[1]가상 머신 환경에서의 타이머 정확도 향상 기법

곽근환*, 유시환, 유혁
*고려대학교 컴퓨터 학과
e-mail : {khkwak, shyoo, hxy}@os.korea.ac.kr

Precise Software Timer Architecture for Virtual Machine Environment

Kuen Hwan Kwak*, See-Hwan Yoo, Hyuck Yoo
* Department of Computer Science, Korea University

요 약

최근에는 일반 운영체제에도 점점 다양한 소프트웨어들이 동작하게 되면서 보다 정확한 타이머의 성능을 요구하는 경우가 늘어나고 있다. 타이머의 낮은 성능은 실시간 태스크의 실행을 보장하지 못하게 되며, 이는 시스템의 응답성을 저하시키는 요인이 된다.. 본 논문에서는 가상화된 환경에서 타이머의 오차 문제를 해결하기 위해 새로운 타이머 구조를 이용해서 정확도를 향상하는 기법을 제안한다. 그리고 이를 통해서 가상 머신 환경에서도 실시간 응답성을 필요로 하는 태스크들을 정확한 시간에 실행 할 수 있음을 보인다.

1. 서론

가상화 기술은 하나의 물리 머신에 여러 개의 게스트 운영체제를 동작 시킴으로써 CPU 활용도를 높이면서 기존의 성능을 그대로 유지하는데 그 목적이 있다. 현재의 가상화 기술은 서버 환경에 초점이 맞추어져 있지만, 임베디드 환경에서도 가상화 기술은 소프트웨어의 신뢰성과 재사용성 향상을 위해서 필수적이다. 하드웨어 환경이 빠르게 바뀌는 상황에서 기존의 소프트웨어를 최소한의 수정만으로 재사용 가능하며, 가상화의 구조적인 특징을 통해서 전체 시스템의 신뢰성을 높일 수 있다는 점은 임베디드 가상화의 큰 장점이다.

임베디드 가상화 환경에서는 서버환경과 달리 응답성이 중요한 성능 요소가 된다. 임베디드 환경에서는 하이퍼바이저가 CPU 를 공평하게 나누는 역할 외에도 실시간성을 필요로 하는 태스크를 정확한 시간에 실행 시켜주는 것도 중요하기 때문이다. 예를 들어, 하나의 물리 머신에 2 개의 게스트 운영체제를 실행하고, 각각에 멀티미디어 재생과, 데이터 처리를 동시에 하고 있다고 가정하면, 기존의 가상화 환경에서는 CPU 의 대역폭을 동일하게 할당 해주지만 멀티미디어 재생하는 게스트 운영체제의 타이머의 실행에 대해서는 보장해 주지 않기 때문에 사용자는 제대로 된 멀티미디어를 볼 수 없게 된다.

하지만, 기존의 가상화 기술은 구조적인 문제로 인해 높은 응답성을 제공 할 수 없다. 가장 큰 원인은 게스트 운영체제가 필요한 물리 머신의 자원을 필요할 때 제공받지 못하기 때문이다.

본 논문에서, 이런 문제점을 해결 하기 위한 새로운 타이머를 구조를 제시하고 실험 및 결과를 통해서 그 결과를 보인다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서 본 논문과 관련된 연구를 설명하고, 3 장에서는 가상 머신 타이머의 문제점을 설명한다. 4 장에서는 문제를 해결하기 위한 새로운 타이머 구조와 구현에 대해서 설명하였으며, 5 장에서는 실험을 통한 타이머 성능 향상을 보인다.

2. 관련 연구

[1] 는 범용 운영체제에서 다양한 종류의 태스크가 동시에 실행되는 환경에서 멀티미디어 주기적인 작업의 실시간 성을 만족시키기는 스케줄러를 구현하고 성능을 평가하였다.

[2] 의 연구에서는 태스크의 QoS 를 보장하기 위해서는 CPU 스케줄링만으로는 부족하기 때문에 디스크 스케줄링을 동시에 수행해서 실시간 태스크의 읽기 작업을 제시간에 만족시킬 수 있도록 하였다. 이는 CPU 와 디스크 스케줄링을 통합하는 것으로 태스크의 연성 실시간성을 보장할 수 있다는 것을 보였다.

[3]는 고정밀 타이머(High Resolution Timer or hrtimer) 구현을 위해서 여러가지 하드웨어 인터럽트와 클럭을 동시에 이용하는 구조를 제안하였다.

