 개념을 제시하고, 가상 머신 간 통신을 통해 가상 머신 사이에 정보를 전달할 수 있음을 보였다. 하지만 수행 중인 가상 머신을 동적으로 중지/재개할 수 없고, 특정 게스트 운영체제(μC/OS-II)만을 지원하기 때문에 활용도가 낮다. 또한 타이머 인터럽트 이외의 다른 인터럽트를 처리할 수 없고, 모든 게스트 운영체제에게 균등한 CPU 시간을 나누어 수행하기 때문에 게스트 운영체제의 실시간성 지원에 대한 고려가 미흡하다.

하나의 임베디드 시스템 상에서 실시간 운영체제인 μC/OS-II와 Nano-Q+를 동시에 수행하고, 두 운영체제 간 통신 기능을 가진 가상화 연구도 진행되었다[8]. 이 연구는 각 가상 머신의 태스크 개수와 상태를 파악하여 다음 수행할 가장 높은 우선순위의 가상 머신을 산출하는 DDOS(Dual OS Scheduler)라는 독자적인 스케줄링 방식을 사용하고, 키 인터럽트를 사용하여 인터럽트를 전달할 대상 가상 머신을 선택할 수 있으며, 인터럽트 전달 및 가상 머신 간 통신에 활용하기 위해 이벤트 채널이 존재한다. 하지만 하나의 μ

2) 국내 여러 연구들에서 서로 다른 명칭으로 사용되고 있어 혼란을 피하기 위해 영문 그대로 표기함.

C/OS-II와 하나의 Nano-Q+만 동시에 수행되도록 구성되어 있었고, 역시 가상 머신의 인터럽트 분배가 제한적이었다.

[9]는 L4 커널 상에서 QEMU를 이용하여 SPARC, PowerPC, ARM, x86 기반 Linux를 함께 수행시키는 연구를 진행하였다. 하나의 머신을 가상화하여 여러 아키텍처로 이식된 운영체제들을 수행할 수 있고, 다른 아키텍처로 Linux를 이식할 때 이를 사용한 편리한 디버깅을 제안한다.

본 논문에서는 시스템 동작 중에도 가상 머신을 동적으로 추가/삭제/중지/재개할 수 있고, 모든 종류의 인터럽트를 적절히 분배할 수 있으며, 여러 종류의 게스트 운영체제를 지원할 수 있는 임베디드 시스템을 위한 가상 머신 모니터를 제안한다.

III. 가상 머신 모니터의 설계 및 구현

본 장에서는 임베디드 시스템을 가상화 하기 위한 가상 머신 모니터의 설계 및 구현 방법을 제안한다.

1. 가상 머신 모니터의 구조

본 논문에서 제안하는 가상 머신 모니터의 전체적인 구조는 그림 1과 같다.

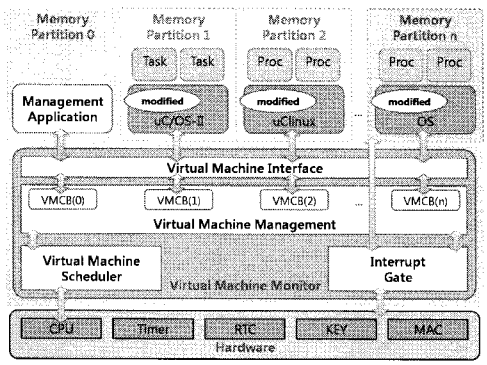

그림 1. 가상 머신 모니터의 구조
Fig. 1. Structure of virtual machine monitor

가상 머신 모니터는 하드웨어와 운영체제 사이에 존재하는 가상화를 위한 소프트웨어 계층이다. 가상 머신 모니터는 크게 가상 머신을 관리하는 부분, 인터럽트를 처리하고 분배하는 부분, 가상 머신을 스케줄링 하는 부분, 게스트 운영체제가 가상 머신 모니터에게 정보를 전달하는 인터페이스 부분 등으로 구성된다.

