L4 기반 가상화 기술을 적용한 임베디드 시스템의 성능평가

고원석*, 임성수*
*국민대학교 컴퓨터공학과
e-mail : kanariya@kookmin.ac.kr, sslim@kookmin.ac.kr

Performance Evaluation of Virtualization Technology based L4 applied to Embedded System

Wonsuk Ko*, Sung-Soo, Lim*
*Dept. of Computer Science, Kook-Min University

요 약

임베디드 시스템 분야가 발전하면서 하드웨어와 소프트웨어의 복잡도가 증가하고 또한 응용프로그램들의 요구사항이 점차 다양해짐에 따라 기존의 범용 운영체제에서 모든 작업을 처리하는 방법은 시스템의 신뢰성과 안정성을 떨어뜨리게 된다. 이를 해결하기 위해 응용프로그램 특성에 맞는 운영체제가 동작할 수 있는 다중 운영체제 플랫폼을 구성하는 것이 임베디드 시스템의 가상화 기술이며 시스템의 안정성과 신뢰성을 증진시키는 목적으로 사용한다. 본 논문은 가상화 기술을 임베디드 시스템에 적용하고 가상화 기술에 알맞은 디바이스 드라이버를 작성하고 이에 대한 성능을 측정, 평가한다.

1. 서론

임베디드 시스템 분야에서의 가상화 기술은 하나의 머신 위에 다양한 운영체제가 공존하여 동작하는 것을 말한다. 서로 다른 특성을 지닌 응용프로그램들(e.g. 실시간 요구사항을 만족해야 하는 응용프로그램, 보안성이 요구되는 응용프로그램 등) 이 하나의 범용 운영체제에서 동작하는 것은 시스템의 신뢰성을 유지하기에 어렵기 때문에 가상화 기술을 도입하여 응용프로그램의 특성에 맞은 운영체제로 응용프로그램을 동작하게 하여 시스템의 신뢰성을 높이는 것이 임베디드 시스템 분야의 가상화 기술이다. 이러한 임베디드 시스템에 가상화 기술을 적용하는 것에 대한 장점으로는 전체적인 시스템의 신뢰성을 증진시킬 수 있으며, 또한 새로운 플랫폼 개발 시 모든 코드를 이식하는 것만이 아닌 VMM 만을 이식하는 것만으로 그 위에 동작하는 응용프로그램들이 실행될 수 있다는 장점이 있다.

본 논문은 OKL4 의 가상화 기술을 적용한 임베디드 시스템을 구축하고 이에 대한 성능을 평가한다. 기존에 발표된 가상화된 리눅스와 네이티브 리눅스간의 성능평가의 연구결과에 보여지듯 L4 의 가상화 기술을 적용한 임베디드 시스템의 성능이 네이티브 리눅스와 비교하여 유사하거나 특정연산에서는 더욱 뛰어는 성능을 보였다[1],[2]. 그러나 이 때 작성된 디바이스 드라이버들은 유저 프로세스로 동작하는 게스트 OS 에서 직접 하드웨어로 접근하여 동작하는 방식으로 작성이 되었다. 이러한 기반에서 게스트 OS 들이 여러 개 존재하는 시스템의 경우 해당 게스트 OS 들이 모두 각각의 디바이스 드라이버를 가진다는 의미이며 이 경우 하나의 장치를 여러 게스트 OS 들이 동시에 접근할 수 있으며 이러한 경우 동시성 문제를 유발할 수 있다[3]. 이러한 문제들을 해결하기 위해 가상화 철학에 위배되지 않는 구조를 지니며 동시성 문제를 해결할 수 있는 구조로 디바이스 드라이버를 작성하여야 하며 언급한 구조를 지니는 디바이스 드라이버를 작성하기 위해 설계하는 방법을 제안하고 구현한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 대하여 기술하고, 3 장에서는 OKL4 의 가상화 기술을 적용한 디바이스 드라이버의 구조와 실제 구현에 대해 논한다. 4 장에서는 구현된 디바이스 드라이버와 네이티브 리눅스의 디바이스 드라이버와 비교한다. 5 장에서는 결론 및 향후 계획에 대해 기술한다.