위 세 가지 연구 모두 연성 실시간성 보장을 위한 스케줄러와 타이머 구조를 제안하였지만, 운영체제가

[1] 이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2010-0029180)

물리 하드웨어를 완전히 사용할 수 있는 비가상화 환경이라는 점이 본 논문과는 다르다고 할 수 있다.

기존의 연구 중에서도 가상화 환경에서 IO 의 성능을 향상시키고 낮은 응답시간을 제공하기 위한 연구가 있었다. [4]에서는 가상화 환경에서 스케줄러에 따른 응답시간의 측정을 통해서 SEDF 과 Credit 스케줄러의 성능을 비교하였다. [5] 는 태스크의 정보를 Gray-Box 방식으로 수집해서 이를 활용한 스케줄링을 통해 CPU 를 공평하게 스케줄링 하면서도 IO 의 성능을 향상시킬 수 있었다.

위 두 가지 연구는 가상화 환경이라는 점에서 본 논문에서는 스케줄러의 수정을 통한 IO 의 성능 향상이 아닌 게스트 운영체제의 타이머의 성능 문제를 지적하고 이를 향상시키기 위한 타이머 구조를 제안한다.

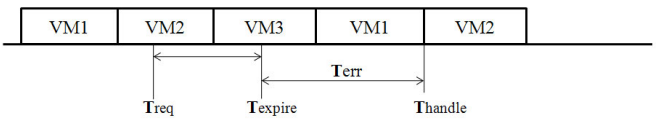
3. 가상 머신 환경의 타이머 문제점

본 절에서는 가상화된 임베디드 가상 머신 환경에서의 문제점과 그것을 해결하기 위한 타이머 구조에 대해서 자세히 설명하도록 한다.

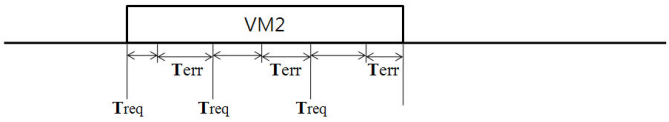
가상 머신 환경에서의 정확도가 낮아지는 원인은 두 가지로 정리할 수 있다. 첫째, 하이퍼바이저의 스케줄링으로 인해 게스트 운영체제의 타이머는 제한된 정확도를 가진다. 하이퍼바이저는 게스트 운영체제의 타이머 이벤트에 대해 알지 못하기 때문에, 게스트 운영체제의 타이머가 만료되더라도 하이퍼바이저는 우선적으로 해당 게스트 운영체제를 스케줄링 할 수 없다.

둘째, 가상머신이 가지는 타이머 장치는 물리 머신의 타이머보다 정확성이 떨어진다. 하이퍼바이저는 물리 타이머 장치를 가상화하여 게스트 운영체제에게 제공한다. 게스트 운영체제가 제공 받는 가상화된 타이머 장치는 물리 타이머 장치에 비해 낮은 정밀도를 가진다. 물리 머신에서 정밀한 타이머 처리를 위해 고정밀 타이머 (High Precision Timer or HPET) 인터럽트가 사용되는데 반하여, 임베디드 가상머신에서는 전적으로 주기적으로 발생하는 타이머 인터럽트에 의존할 수 밖에 없다.

특히 임베디드 환경에서는 타이머 처리의 오버헤드로 인해, 주기적 타이머 인터럽트가 매우 긴 간격으로 발생하며 1 개의 틱은 10ms 정도의 시간 간격으로 동작하게 된다. 따라서 게스트 운영체제에서 10ms 보다 정밀한 시간 간격으로 타이머를 요청하게 될 경우, 제시간에 해당 타이머의 핸들러를 실행 할 수 없다.



(그림 1) 기존 가상 머신 스케줄링의 타이머 오차



(그림 2) 틱 기반의 스케줄링에서 타이머 오차

그림 1 은 일반적인 게스트 운영체제간에 스케줄링에 의한 타이머 오차를 나타낸다. VM1 과 VM2 는 각각 게스트 운영체제를 나타낸다. VM2 가 요청한 타이머는 **T**expire 시점에 정확히 핸들러가 실행되어야 하지만 VM3 와 VM1 가 실행된 이후에 VM2 가 스케줄링 된 시점에서 실행된다. 즉, 타이머 핸들러는 예상한 시간에서 훨씬 벗어난 시점(**T**handle)에 실행 된다. Terr 만큼의 오차가 발생하게 된다.