일반적으로 각 가상 머신에는 기존 운영체제를 수정한 게스트 운영체제가 등록되어 동작한다. 다만 VM0로 명명된 특수 가상 머신은 가상 머신 모니터가 초기화되는 과정에서 자동으로 생성되어 사용자의 가상 머신 모니터에 대한 명령 등을 처리한다.

본 연구에서 설계한 가상 머신 모니터는 에이디칩스사의 EISC 기반 SE3208 프로세서를 탑재한 Jupiter Education Board[10] 상에서 구현 되었고, 아키텍처 의존적인 코드를 분리하여 쉽게 다른 아키텍처로 이식할 수 있도록 하였다. 게스트 운영체제로는 임베디드 운영체제들 중에서 활용도가 높은 μClinux[11]와 μC/OS-II[12]를 수정하여 사용하였다.

2. 가상 머신의 관리

가상 머신 모니터는 가상 머신들을 관리하기 위한 가상 머신 정보를 가지고 있다. 가상 머신 모니터는 가상 머신 제어 블록(Virtual Machine Control Block, VMCB)이라는 가상 머신 구조체에 가상 머신의 수행 여부를 결정하기 위한 상태 정보, 가상 머신의 수행 빈도를 조절하는 스케줄링 파라미터, 가상 머신에 자원을 할당하기 위한 정보 등을 포함하고 있다.

가상 머신 모니터는 가상 머신의 상태에 따라 수행 유무나 인터럽트 전달 유무를 결정한다. 가상 머신의 상태는 여러 가상 머신이 동작하고 있는 중에도 변경이 가능하며, 기본적으로 VM0를 통해서 변경한다. 경우에 따라서 일반 가상 머신이 자기 자신을 포함한 특정 가상 머신의 상태를 변경할 수도 있다.

그림 2는 가상 머신의 상태와 상태 전이를 나타낸다. 각 상태가 의미하는 바는 다음과 같다.

FREE: 게스트 운영체제가 지정되지 않은 상태
SHUTDOWN: 게스트 운영체제가 종료 된 상태
BOOTING: 게스트 운영체제가 부팅 중인 상태
RUNNING: 게스트 운영체제가 수행 중인 상태
PAUSE: 게스트 운영체제가 수행 중지 된 상태

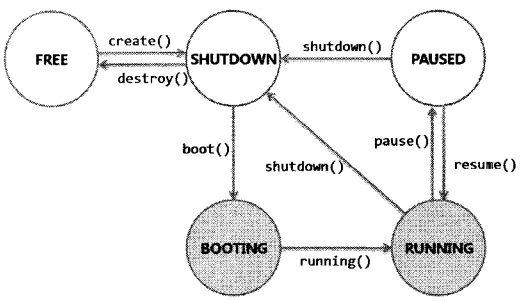

그림 2. 가상 머신의 상태 전이
Fig. 2. State transition of virtual machine

BOOTING 상태의 가상 머신은 인터럽트를 무시하도록 설정되고, 가상 머신이 RUNNING 상태로 변경될 때 가상 머신이 인터럽트를 처리할 수 있도록 조정한다. SHUTDOWN 상태는 가상 머신 재개 시에 게스트 운영체제의 초기화부터 시작하고, PAUSE 상태는 이전 수행을 중지했던 시점부터 시작한다.

3. 가상 머신의 스케줄링

가상 머신 모니터는 운영체제가 프로세스들을 수행하는 것과 유사한 방법으로 다수의 가상 머신들을 스케줄링 한다. 가상 머신 모니터는 매 타이머 인터럽트 발생 시마다 다른 가상 머신으로의 전환 여부를 검사하여, 스케줄링이 필요하다고 판단되면 다른 가상 머신으로 전환한다. 여기에는 현재 수행 중인 게스트 운영체제가 IDLE 상태로 전환되는 경우도 포함된다.

