VIT: 게스트 운영체제의 실시간성 지원을 위한 타이머 하이퍼콜

(VIT: A Timer Hypercall to Support Real-time of Guest Operating Systems)

박 미 리 † 홍 철 호 † 유 시 환 † 유 혁 ††
(Miri Park) (Cheol-ho Hong) (See-hwan Yoo) (Chuck Yoo)

요 약 가상화 환경에서는 여러 개의 게스트 운영체제가 자원을 공유하고 있는데 특히 CPU는 시간 분할 방식에 의해 분배 된다. 따라서 각 가상 머신은 모든 물리 시간을 점유하지 못하고, 이는 CPU를 점유하지 못한 시간에 동작해야 하는 태스크 실행이 보장되지 못하는 결과를 야기시킨다. 이와 같은 응답성의 저하는 기존의 서버 가상화에서 치명적인 문제가 되지 않으나 임베디드 시스템에서 동작하는 실시간 태스크 측면에서는 중요하게 다루어 져야 하는 요구사항이 된다. 본 논문에서 우리는 실시간성과 관련된 타이머를 가상 머신 모니터에 등록할 수 있는 하이퍼콜을 제안한다. 이 하이퍼콜을 통하여 등록된 타이머는 만료된 시점에 해당 가상 머신이 실행되게 한다. 따라서 실시간성을 요하는 태스크들이 의도된 시간에 실행 가능하게 하며 다른 가상 머신의 기아 현상을 완화시켜 준다. 이어지는 실험에서는 Xen-Arm에 이를 구현하여 게스트 운영체제인 리눅스에서의 실시간성을 평가한다. 특히 테스트 응용과 Mplayer의 응답성 및 초당 프레임 수를 측정하여 한 개의 게스트 운영체제만이 동작하는 환경에서와 비슷한 실시간성이 지원될 수 있음을 보인다.

키워드 : 가상화, 가상 머신 모니터, 하이퍼콜, 실시간성, 임베디드 소프트웨어

Abstract Guest operating systems running over the virtual machines share a variety of resources. Since CPU is allocated in a time division manner it consequently leads them to having the unknown physical time. It is not regarded as a serious problem in the server virtualization fields. However, it becomes critical in embedded systems because it prevents guest OS from executing real time tasks when it does not occupy CPU. In this paper we propose a hypercall to register a timer service to notify the timer request related real time. It enables hypervisor to schedule a virtual machine which has real time tasks to execute, and allows guest OS to take CPU on time to support real time. The following experiment shows its implementation on Xen-Arm and para-virtualized Linux. We also analyze the real time performance with response time of test application and frames per second of Mplayer.

Key words : Virtualization, Virtual Machine Monitor, Hypercall, Real Time, Embedded Software

† 학생회원 : 고려대학교 컴퓨터전파통신공학과
mrpark@os.korea.ac.kr
chhong@os.korea.ac.kr
shyoo@os.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터전파통신공학과 교수
hxy@os.korea.ac.kr
논문접수 : 2009년 10월 12일
심사완료 : 2009년 11월 3일

정보과학회논문지: 시스템 및 이론 제37권 제1호(2010.2)

1. 서 론

운영체제 가상화에서 가장 중요한 요건 중 하나는 가상화를 하지 않은 상태에서 보여주던 성능을 가상 머신 상에서도 그대로 유지하는 것이다. 운영체제 가상화가 서버 분야에서 이용되게 된 동기 중 하나는 CPU 사용률이 10-20%정도로 낮은 서버들을 하나의 물리 머신에서 동작시킴으로써 자원을 절약하는 데에 있다. 그리고 이 때 기존의 성능이 크게 손상되지 않아야 상용 수준으로 활용할 수 있으므로 성능의 보장은 주요 조건이 된다.

임베디드 장치 위에서 동작하는 소프트웨어는 일반적으로 서버에서 동작하는 소프트웨어가 보여주어야 하는

성능 조건 외에도 다른 제약 조건을 가진다. 특히 서버 환경에 비해 제한적인 자원을 사용해야 하면서도 그 동작성에 있어 실시간성을 지원해야 한다.

기존의 가상화 기술에서 실시간성은 고려되지 않았다. 왜냐하면 서버 가상화에서 자원을 배분함에 있어 가장 중요한 목표 중 하나는 자원의 활용률 측면에서의 "공평성(fairness)"이기 때문이다. 즉, 여러 개의 VM에 공평한 자원을 분배해 주어야 각 VM에서 동작하는 서버가 물리 자원을 골고루 사용할 수 있다. 즉, 서버 가상화에서는 얼마만큼의 CPU점유율을 가질 수 있느냐만 고려할 뿐 언제 CPU를 점유할 수 있는가는 반영하지 않는다.

최근 몇 년간 [1-3] 등에서 임베디드 가상화 기술에 대한 연구가 이루어져왔으나 실시간성에 대한 연구는 [4,5] 등에서만 논의되었다.