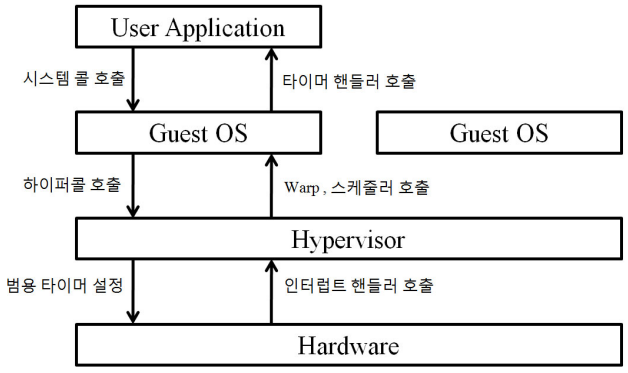
그림 2 은 가상의 타이머 인터럽트로 인한 정확도 문제에 대해서 나타낸다. VM2 가 실행되면서 10ms 마다 2ms, 4ms, 6ms 를 요청하였지만 틱 기반의 분해능으로 타이머 만료를 체크하기 때문에 각각 10ms, 20ms, 30ms 시점에 타이머 핸들러가 실행된다. 즉 10ms 이하의 오차(**T**err)가 매번 발생하는 것이다.

가상 머신 환경에서는 위 두 가지 원인으로 인해 타이머 정확도가 낮아지게 된다. 다음 절에서는 문제점을 해결하기 위한 새로운 타이머 구조를 제안한다.

4. 새로운 타이머 구조와 구현

새로운 타이머의 구조는 하이퍼바이저가 타이머 정보를 직접 관리하고 물리하드웨어 자원을 직접 사용하게 함으로써 기존의 문제점을 해결한다. 새로운 타이머의 구조는 그림 3 과 같다. 응용 프로그램은 새로운 타이머를 사용하기 위하여 시스템 콜을 호출한다. 게스트 운영체제의 시스템 콜은 하이퍼콜을 통해서 타이머 정보를 하이퍼바이저에 저장한다. 하이퍼콜은 해당 타이머의 만료 시간을 하드웨어의 범용 타이머를 통해서 인터럽트가 발생하도록 설정한다.

타이머의 만료시간이 되면 물리하드웨어에서 인터럽트가 발생한다. 하이퍼바이저에 인터럽트 핸들러가 호출이 되고 인터럽트 핸들러에서는 해당 게스트 운영체제를 가장 먼저 스케줄링이 되도록 하며 즉시 스케줄러를 호출 함으로써 타이머를 지연 시간 없이 빠르게 처리 할 수 있다.



(그림 3) 새로운 타이머 호출 구조

새로운 구조에서는 응용 프로그램에서 타이머의 설정이 요청되면 하이퍼바이저에서 추가의 타이머 인터럽트를 직접 할당하기 때문에 하이퍼바이저 내의 주기적인 타이머를 통해서 검사하는 것보다 작은 오차로 타이머 처리가 가능하다. 그 외에도 타이머 인터럽트가 발생 했을 때, 하드웨어 인터럽트부터 게스트 운영체제의 호출까지 대기하는 부분이 전혀 없기 때문에 정해진 시간 내에 타이머 핸들러 호출이 가능하다.

하이퍼바이저는 Xen-ARM [6]을 사용하였으며, 게스트운영체제내의 타이머의 구현은 XenoLinux 2.6.21 버전에 시스템 콜 중 하나인 sys_nanosleep 을 수정해서 구현하였다. 기존의 sys_nanosleep 함수는 HR-timer(Hight Resolution)를 통해서 구현이 되어 있지만 앞에서 제시한 대로 하이퍼 콜을 통해서 도메인과 타이머 정보를 저장하고 해당 시간에 하드웨어 타이머 인터럽트가 발생하도록 한다. 실험에 사용한 하드웨어 환경(FreeScale-iMX2)에서는 총 4 개의 범용 타이머가 있고 Xen-ARM 이 사용하는것은 1 개뿐이다. 따라서 나머지 3 개의 범용 타이머를 게스트 운영체제의 타이머에 활용 한다. sys_nanosleep 이 호출되고, 게스트 운영체제에서 하이퍼 콜을 호출하면 하이퍼바이저에서는 하드웨어 타이머 인터럽트를 깨어나야 하는 시간으로 설정을 한다. 요청한 Sleep 시간이 지나고 인터럽트 핸들러가 호출되면 Xen-Arm 에서 사용하는 BVT 스케줄링[7] 환경에서 타이머가 완료된 태스크가 있는 게스트 운영체제가 우선 실행될 수 있도록 Warp 를 시키고 BVT 의 스케줄러를 즉시 실행 시켜서 게스트 운영체제가 바로 수행 되도록 한다.