본 연구에서 구현된 가상 머신 모니터는 다음 수행할 가상 머신을 선택하기 위한 스케줄링 알고리즘으로 RR(Round-Robin) 스케줄러와 SEDF(Simple Earliest Deadline First) 스케줄러[13]를 지원한다. 특히 스케줄러 등록 인터페이스를 통해 새로운 스케줄러를 쉽게 추가할 수 있도록 하였다.

스케줄러에 의해 다음에 수행할 가상 머신이 결정되면 현재 수행 중인 가상 머신으로부터 선택된 가상 머신으로 전환이 발생하며, 그 과정은 다음과 같다.

① PC 등 특수 CPU 레지스터들을 스택에 저장.
② 인터럽트를 비활성화 상태로 변경.
③ 빈 스택 공간을 생성.
④ 범용 CPU 레지스터들을 스택에 저장.
⑤ SP를 현재 수행 중인 가상 머신의 VMCB에 저장.
⑥ 인터럽트 게이트를 수행하여 가상 머신을 전환하고 분기할 인터럽트 핸들러를 선택.
⑦ 다음 수행할 가상 머신의 VMCB에서 SP를 복구.
⑧ ③번의 빈 스택에 인터럽트 핸들러의 주소 저장.
⑨ 범용 CPU 레지스터들을 스택에서 복구.
⑩ ⑧번에서 저장한 인터럽트 핸들러 주소로 분기.
⑪ 인터럽트 서비스 루틴 수행.
⑫ 인터럽트를 활성화 상태로 변경.
⑬ PC 등 특수 CPU 레지스터들을 스택에서 복구.

4. 가상 머신의 인터럽트 처리

인터럽트를 처리하는 방법은 마이크로프로세서의 종류에 따라 상이하다. 여기서는 본 논문의 개발 환경인 Jupiter Education Board 상의 SE3208 프로세서의 인터럽트 처리 방식을 기준으로 설명한다.

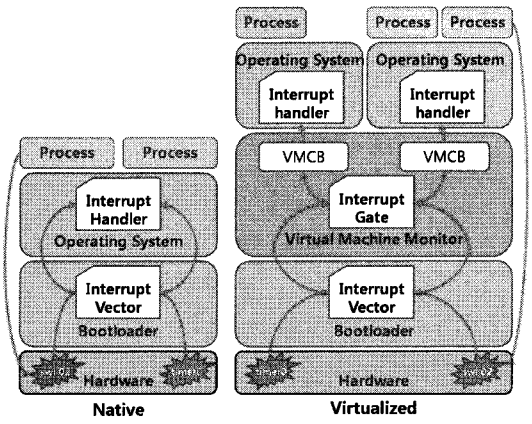

그림 3. 인터럽트 처리 방법 비교
Fig. 3. Comparison of interrupt handling

그림 3은 기존 운영체제와 비교하여 가상 머신 모니터의 인터럽트 처리를 나타낸 그림이다. 가상 머신 모니터는 초기화 시 인터럽트 벡터가 지정한 영역에 인터럽트 게이트를 구성한다. 그리고 게스트 운영체제는 게스트 운영체제가 사용하는 메모리 공간 중 특정 영역에 인터럽트 핸들러를 구성한다. 이 경우 가상 머신 모니터의 인터럽트 게이트가 발생하는 모든 인터럽트를 우선적으로 처리하기 때문에 게스트 운영체제들에게 상황에 맞게 인터럽트를 분배할 수 있다.

가상 머신 모니터의 인터럽트 게이트는 인터럽트가 발생하면 해당 인터럽트를 어떤 가상 머신으로 전달할지 결정한다. 소프트웨어 인터럽트와 같은 내부 인터럽트는 항상 현재 수행 중인 가상 머신에서 발생하므로 현재 수행 중인 가상 머신에게 전달된다. 하지만 IRQ와 같은 외부 인터럽트는 외부 장치에서 발생되므로 어떤 가상 머신에게 전달해야 하는지 불분명하다. 따라서 가상 머신 모니터는 인터럽트 분석기를 사용하여 외부 인터럽트와 가상 머신의 관계를 파악한다.