[4]는 우선 순위를 이용한 스케줄링 기법을 통해 임베디드 가상화 환경에서 실시간성 보장을 하게 하는 방안을 제안하였다. 이 연구는 각 VM에서 동일한 성격의 태스크들만이 동작한다고 가정하였다. 즉, 시간 기반 실시간 작업, 이벤트 기반 실시간 작업, 그리고 비실시간 작업을 하는 태스크들이 각각의 VM에 탑재된 환경으로 제한 시킨 것이다. 그리고 실시간 작업을 하는 VM에 높은 우선순위를 부여하여 우선순위 기반 스케줄링을 하도록 하였다. 그러나 이 연구는 실시간 VM에 우선순위를 높게 준다고만 밝혔을 뿐, 실시간성이 실제로 어떻게 보장되는지 보여주지 못했다. 특히나 특정 VM에 무조건적으로 높은 우선순위를 부여하는 것은 자칫 다른 VM의 기아(starvation) 현상을 초래할 수도 있다. 따라서 좀더 유연하며 정교한 방안이 필요할 것이다.

우리는 먼저 가상화 환경에서 왜 실시간성이 지원되지 않는지를 가상 머신의 CPU 배분 방식을 통해 분석하였다. 그리고 VIT라는 새로운 하이퍼콜을 통해 태스크가 수행되어야 할 시점을 알려주도록 하고 이를 스케줄링에 반영할 수 있도록 하였다. 또한 임베디드 기반 공개 가상 머신 모니터 Xen-Arm에 구현하여 게스트 운영체제의 실시간성을 평가하였다.

2. 실시간성 지원을 위한 가상 머신 스케줄러

실시간성 연구 분야에서는 언제 CPU를 점유해야 하는가 하는 문제를 흔히 해당 태스크에 대한 주기, 실행시간, 데드라인을 정의함으로써 해결하여 왔다. 이와 같은 정보를 바탕으로 태스크 사이의 의존성 및 인터럽트 등의 다른 자원의 상태를 분석하여, 실시간 태스크가 정의된 데드라인 이내에 수행될 수 있는지 예측하고 스케줄링 한다. RM(Rate Monotonic)이나 EDF(Earliest Deadline First)등의 실시간 스케줄링 알고리즘[6]이 이 분야에서 제안되었다.

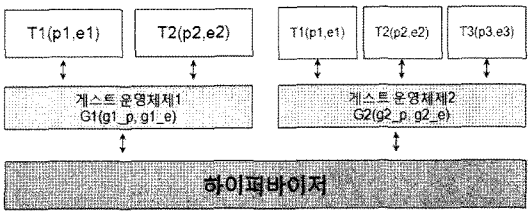


그림 1 실시간성 보장을 위한 계층적 스케줄링

그러므로 그림 1에서와 같이 각 VM에서 동작하는 태스크의 정보를 미리 알 수만 있다면 이를 바탕으로 각 VM에서 필요로 하는 자원 요구량을 파악할 수 있다. 그리고 하이퍼바이저는 이 요구량을 이용하여 실시간 스케줄러로 VM을 스케줄링 해주고, 게스트 운영체제는 다시 자신의 태스크를 스케줄링 하여 실시간성을 보장할 수 있다. 하지만 실시간을 보장해 주어야 할 VM에서 동작하는 태스크들의 정보를 미리 알지 못한다면 이와 같은 방식은 적절히 활용되기가 어렵다. 즉, 실시간 스케줄링 기법의 특성상 실시간성 보장을 위한 각 태스크의 주기, 실행시간, 데드라인 등을 미리 알아야 하는데 이를 모른다면 해당 VM이 언제 자원을 필요로 하는지 알 수가 없다. 따라서 미리 정보를 알지 못하는 상황에서 실시간성을 지원하는 방식이 필요할 것이다.

우리는 일반적으로 많은 소프트웨어들이 주기적으로 실행해야 하는 일들을 위해 타이머 서비스를 활용하며, 이는 실시간성을 지원하는 데에 밀접한 역할을 한다는 점에 주목했다. 예를 들어, 비디오 플레이를 하는 경우 특정 시간 간격 마다 버퍼에 있는 내용을 디스플레이 하기 위해 타이머를 설정하고, 이 타이머가 만료되었을 때마다 실행되는 루틴이 프레임을 디스플레이 한다. 만약 타이머 서비스가 제대로 동작하지 못해서 특정 시간에 디스플레이 루틴이 실행되지 않는다면, 사용자는 끊긴 화면을 보게 될 것이고 이는 곧 실시간성 보장의 실패가 된다. 따라서 타이머가 만료 되었을 때 해당 루틴이 실행되는 것이 중요하다.

그런데 CPU 점유율만을 고려한 기존의 VM 스케줄링 기법에서는 타이머 만료되는 시점에 수행이 보장되지 않는다.