5. 실험 환경 및 결과

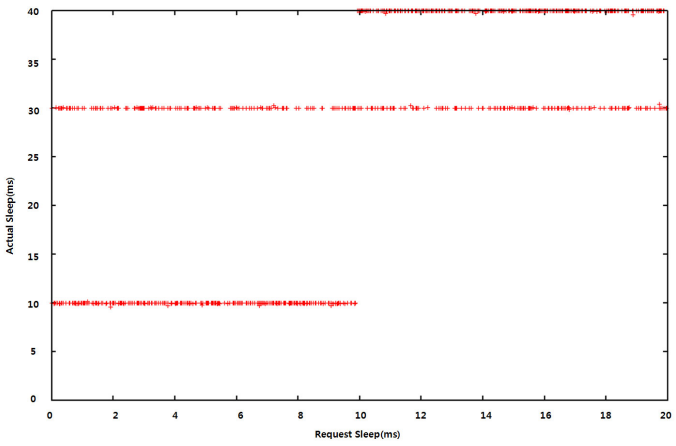
기존의 시스템의 타이머 정확도를 측정하기 위해서 가정한 실험 환경은 2 개의 Domain 이 실행 되면서 1 개의 도메인에서는 멀티미디어를 재생하는 Mplayer 가 주기적으로 타이머 함수를 호출하고 다른 1 개의 도메인 에서는 CPU 를 사용하는 행렬 곱셈 연산을 수행한다. 위의 실험에서 CPU 를 공평하게 분배하는 기존의 스케줄링 환경의 경우에 타이머 성능을 보이고 새로운 타이머 구조를 통해서 동일한 실험을 하였을 때와 비교해서 성능이 향상되는 것을 보인다. 타이머의 정확도는 타이머를 사용할 때 원하는 시간에서 얼마나 벗어난 이후에 깨어나서 실행 하는가로 판단한다. 타이머가 만료된 이후에 짧은 시간 이내에 실행되는 것을 보장한다면 태스크의 실시간성을 만족시킨다고 할 수 있다.

Mplayer 는 대표적인 미디어 플레이어로 임베디드 장치에서도 많이 쓰인다. Mplayer 는 동영상을 재생할 때 순서대로 하나의 프레임을 디코딩, 재생 그리고 다음 프레임까지 Sleep 을 주기적으로 반복하는데 Mplayer 는 매 프레임 마다 다음 프레임을 처리해야 할 시간을 계산해서 Sleep 을 한다. 만약 Sleep 을 한 시간보다 늦게 Mplayer 가 깨어난다면 동영상의 프레임이 밀리거나 음성과 동기화가 맞지 않는 현상이 발생하게 되는 것이다. 본 실험에서 Mplayer 를 통해서 테스트한 동영상의 상세 정보는 표 1 과 같다.

동영상 코덱	Xvid(MPEG-4)
동영상 크기	240 x 180
초당 프레임 수	20 fps
재생 시간	1 분 19 초

(표 1) 실험에 사용한 동영상 파일 정보

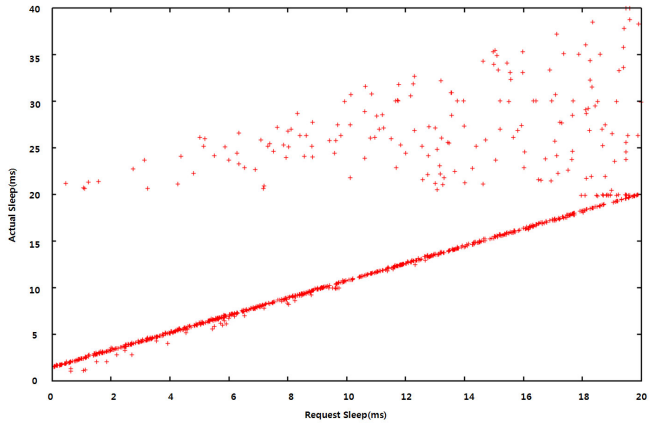
실험은 Mplayer 를 일부 수정해서 시간을 기록하는 방법으로 결과를 확인하였다. Mplayer 는 Frame 을 처리하고 다음 Frame 을 처리하기까지 Sleep 을 요청한다. Mplayer 가 요청하는 Sleep 시간은 0 부터 20 까지 다양하게 나타난다.