인터럽트 분석기는 외부 인터럽트를 크게 세 가지 기준으로 분류한다. 첫째, 어떤 가상 머신도 사용하지 않는 인터럽트라면 이를 무시한다. 둘째, 특정 가상 머신이 독점하는 인터럽트라면 해당 가상 머신의 VMCB에 인터럽트가 발생했음을 기록한다. 셋째, 모든 가상 머신이 공유하는 인터럽트라면 인터럽트 특성에 따라 가상 머신을 결정하고 해당 VMCB에 인터럽트가 발생했음을 기록한다. 모든 가상 머신이 공유하는 인터럽트는 가상 머신 모니터를 초기화 할 때 지정할 수 있으며, 특정 가상 머신이 독점하는 인터럽트는 가상 머신을 생성할 때 설정할 수 있다.

5. 가상 머신 모니터의 메모리 관리

가상 머신 모니터는 각 가상 머신이 사용하는 메모리 공간을 관리한다. 그림 4는 가상화 된 환경에서 가상 머신 모니터와 게스트 운영체제의 메모리 맵이다.

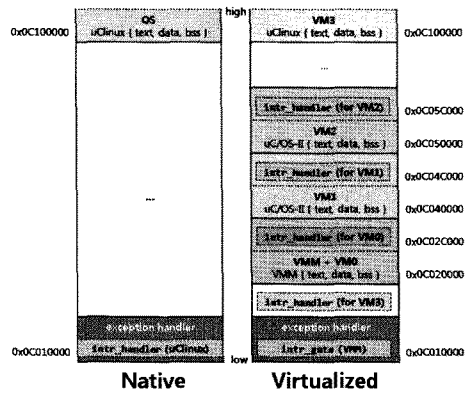

그림 4. Jupiter education board에서의 메모리 맵
Fig. 4. Memory map on Jupiter education board

본 논문에서 제안한 가상 머신 모니터는 MMU(Memory Management Unit) 기능이 없는 프로세서 상에서 구현되었기 때문에, 링크 시에 각 게스트 운영체제들을 서로 다른 메모리 영역에 배치해야 한다. 특히 게스트 운영체제가 부팅하면서 초기화되는 인터럽트 핸들러는 해당 운영체제의 영역에 위치시켜 다른 영역에 겹치지 않도록 주의해야 한다.

6. 가상 머신의 시간

일반적으로 운영체제는 타이머 인터럽트를 기준으로 시간의 경과를 체크한다. 하지만 가상 머신 모니터를 사용하면 현재 수행 중인 가상 머신만 타이머 인터럽트를 처리할 수 있기 때문에 실제 시간과 게스트 운영체제의 시간에 차이가 발생한다. 게스트 운영체제의 실시간성을 보장하려면 실제 시간을 통해 가상 시간을 보정하는 작업이 필요하다. 타이머 인터럽트가 발생하면 가상 머신 모니터는 실제 시간을 갱신하고, 게스트 운영체제는 타이머 인터럽트 처리 루틴에서 갱신된 시간을 전달받아 가상 시간과 관련된 부분을 보정한다. 이를 위해 게스트 운영체제의 수정이 필요하다.

7. 게스트 운영체제의 수정

가상 머신 모니터는 가상 머신에서 동작하는 게스트 운영체제의 수정이 필요하다.