그림 2는 공평한 자원 분배를 위해 각 VM을 교대로 실행시킬 때의 CPU 점유 상황을 보여준다. 태스크1은 ① 시점에서 50ms 동안 타이머를 설정하였지만 실제

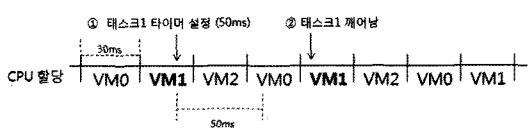


그림 2 기존 VM 스케줄링의 문제

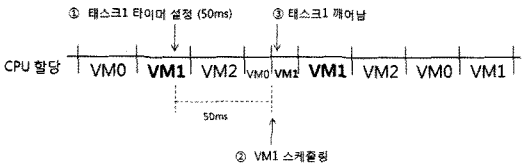


그림 3 타이머 정보를 이용한 VM 스케줄링

50ms가 지난 시점에서는 VM0가 동작하고 있다. 따라서 VM1의 커널은 이 태스크를 실행시킬 수 없고, 이후에 스케줄링 되는 ② 시점에서 가서야 타이머가 만료되었음을 알게 된다. 즉, 원래 의도했던 타이머 설정 시간은 50ms였지만 실제 타이머 만료 시점은 그림 2에서와 같이 그 이상이 될 수도 있다.

하지만 만약 그림 3에서와 같이 ② 시점에서 하이퍼바이저(hypervisor)가 타이머가 만료되는 태스크가 속한 VM을 스케줄링 해준다면, 태스크1은 원하는 시간에 수행될 수 있을 것이다. 즉, 하이퍼바이저가 VM1의 타이머 설정 내용을 알고 있다면 해당 시간에 VM1이 실행되도록 스케줄링 해줄 수 있다.

우리는 해결 방안으로 타이머 서비스를 가상 머신 모니터에 등록할 수 있는 VIT를 제안한다. VIT는 하이퍼바이저가 VM에게 제공하는 하이퍼콜(hypercall)로 실시간성이 필요한 타이머 요청을 하이퍼바이저에게 알리고 만료되는 시점에 스케줄링을 보장해 주는 역할을 한다. 즉, 실시간(real time) 지원을 위한 타이머인 경우 만료되었을 때 바로 해당 VM에게 수행을 넘겨주게 된다. 따라서 미리 실시간 태스크에 대한 정보를 가지고 있지 않아도 하이퍼콜을 호출함으로써 VM에서 동작하는 소프트웨어의 실시간성을 지원할 수 있게 된다. 하이퍼콜을 이용해야 하는 방법의 제약상 이는 Xen 등의 반가상화(para-virtualization)에서 활용될 수 있다. 반가상화는 VM에서 동작하는 게스트 운영체제를 수정하여 성능 면에서 이득을 얻는 방식으로 우리가 제시하는 VIT 역시 게스트 운영체제의 수정이 불가피하다. 왜냐하면 하이퍼콜을 이용하여 특정 타이머 설정을 등록하여야 하기 때문이다.

전가상화(full virtualization)에서는 하이퍼콜을 호출하는 대신 타이머 서비스를 등록하는 소프트웨어 인터럽트를 감지함으로써 게스트 운영체제의 타이머 정보를 알 수 있다. 그러나 이 경우 어떤 타이머가 실시간성 지원을 위한 것인지 판단하기가 어렵다. 따라서 반가상화에서 하이퍼콜을 제공하는 방식이 실시간성 지원을 명시적으로 요청하게 하여 선택적인 응답성 향상을 가능하게 한다.

Xen은 이미 VM에게 타이머 설정을 위한 하이퍼콜을 제공하고 있다. 그러나 이 하이퍼콜은 VM에게 해당 타이머가 만료되었음을 알리는 역할만 할 뿐, 그 순간에 수행됨을 보장하지 않는다. VM의 타이머가 만료되었다 하더라도 다른 VM이 동작하고 있고 이 VM의 우선 순위가 더 높다면 타이머가 만료된 VM은 수행되지 않는다. 그러므로 여전히 실시간 태스크가 실행될 수 있는 여지는 없다. 그러나 VIT로 설정한 타이머는 만료되는 순간에 하이퍼바이저가 해당 VM을 스케줄링 시켜 주므로 실시간 태스크가 실행될 수 있다.

앞서 설명한 바와 같이 만약 VM에서 실행되는 실시간 태스크의 정보를 미리 알고 있다면 VM들을 실시간 스케줄링을 해주어 실시간성을 보장할 수 있다. 그러나 우리가 가정한 환경은 이와 같은 정보가 알려지지 않아 실시간 스케줄링이 어려운 경우이다. 대신에 실시간과 관련된 타이머 서비스를 하이퍼바이저에게 요청하고 하이퍼바이저는 해당 시간에 실행을 요하는 VM을 스케줄링 해준다.

3. VIT 디자인

3.1 하이퍼콜의 추가

기존의 시스템은 응용 프로그램이 사용하는 타이머가 VM내부에서만 관리되어 하이퍼바이저에는 가려져 있다. 그러므로 타이머가 만료되었음을 인지하고 해당 응용 프로그램을 실행시키기 위해서는 그 VM이 스케줄링 된 상태여야 한다.