(그림 4) 기존 시스템에서 Mplayer 의 Sleep 성능

그림 4 의 그래프의 가로 축은 Mplayer 가 요청한 Sleep 시간을 나타내며 세로 축은 실제로 Sleep 한 시간을 나타낸다. 단위는 ms 이다. 0~10ms 사이의 Sleep 요청은 대부분 10ms 와 30ms 에 몰려서 처리 된 것을 확인 할 수 있다. 10ms 에 몰려서 처리된 Sleep 요청의 경우에는 게스트 운영체제가 H/W 의 타이머를 직접 사용하지 못하고 하이퍼바이저가 제공하는 틱(10ms) 단위로 가상 타이머 인터럽트로 처리 되기

때문에 생기는 문제라고 할 수 있다. 30ms 와 40ms 에 몰려서 처리되는 Sleep 요청은 Sleep 하는 중에 다른 게스트 운영체제가 실행되고 이후 다시 스케줄링을 받아서 처리되는 경우이다. 그림 4 를 통해서 Mplayer 가 원하는 타이머 정확도가 제대로 만족되지 않는 것을 확인 할 수 있다. 측정결과 평균 오차는 7ms 이다.



(그림 5) 새로운 시스템에서 Mplayer 의 Sleep 성능

그림 5 는 새로운 타이머 구조를 통해서 동일한 실험을 한 결과이다. 마찬가지로 가로축은 요청한 Sleep 시간, 세로축은 실제로 동작한 Sleep 시간이다. 앞의 결과와는 다르게 대부분의 타이머가 2ms 이내에 제대로 실행 된 것을 확인 할 수 있다. 하이퍼바이저가 타이머의 만료되는 시점에 바로 게스트 운영체제를 스케줄링 하기 때문에 게스트 운영체제의 스케줄링에 따른 문제와 틱 기반의 스케줄링에 따른 문제가 제거 된 것을 알 수 있다. 새로운 타이머의 경우 평균 타이머 오차가 400us 로 줄어들었다. 즉, 가상화 환경에서도 Mplayer 가 요구하는 실시간성이 제안된 타이머 구조를 통해서 만족된다고 볼 수 있다.

6. 결론

임베디드 시스템에서는 정확한 시간에 요청한 작업을 하는 응답성이 중요하다. 기존의 가상화 환경에서는 이런 부분에 대해서 중요하게 다루지 않았지만 최근에 임베디드 가상화가 활발히 이루어 지면서 가상화 환경에서도 연성 실시간 프로그램을 실행시켜야 하는 필요가 있다. 본 논문에서는 멀티미디어 소프트웨어를 통해서 기존 시스템의 오차와 제안한 시스템의 오차를 비교 측정하여 성능 향상을 보였으며, 임베디드 가상화 환경에서도 평균 400us 이내의 오차로 타이머 성능을 낼 수 있음을 확인하였다.

참고문헌

[1] J, Nieh and Monica S. Lam, "The Design, implementation and Evaluation of SMART : A Scheduler for Multimedia Applications," Proceedings of 16th ACM Symposium on Operating Systems Principles, St Malo, France, October, 1997

[2] S.Childs and D.Ingram, “The Linux-SRT integrated multimedia operating Syetem : bringing QoS to desktop”, Proceedings of 7th IEEE Real-Time Technology and Application Symposium, 2001

[3] T.Gleixner and D.Niehaus, “Hrtimer and Beyond : Transforming the Linux Time Subsystems”, Proceedings of the Linux Symposium Ottawa, Ontario Canada, July 2006

[4] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. Proc. VEE, 2008.

[5] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee. Task-aware virtual machine scheduling for I/O performance. VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2009.

[6] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. Consumer Communications and Networking Conference, 2008.

[7] Kenneth J. Duda, David R. Cheriton. Borrowed virtual time (BVT) scheduling: supporting latency sensitive threads in a general-purpose scheduler. Proceedings of the seventeenth ACM symposium on Operating systems principles, pp.261-276, 1999.

[8] 박미리, 홍철호, 유시환, 유혁. VIT 게스트 운영체제의 실시간성 지원을 위한 타이머 하이퍼콜. 정보과학회논문지: 시스템 및 이론 제 37 권 제 1 호. 2010.02