첫 번째로 가상 머신 모니터와 게스트 운영체제 간 통신을 위해서 게스트 운영체제에 가상 머신 모니터의 인터페이스를 연결하도록 추가해야 한다. 두 번째로 운영체제마다 독립적인 인터럽트 핸들러를 배치하기 위해서 인터럽트 핸들러의 위치를 해당 운영체제의 영역으로 수정해야 한다. 세 번째로 하드웨어 초기화의 중복을 방지하기 위해서 게스트 운영체제의 하드웨어 초기화 과정을 수정해야 한다. 네 번째로 게스트 운영체제가 부팅을 완료할 때, 가상 머신 모니터에게 게스트 운영체제의 부팅 종료를 알리는 과정을 추가해야 한다. 다섯 번째로 게스트 운영체제가 IDLE 상태로 진입할 때, 가상 머신 모니터에게 게스트 운영체제의 상태 변경을 알리는 과정을 추가해야 한다. 마지막으로 가상 머신 모니터가 가지고 있는 실제 시간과 게스트 운영체제가 인식하는 실제 시간의 차이를 보정하기 위해 게스트 운영체제의 시간 처리 관련 루틴을 수정해야 한다.

IV. 가상 머신 모니터의 성능 분석

본 장에서는 제안된 가상 머신 모니터의 성능 측정 결과를 분석한다. 모든 실험은 Jupiter education board 상에서 진행하였다. 가상 머신 모니터는 Cygwin 환경에서 C언어로 개발되었으며, 게스트 운영체제로는 μC/OS-II 2.52와 linux kernel 2.4 기반의 μClinux를 수정하여 사용하였다.

1. 가상 머신 수행 시간

우선 가상 머신 모니터 없이 단독 수행한 경우와 가상 머신 모니터를 통해 다수의 가상 머신들을 수행한 경우의 수행 시간을 비교하였다. 이 실험에서는 매 100tick마다 0부터 255까지의 수를 LED에 출력하는 태스크, 통계 태스크, 그리고 IDLE 태스크로 구성되는 μC/OS-II를 사용하였으며, 타임 퀀텀이 10tick인 RR 스케줄링을 사용하였다.

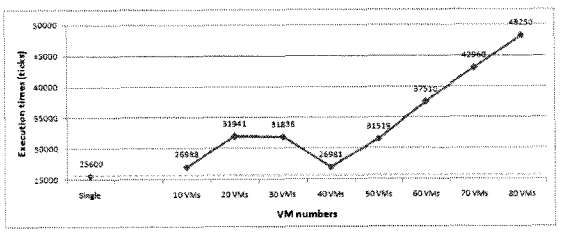

그림 5. 가상 머신의 수에 따른 수행 시간
Fig. 5. Execution time of virtual machine

그림 5는 가상 머신 개수에 따른 수행 시간의 변화를 나타낸다. 기본적으로 가상 머신 모니터를 통해 가상 머신들을 수행하면 가상 머신 관리를 위한 시간이 추가적으로 발생하여 가상화를 사용하지 않은 경우보다 많은 수행 시간을 소모한다. 그러나 40개의 가상 머신이 수행될 때까지는 가상화되지 않고 수행되는 경우와 비교적 유사한 수행 시간을 소모한다. 이는 한 가상 머신의 유휴 시간동안 가상 머신 모니터가 다른 가상 머신들을 스케줄링 하여 수행시키기 때문이다. 반면 가상 머신 개수가 40개 이상인 경우는 가상 머신 개수에 비례하여 수행 시간이 증가하는 것을 확인할 수 있다. 이는 한 가상 머신의 유휴 시간동안 나머지 가상 머신들을 모두 수행시킬 수 없기 때문이다.

이 실험에서 확인한 바와 같이 대기 시간이 많은 가상 머신의 경우 그 동안 다른 가상 머신을 수행할 수 있기 때문에 CPU를 효율적으로 사용할 수 있다. 임베디드 시스템은 수행 시간의 대부분을 대기 시간으로 소모하는 경우가 많기 때문에 가상 머신 모니터를 사용하면 대기 시간을 활용한 효율적인 수행이 가능하다.