우리가 제안하는 시스템은 그림 4와 같이 VIT를 통해 하이퍼바이저가 타이머 정보를 알 수 있다. 또한 이 정보를 바탕으로 해당 VM을 실행시켜줌으로써 실시간성을 지원하게 된다.

하이퍼바이저는 중요한 타이머 설정을 위해 하이퍼콜인 VIT를 제공한다. 이 하이퍼콜을 통해서 VM위에서 동작하는 소프트웨어는 자신이 동작하고 있는 VM이 특정 시간 T_{sleep} 이후에 수행되도록 할 수 있다.

hypercall_request_VIT(T_{sleep})

하이퍼바이저는 하이퍼콜을 통해 받은 정보를 바탕으

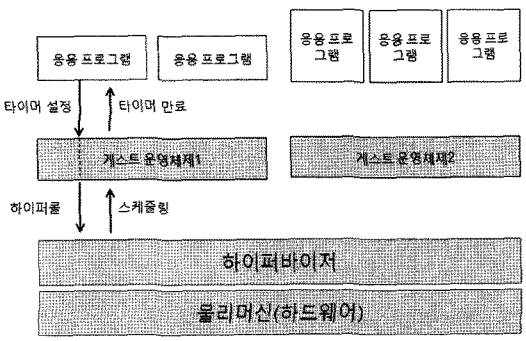


그림 4 VIT가 추가된 하이퍼바이저

로 타어머가 만료될 시점을 계산하고 이 값을 관리한다. 타이머가 만료되는 시점은 다음과 같다.

$$Texpire = T_{sleep} + T_{current}$$

시스템의 시간 관리를 위해 특정 시간마다 하드웨어에서 틱이 발생되고 이 틱 인터럽트 핸들러는 하이퍼바이저에 존재한다. 따라서 하이퍼바이저는 매 틱마다 만료되는 타이머가 있는지 확인할 수 있다. 타이머가 만료되는 조건은 다음과 같다.

$$T_{expire} <= T_{current}$$

하이퍼바이저는 이와 같은 조건을 검사하여 만료된 타이머가 있다면 해당 VM이 스케줄링 될 수 있도록 한다. 또한 일정 시간 수행이 된 후에는 원래의 스케줄링 방식으로 되돌아가 다른 VM이 스케줄링 될 수 있도록 한다.

만약 두개 이상의 VM에서 요청한 VIT가 만료된 경우에는 먼저 요청한 VM이 먼저 스케줄링된다. 이는 우리가 가정한 시스템이 가상 머신에 우선순위를 정의하지 않았고 큐를 사용하는 타이머 관리 구현상 먼저 요청된 타이머의 만료 여부가 먼저 검사되기 때문이다.

VIT를 이용하여 스케줄링하는 경우 자칫 특정 VM에게만 CPU자원이 계속하여 할당될 수 있다. 그러나 우리의 목표는 스케줄링 시점을 변경시켜 응답성을 지원하게 하는 것일 뿐, 점유율 측면에서의 공평성은 여전히 지켜져야 한다. 따라서 타이머가 만료되어 실행되는 VM은 그 이후에 스케줄링이 제한되어 전체적인 공평성이 지원될 수 있도록 해야 할 것이다. 우리는 VIT를 5장에서 자세히 설명할 BVT에 구현하여 이와 같은 문제가 발생하지 않도록 하였다.

3.2 응용

미디어 플레이어의 경우 일반적으로 주기적인 동작을 통해 음악 혹은 동영상을 재생한다. 그리고 주기적인 동작이 이루어 지기 위해서는 일정 시간 동안 sleep 하게 된다.

예를 들어 미디어 플레이어의 주요 루틴은 파일의 끝이 나올 때까지 루프(loop)를 실행하며 음악을 재생하게 구현되어 있다. 이때 sleep함수는 커널이 관리하는 타이머 서비스를 호출하여 타이머가 만료되기를 기다리게 한다.

또한 동영상을 재생해야 하는 경우 디스플레이 되는 프레임과 오디오 재생의 동기화를 위해서도 타이머 서비스는 이용된다. 보통 비디오 프레임이 디코딩 되고 나서 타임 스탬프를 오디오 타임 스탬프와 비교하게 된다. 이때 비디오 타임 스탬프가 더 작은 값을 가지게 되면 바로 디스플레이가 되지만 더 큰 값을 가지게 된다면 그 차이만큼 sleep을 하여 동기화를 하게 된다.

따라서 이때 이용하는 타이머 서비스를 VIT가 지원하게 하여 응답성을 높여주고 이로 인해 더 나은 실시간성을 보여줄 수 있을 것이다.

4. 실험 환경 및 구현

성능 평가를 위한 환경을 만들기 위해 Xen-Arm을 수정하여 하이퍼콜을 추가하였다. Xen-Arm은 Xen[7] 공식 커뮤니티에서 Arm 프로세서를 위해 릴리즈[8] 하였으며 현재 두번째 버전까지 나와 있다. 게스트 운영체제로는 리눅스와 실시간 커널인 uC/OS-II가 릴리즈 되었으며 본 논문에서는 리눅스를 대상으로 실험을 수행하였다. 하드웨어 자원 및 실험 환경에 대한 요약은 표1에 정리하였다.