2. 타이머 주기 최적화

가상 머신 모니터는 타이머 인터럽트가 발생할 때마다 시스템의 시간을 갱신하고 스케줄링 조건을 확인하여 가상 머신을 전환한다. 타이머 주기가 짧을수록 정밀한 시간 단위의 수행은 가능하나 인터럽트 처리에 대한 부담이 증가한다. 이 실험은 가상 머신 모니터에서의 최적의 타이머 주기를 알아보기 위한 실험이다.

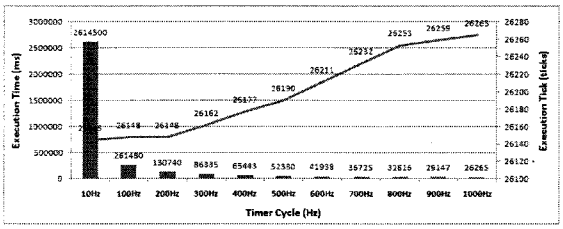

그림 6. 타이머 주기에 따른 수행 시간 및 tick 수
Fig. 6. Execution time and ticks

그림 6은 앞 절과 동일한 실험 환경 하에서 타이머 주기에 따른 수행 시간 및 tick 수를 측정한 결과이다. 꺾은선 그래프는 타이머 주기에 따른 tick 수를 나타낸다. 그림에서 보는 바와 같이 타이머 주기가 짧을수록 오버헤드가 가중되어 tick 수는 증가한다. 막대 그래프는 타이머 주기에 따른 수행 시간을 나타낸다. 타이머 주기가 길어질수록 수행 시간은 증가하며, 특히 10Hz와 100Hz 사이에서 급격히 증가한다. 타이머 주기는 타이머 인터럽트 처리의 오버헤드를 최대한 줄이면서 가급적 정밀한 시간 측정이 가능하도록 설정하는 것이 좋다. 이 실험의 경우에는 수행 tick이 작으면서 타이머 주기가 작은 200Hz가 최적의 타이머 주기가 된다.

3. 스케줄러 최적화

가상 머신의 워크로드 유형에 따른 SEDF 스케줄러의 최적의 slide와 period 값을 찾기 위한 실험을 진행하였다. 사용된 가상 머신의 워크로드는 다음과 같다.

- 96% VM: CPU-bound 태스크 9개와 I/O-bound 태스크 1개로 구성된 µC/OS-II. (CPU Usage 96%)
- 60% VM: CPU-bound 태스크 5개와 I/O-bound 태스크 5개로 구성된 µC/OS-II. (CPU Usage 60%)
- 12% VM: CPU-bound 태스크 1개와 I/O-bound 태스크 9개로 구성된 µC/OS-II. (CPU Usage 12%)

표 1. 워크로드 유형과 period/slide 조합에 따른 전체 CPU 사용률
Table 1. CPU usage for workload type and period/slide

(a)

VM1	VM2	VM3	Usage(%)
1/10	2/10	3/10	59
3/10	2/10	1/10	59
10/100	20/100	30/100	47
30/100	20/100	10/100	53

(b)

VM1	VM2	VM3	Usage(%)
1/10	2/10	3/10	59
3/10	2/10	1/10	59
10/100	20/100	30/100	21
30/100	20/100	10/100	25

(c)

VM1	VM2	VM3	Usage(%)
1/10	2/10	3/10	39
3/10	2/10	1/10	39
10/100	20/100	30/100	5
30/100	20/100	10/100	5

(d)

VM1	VM2	VM3	Usage(%)
1/10	2/10	3/10	44
3/10	2/10	1/10	59
10/100	20/100	30/100	18
30/100	20/100	10/100	36

표 1은 가상 머신의 워크로드 유형별로 SEDF 스케줄러의 period와 slide를 변경하면서 전체 CPU 사용률을 측정한 결과이다(설명의 편의를 위해 period와 slide를 period/slide로 표기한다). 표 1에서 (a)는 CPU 사용률 96%인 가상 머신 3개를 수행한 결과이고, (b)는 CPU 사용률 60%인 가상 머신 3개를 수행한 결과이며, (c)는 CPU 사용률 12%인 가상 머신 3개를 수행한 결과이고, (d)는 각 가상 머신을 한 개씩 총 3개 수행한

결과이다. 이 실험을 통해 SEDF 스케줄러는 워크로드 유형에 따라 다음과 같은 몇 가지 특징을 가지고 있음을 알 수 있다.

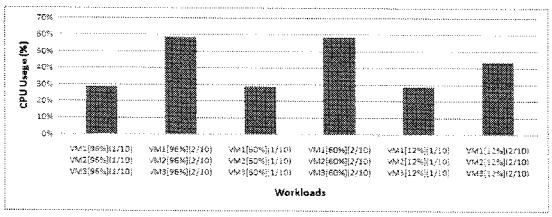

그림 7. period의 차이에 따른 전체 CPU 사용률
Fig. 7. CPU usage for period

그림 7을 통해 가상 머신의 period 값을 크게 설정할수록 전체 CPU 사용률이 향상되는 것을 알 수 있다. SEDF 스케줄러에서 period 값은 CPU의 사용 시간을 결정한다. 따라서 CPU를 많이 사용하는 가상 머신은 period 값을 크게 주고, CPU를 적게 사용하는 가상 머신은 period 값을 작게 설정하는 것이 좋다.

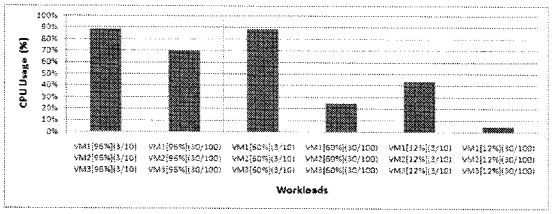

그림 8. slide의 차이에 따른 전체 CPU 사용률
Fig. 8. CPU usage for slide

그림 8은 SEDF 스케줄러의 slide 값이 작을수록 더 높은 전체 CPU 사용률을 얻을 수 있다는 것을 보여준다. SEDF 스케줄러에서 slide 값은 작을수록 우선적으로 스케줄링 된다. 따라서 반드시 처리해야 하는 중요한 가상 머신일수록 slide 값을 작게 설정하는 것이 좋다.

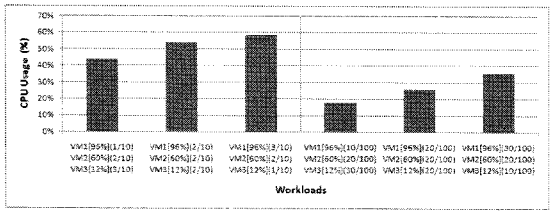

그림 9. 각 가상 머신의 CPU 사용률이 다른 경우의 전체 CPU 사용률
Fig. 9. CPU usage for VM's CPU usage

그림 9는 각 가상 머신의 CPU 사용률이 다른 경우에 slide와 period에 따른 전체 CPU 사용률 측정 결과이다. 이를 통해 높은 CPU 사용률을 보이는 가상 머신에 상대적으로 period를 많이 할당하는 것이 전체 CPU 사용률을 증가시킬 수 있는 방법이다. 단, CPU 사용량이 많은 가상 머신의 period 값을 작게 설정하여 QoS를 제공할 수 있다. 컴퓨팅 파워가 많이 필요한 가상 머신과 적게 필요한 가상 머신에 차등을 두어 일정 수준 이상으로 CPU를 사용할 수 없도록 제한할 수 있다.

V. 결론 및 향후 연구

임베디드 시스템의 가상화에 대한 연구는 아직 초기 단계에 머무르고 있다. ARM 프로세서로 이식된 Xen이 있지만, 범용 시스템을 대상으로 설계한 Xen의 특성 상 임베디드 시스템용 가상화 제품이라고 보기는 어렵다. 다른 임베디드 시스템의 가상화에 대한 연구 또한 동적인 가상 머신 관리가 어렵거나 특정 운영체제만 지원하는 등 한계를 가지고 있다.