표 1 자원 및 실험 환경

Hardware	
Freescale i.MX21 (ARM926EJ-S)	
Software	
Hypervisor	Xen-Arm (based on x86 Xen-3.0.2-2 plus security feature
Guest OS	(para-virtualized) Linux 2.6.21.1

4.1 측정치(Metric)

실시간성 보장은 정형적인 기법이나 실험을 통하여 평가될 수 있다. [9]은 실험적인 방법 중 하나로 응답성(latency)을 측정하는 실험을 통해 리눅스에서 실시간성을 분석하였다. 우리도 세가지 측정치(metric)를 정의하여 실험적으로 실시간성을 평가하였다.

첫번째는 $T_{latency}$(sleep)으로 어떤 태스크가 sleep을 하며 요청한 타이머의 응답성을 수치화한다. 즉, 태스크가 sleep한 시점부터 깨어난 시점까지 지난 시간을 $T_{latency}$(sleep)으로 정의 한다.

$T_{latency}$(sleep)이 요청한 값에 가까울수록 실시간성이 지켜질 가능성이 더 커진다. 그러나 엄밀한 의미에서 지원해야 하는 실시간성은 sleep에서 깨어나 특정 작업이 끝난 시점까지를 보장함을 말한다. 두 번째 측정치로 이를 수치화하기 위해 그림 5에서와 같이 sleep에서 깨어나 일정 작업을 마칠 때까지 걸리는 시간을 $T_{latency}$(work)로 정의한다.

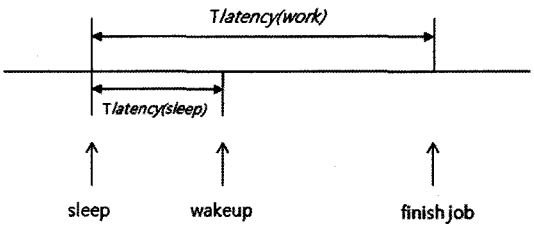


그림 5 $T_{latency}$ 정의

마지막 측정치는 Mplayer를 이용한 응용 실험에서 실시간성을 보이기 위하여 고려되었다. 5장에서 설명한 멀티미디어 플레이어, 특히 비디오 플레이어의 경우 초당 프레임이 미리 정해져 있기 때문에 실제 재생된 초당 프레임 수를 보이는 것으로 실시간성이 얼마나 지원되는지를 나타낼 수 있을 것이다.

4.2 Xen-Arm 스케줄러 수정

Xen-Arm의 최신 버전은 기본 스케줄러로 BVT (Borrowed-Virtual-Time)[10] 스케줄러를 사용하고 있다. BVT 스케줄러는 스케줄링 단위가 각자의 가상 시간을 갖는다는 개념을 이용하여 미리 정해진 가중치에 따른 자원을 공평하게 분배한다. 여기에 warp이라는 기능을 이용하여, 특정 스케줄링 단위가 warp되는 순간에 자신이 사용하게 되는 자원을 미리 당겨 쓰게 한다. 그리고 이로 인해 그 순간에 응답시간을 높일 수 있다. 또한 warp이 해제되면서 당겨 썼던 시간만큼 다른 스케줄링 단위가 우선적으로 CPU를 분배 받게 되므로 공평한 분배가 가능하다. 따라서 평소에는 공평하게 CPU자원을 분배 받다가 VIT가 만료되어 특정 도메인이 스케줄링 되어야 하는 경우에 warp을 활성화하는 방식을 사용하여 우리의 아이디어를 Xen-Arm에 구현하였다.

5. 실험 결과

5.1 테스트 응용

성능 평가의 첫 번째 응용으로 리눅스에서 동작하는 간단한 테스트 어플리케이션을 만들었다. 이 어플리케이션은 30ms동안 sleep을 하고 난 후 깨어나 CPU를 소모하는 행렬 연산 수행을 반복하는데 평균 32.7%의 CPU점유율을 보인다.

표 2에서 보는 것과 같이 도메인이 1개일 때 평균 40ms였던 $T_{latency}$(sleep)은 도메인 2개가 동작하면서 80.5ms으로 늘어 난다. 이는 앞서 설명한 것과 같이 도메인이 CPU를 공평하게 분배(fair share) 받고 있으므로 타이머가 만료되어도 해당 태스크가 스케줄링 될 수 없기 때문이다. 그러나 VIT를 이용한 경우 도메인이 깨어날 시간에 스케줄링시켜 줌으로서 $T_{latency}$(sleep)은 40ms으로 줄어 들게 되며, 이 값은 도메인 한 개만 동작할 때의 수준과 같은 값이다.

표 2 테스트 응용의 응답시간

30ms sleep에 대한 평균 $T_{latency}$(sleep) (ms)	
도메인 1개	40
도메인 2개	80.5
도메인 2개 + VIT	40
30ms sleep에 대한 평균 $T_{latency}$(sleep) (ms)	
도메인 1개	120
도메인 2개	211.4
도메인 2개 + VIT	120.8

테스트 응용은 sleep 후에 깨어나 CPU를 소모하는 행렬 연산을 수행한다. 이에 대한 응답시간은 도메인이 1개일 때 평균적으로 120.8ms가 걸린다. 이 값은 두 개의 도메인일 때 211.4ms까지 길어지게 되나 VIT를 이용하여 기존 수준의 성능을 확보할 수 있었다.

우리의 또 다른 관심은 이와 같은 방식이 정적 우선순위 스케줄링 방식과 어떤 차이점을 가지느냐는 것이다. 앞서 지적한 바와 같이 실시간 태스크가 있는 VM에 무조건적으로 높은 우선순위를 준 경우에도 똑같이 응답성은 나아질 수 있기 때문이다. 우리는 두 개의 도메인 A, B가 있을 때 A는 실시간성을 지원하는 도메인, B는 보통 도메인이라고 가정하였다. 그리고 도메인 B에서 테스트 응용을 동작시켜 전체 수행시간을 측정하였다. 표 3은 서로 다른 스케줄링 방식에 따라 도메인 A에서 실시간성을 지원 받을 때 비실시간 도메인 B의 응답시간을 보여준다.

표 3 스케줄 방식에 따른 비실시간 도메인의 응답시간

도메인 B의 테스트 응용 평균 작업 시간 (초)	
도메인 1개	6.04
도메인 2개 / BVT	10.06
도메인 2개 / 정적우선 순위	13.43
도메인 2개 / VIT를 적용한 BVT	10.21

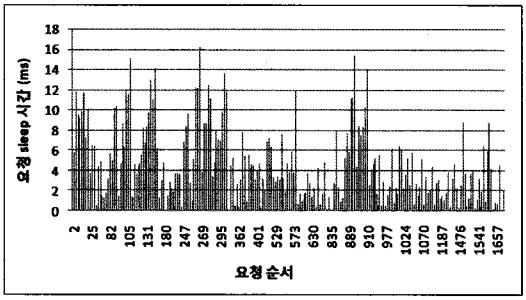
수정하지 않은 BVT(기본 설정)는 라운드 로빈 방식과 같이 두 개의 도메인을 교대로 실행시킨다. 이때 한 도메인에서 측정한 응답시간은 10.06초로 CPU를 독점하여 사용하는 도메인 1개일 때에 비해 늘어난 수치를 보인다. 하지만 정적 우선 순위 방식을 쓰면 도메인 A의 우선순위가 언제나 높으므로 도메인 B는 CPU를 자주 할당 받지 못한다. 그리고 그 결과로 13.43초의 응답성을 보인다. 그러나 VIT를 이용한 스케줄링의 경우 10.21초의 응답시간을 기록하는데 이는 기본 설정값을 이용하는 BVT 방식을 사용할 때와 비슷한 수준의 값이다. 이것은 타이머가 만료되는 순간에만 순간적으로 도메인 A가 CPU를 점유하며 그 외에는 도메인 B가 스케줄링 되어 기아현상을 막기 때문이다. 반면에 정적 우선 순위 방식은 계속 CPU를 점유하고 있기 때문에 실시간성을 지원받지 않는 도메인의 기아 현상을 초래할 수 있다.

5.2 MPlayer

MPlayer는 타이머를 이용하는 미디어 플레이어로 리눅스에서 동작하는 공개 프로그램이다. 우리는 VIT를 활용하여 MPlayer의 타이머 요청을 처리해 줄 수 있도록 하였다. 또한 테스트를 위하여 avi파일을 사용하였으며, avi 파일의 인코딩 속성은 표 4와 같다.

표 4 샘플 avi 파일 정보

코덱 종류	Xvid (MPEG-4) (XVID)
이미지 크기	320 × 240 (1.33 : 1)
평균 비트 전송률	401.33 KBits / Sec
초당 프레임수	20.00 Frames / Sec
총 프레임수	1,268
재생 시간	00:01:03 (63 Sec)

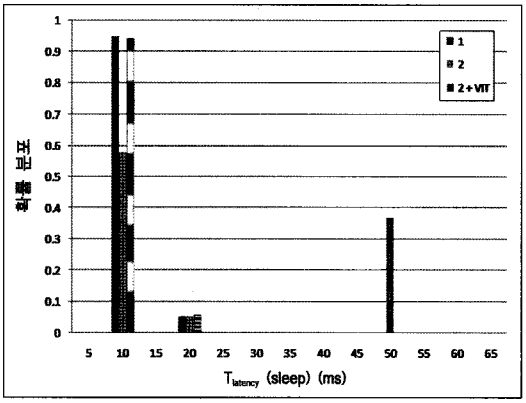


그래프 1 Mplayer의 타이머 요청

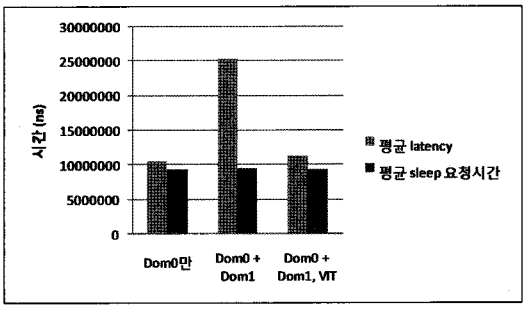
그래프 1은 MPlayer가 요청하는 sleep 요청 시간을 나타낸 것으로 Mplayer가 실행되는 시점부터 각 sleep을 요청하는 시점에 얼마만큼의 타이머 서비스를 요청하는지 보여준다.