본 논문에서 제안하는 가상 머신 모니터는 동적으로 가상 머신을 생성/제거/정지/재개할 수 있고, μC/OS-II와 μClinux 등 여러 임베디드 시스템 운영체제를 지원하며, 가상 머신 스케줄러를 손쉽게 추가할 수 있고, 인터럽트 분배에 대한 확장성을 제공한다.

향후 ARM과 같이 MMU 기능이 있는 프로세서로 이식하면 마이그레이션 기능을 추가할 수 있어 특정 시스템에 부하가 집중될 때 가상 머신을 이동시켜 효율적으로 부하를 분산시킬 수 있다. 또한 실시간성에 대한 지원을 강화하기 위해 인터럽트 처리 및 가상 머신 스케줄러에 대한 추가 연구가 필요하다.

참고문헌

[1] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and Performance in The Denali Isolation Kernel," ACM SIGOPS Operating Systems Review, Vol. 36 , Issue SI (Winter 2002), pp. 195-209, 2002.

[2] 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," ACM Symposium on Operating Systems Principles, in Proceedings of the nineteenth

ACM symposium on Operating systems principles, pp. 164-177, 2003.
[3] VMware, Inc., *"VMware ESX 3.5 Product Datasheet,"* VMware, Inc.
[4] 유시환, 유혁, "임베디드 시스템 가상화: 도전과 과제," 한국정보과학회 2007 한국컴퓨터종합학술대회 논문집, 제34권, 제1호(B), 304-309쪽, 2007년 6월.
[5] 손성훈, 전상일, "Xen 기반 가상 데스크탑 서비스를 위한 부하 분산 정책," 한국컴퓨터정보학회논문지, 제13권, 제1호, 55-64쪽, 2008년 1월.
[6] Gernot Heiser, "Virtualization for Embedded Systems," Open Kernel Labs, Inc., pp. 10-12, 2007.
[7] 신동하, 김지연, "uC/OS-II 실시간 커널의 가상화를 위한 하이퍼바이저 구현," 한국컴퓨터정보학회논문지, 제12권, 제5호, 103-112쪽, 2007년 11월.
[8] 양종철, 김한빛, 조상준, 조재일, 안우현, "임베디드 시스템에서의 실시간 운영체제 가상화 설계 및 구현," 한국정보과학회 2008 종합학술대회 논문집, 제35권, 제1호(B), 308-312쪽, 2008년 6월
[9] Y. Kinebuchi, H. Koshimae, S. Oikawa, and T. Nakajima, "Virtualization Techniques for Embedded Systems," The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, in Proceedings of the Work-in- Progress Session, Aug. 2006.
[10] Advanced digital chips, Inc., *"Jupiter Education Board Manual,"* Advanced digital chips, Inc.
[11] μClinux Project, *http://www.uclinux.org*
[12] Jean J. Labrosse, *"MicroC/OS-II Real-Time Kernel,"* CMP Media, Inc., 2002.
[13] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The Design and Implementation of An Operating System to Support Distributed Multimedia Applications," IEEE Journal, on Selected Areas In Communications, 14(7):1280-1297, Sept. 1996.

저 자 소 개

손 성 훈

1991: 서울대학교 계산통계학과 학사
1993: 서울대학교 전산과학과 석사
1999: 서울대학교 전산과학과 박사
2004: 한국전자통신연구원 선임연구원
현 재: 상명대학교 컴퓨터과학과 조교수
관심분야: 임베디드 컴퓨팅, 가상화

이 재 현

2008: 상명대학교 컴퓨터과학과 학사
현 재: 상명대학교 일반대학원 컴퓨터과학과 석사과정
관심분야: 임베디드 컴퓨팅, 가상화