Mplayer는 최소 0.004ms부터 최대 13.679ms까지 평균 9.271ms의 sleep을 요청하였다. 그래프 2는 이 요청에 대한 응답성, 즉 $T_{latency}$(sleep)의 분포를 나타낸 것이다. 도메인의 개수가 1개, 2개인 경우와 도메인이 2개이면서 VIT를 쓴 경우 세가지 케이스에 대한 실험을 수행하였다.

도메인이 2개가 되면서 $T_{latency}$(sleep)중 약 40%가 50ms의 응답성을 보이게 된다. 이는 앞서 실험한 테스트 응용에서와 마찬가지로 CPU 할당을 제한적으로 받게 되어 해당 시간에 도메인이 스케줄링 되지 않기 때



그래프 2 Mplayer 타이머 서비스의 응답시간 분포



그래프 3 평균 타이머 요청 시간 및 응답시간

문이다. 그러나 VIT를 사용한 케이스에서는 도메인이 1개인 상황과 비슷한 응답성 분포를 보이며 평균값은 약 11.20ms가 된다. 그래프 3은 도메인 개수와 VIT사용 케이스에 대한 평균 sleep요청시간과 $T_{latency}$(sleep)을 비교하여 보여준다.

표 5는 실험 결과를 정량적으로 보여준다. 평균 $T_{latency}$(sleep)은 앞서 설명한 바와 같이 도메인 2개가 동작하면서 평균 25.383ms까지 길어지며 VIT를 이용하였을 때 11.202ms가 되었다.

Mplayer가 재생하는 초당 프레임 수는 실시간성을 보여주는 주요 척도이다. 이 값은 도메인 2개시에 11.3 fps까지 떨어졌으나 VIT를 이용하면서 19.4fps가 된다. 즉, 기존의 실시간성이 도메인이 늘어나면서 고려되지 않았으나 VIT를 이용하면서 지원된다고 볼 수 있다.

표 5 Mplayer의 응답시간 및 fps

평균 sleep 요청 시간 (ms)	
도메인 1개	9.271
도메인 2개	9
도메인 2개 + VIT	9.283
평균 $T_{latency}$(sleep) (ms)	
도메인 1개	10.488
도메인 2개	25
도메인 2개 + VIT	11.202
초당 재생 프레임수 (fps)	
도메인 1개	19.4
도메인 2개	11
도메인 2개 + VIT	19.4

6. 관련 연구

Cherkasova, Gupta, Vahdat은 [11]을 통해 Xen에 구현된 세 가지 VM 스케줄러의 성능을 비교 하였다. Xen은 분리된 디바이스 모델(Isolated Device Driver model)을 따르며, Dom0가 디바이스 드라이버를 가지고 있다. 저자들은 BVT, SEDF, Credit 스케줄러가 각각 어떤 특징을 가지는지 설명하고 스케줄링 파라미터 변화가 성능에 어떤 영향을 주는지를 실험을 통해 보여

주었다.

[12]은 Xen에서 동작하는 게스트 운영체제의 네트워크 성능이 Dom0의 스케줄링에 영향을 받는다는 사실을 지적하였다. 패킷을 처리하기 위해서는 우선 디바이스 드라이버가 존재하는 Dom0가 스케줄링되어야 하며, 그 이후에도 Dom0와 DomU가 통신을 하여야 하는 등의 지연이 존재한다. Govindan, Nath, Das는 각 도메인별로 패킷을 기록하여 그 횟수가 많은 도메인이 우선적으로 스케줄링 될 수 있도록 하여 응답시간을 높여 주었다.

Ongaro, Cox, Rixner 역시 [13]에서 VM스케줄링이 IO성능에 영향을 준다는 점을 분석하고, Credit 스케줄러를 수정하여 IO성능을 높일 수 있는 방안을 보였다. 저자들은 이벤트 채널 패치, 런큐의 정렬, 선점 횟수 최소화 등을 구현하여 각 수정사항이 IO성능에 어떤 영향을 미치는지를 설명했다. 또한 응답성이 중요한 태스크의 성능은 CPU를 많이 사용하는 태스크와 다른 도메인에 있을 때 좋아짐을 실험을 통해 보여주었다.

[14]에서는 태스크의 동작을 gray-box 기법을 통해 수집한 뒤 이를 스케줄링에 반영하는 방식을 제안하였다. 저자들은 IO-bound 태스크들을 partial boosting 해줌으로써 IO 성능을 향상시켜 주면서도 CPU 배분의 공정성을 유지할 수 있음을 보였다.

[5]는 임베디드 시스템에서 실시간 태스크들이 여러 게스트 운영체제에 걸쳐 있을 수 있음을 지적하고, 태스크 레벨의 스케줄링 방식을 제안하였다. 각 도메인별로 각기 다른 우선순위를 가지는 태스크들에게 글로벌 우선 순위를 부여 함으로써 실시간 태스크가 어떤 도메인에 존재 하더라도 실시간성이 보장되도록 하였다.

본 논문은 기존 연구의 방향과는 달리 스케줄링 방식만으로는 게스트 운영체제의 실시간성을 지원하기가 힘들다는 점을 지적하였다. 태스크들이 타이머 서비스를 요청하고 sleep하더라도 깨어나는 시점에 다른 도메인이 CPU를 점유하고 있어 해당 태스크가 실행되지 못할 수 있기 때문이다. 따라서 태스크 레벨의 타이머를 하이퍼바이저가 알고 있게 함으로써 깨어나는 즉시 실행을 지원하고, 태스크의 응답성을 향상시켜 주었다.

7. 결 론

실시간성을 요하는 소프트웨어는 주로 타이머를 이용하여 주기적인 작업을 수행한다. 그러나 기존의 가상화 환경은 이를 고려하지 않아 타이머가 만료된 시점에 해당 태스크가 수행되지 않을 수 있다. 이는 응답성의 저하로 이어져 실시간성을 떨어뜨리는 요인이 된다. 우리는 이러한 타이머 요청을 하이퍼콜 형태로 하이퍼바이저에게 전달하여 타이머가 만료되는 시점에 해당 VM을 스케줄링하게 하였다. 이 결과 타이머 요청에 대한 응답성이 향상되었으며 이는 멀티미디어 플레이어가 재생하는 초당 프레임 수의 향상으로 이어졌다.

이와 같은 방식은 VM의 태스크 정보를 미리 알지 못해 자원의 실시간 분배가 어려운 경우에 활용될 수 있으며, 실제 Xen-Arm에서 동작하는 게스트 운영체제인 리눅스에 적용하여 그 결과를 확인할 수 있었다.

참 고 문 헌

[1] D. R. Ferstay. Fast secure virtualization for the arm platform. Master's thesis, University of British Columbia, 2006.

[2] Trango: secured virtualization on ARM. Trango. http://www.trango-vp.com.

[3] VirtualLogix Real-Time Virtualization and VLX. VirtualLogix. http://www.osware.com.

[4] R. Kaiser. Alternatives for Scheduling Virtual Machines in Real-Time Embedded Systems. 1st Workshop on Isolation and Integration in Embedded Systems, 2008.

[5] Yuki Kinebuchi, Midori Sugaya, Shuichi Oikawa, Tatsuo Nakajima. Task Grain Scheduling for Hypervisor-Based Embedded System. *HPCC*, pp.190-197, 2008.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM*, vol.20, no.1, pp.46-61, 1973.

[7] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew arfield. Xen and the art of virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[8] 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.

[9] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, Jonathan Walpole. A Measurement-Based Analysis of the Real-Time Performance of Linux. *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2002.

[10] 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.

[11] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three CPU schedulers in Xen. SIGMETRICS Perform. Eval. Rev., 35(2):42-51, 2007.

[12] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms. *Proc. VEE*, 2007.

[13] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. *Proc. VEE*, 2008.
[14] 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.

박 미 리
2006년 2월 고려대학교 컴퓨터학과 학사 2006년~2007년 모토로라 코리아. 2008년~현재 고려대학교 컴퓨터전파통신공학과 석사과정. 관심분야는 가상화, 임베디드 시스템

홍 철 호
2001년 2월 고려대학교 컴퓨터학과 학사 2003년 2월 고려대학교 컴퓨터학과 석사 2007년~현재 고려대학교 컴퓨터전파통신공학과 박사과정. 관심분야는 가상 머신 플랫폼, 멀티 코어 운영체제

유 시 환
2002년 2월 고려대학교 컴퓨터학과 학사 2004년 2월 고려대학교 컴퓨터학과 석사 2004년~현재 고려대학교 컴퓨터전파통신공학과 박사과정. 관심분야는 Real-time OS, 내장형 시스템, 시스템 가상화

유 혁
1982년 2월 서울대학교 전자공학과 학사 1983년 2월 서울대학교 전자공학과 석사 1986년 8월 Master of Computer Science in University of Michigan. 1990년 8월 Ph.D of Computer Science in University of Michigan. 1990년~1995년 Sun Microsystems Lab.Researcher. 1995년~현재 고려대학교 컴퓨터학과 교수. 관심분야는 시스템 가상화, 멀티코어 플랫폼, 커널 네트워킹, 센서 네트워킹, 멀티미디어 스트리밍