2. 관련 연구

임베디드 시스템에서 가상화란 하나의 플랫폼에서 여러 게스트 OS 들이 공존할 수 있는 환경을 제공하기 위해 CPU 와 메모리, I/O 장치들을 가상화하는 것이라고 이전 장에 언급하였다. 가상화 기술을 활용한 여러 솔루션들이 존재하고 있으나 성능의 차이를 보이는 것은 각각의 CPU, 메모리, I/O 장치에 대한 가상화 방식이 다르기 때문이다[4]. L4 는 반 가상화를 사용하고 있으며 이와 유사한 연구로는 XenARM 이 진

행 중이다.

2.1 XenARM

데스크탑에서 사용되는 가상화 기술로 널리 알려진 Xen 을 개발한 오픈 소스 프로젝트에서 ARM 코어를 사용하는 임베디드 시스템에서 사용할 수 있게 XenARM 을 개발하였다. XenARM 에서 디바이스 드라이버를 위한 프레임워크를 위해 Split/Coordinate 모델을 제공한다. Split 디바이스 드라이버 모델은 XenX86 에서 사용되는 I/O 가상화 방식으로 실제 장치에 접근하는 Native/Backend 드라이버와 가상드라이버 역할을 하는 Frontend 드라이버로 구성된다. Backend 드라이버는 여러 Frontend 드라이버의 요청을 받아 이를 처리하는 기능을 한다. Coordinate Native 드라이버 모델은 XenARM 을 위해 도입된 개념으로 사용자가 수행하는 응용프로그램이 존재하는 게스트 OS 가 동작하는 도메인이 자체적으로 가지고 있는 Native 디바이스 드라이버에 시스템 자원의 사용을 요청하여 실제 하드웨어 접근한다.

XenARM 기반의 시스템에서의 성능 평가는 본 논문와 같은 LMBench 를 사용하여 측정하며 비교대상은 네이티브 리눅스이다. 네이티브 리눅스에 비해 약 80%정도의 성능을 보였음을 실험 결과로 알 수 있다 [5].

3. L4 드라이버의 구조 및 설계

본 논문에서는 가상화 철학을 유지하면서 여러 게스트 OS 들이 접근 시에 발생할 수 있는 동시성 문제까지 해결하는 디바이스 드라이버를 설계 및 구현하기 위하여 OKL4 에서 제공하는 디바이스 드라이버 프레임워크인 드라이버 버전 2(Driver V2)라는 구조를 사용하여 구현한다[3].

3.1 드라이버 버전 2 의 구조

그림 1은 기존 네이티브 리눅스의 디바이스 드라이버와는 상이한 구조를 가지고 있는 것을 보여준다. 먼저 게스트 OS 인 Wombat 이 디바이스 드라이버를 사용할 수 있도록 인터페이스를 제공해주며 서버-클라이언트 구조를 가지며 동시성 문제를 해결하고자 큐-관리 기법을 제공하는 vDevice 서버가 사용자 영역에 위치하며 이 vDevice 서버의 요청을 처리하기 위해 커널 공간에서 실제 디바이스를 제어하는 부분으로 나뉘어져 있다. 자세히 살펴보면 가상화된 게스트 OS 인 Wombat 에서 vDevice 서버에 접근하기 위한 vdev server 인터페이스를 제공하는 부분, 해당 인터페이스의 루틴이 정의된 vDevice 서버, vDevice 서버가 커널 영역의 디바이스 드라이버에 접근할 수 있게 제공되는 di 인터페이스, 실제 하드웨어를 접근하는 기본 디바이스 드라이버, 디바이스가 접근할 메모리 영역을 지정하는 dx 인터페이스, 총 다섯 개의 구성요소로 이루어져있다.

3.2 드라이버 버전 2 의 설계

드라이버 버전 2 로 드라이버를 작성하기 위해서는 dx 인터페이스에 접근하고자 하는 메모리 영역을 지정해야 하고 지정한 메모리 영역을 사용하는 연산을 기본 디바이스 드라이버에 작성하는 등의 각 연산들을 어떻게 어디에 구현할 것인가에 대해 설계를 해야 한다. 또한 같은 드라이버 버전 2 프레임워크로 작성하였다 할지라도 같은 연산 수행 시 하이퍼콜(Hypercall)의 횟수가 차이가 나면 그 횟수의 차이만큼 드라이버의 성능이 차이가 나기 때문이다. 그러므로

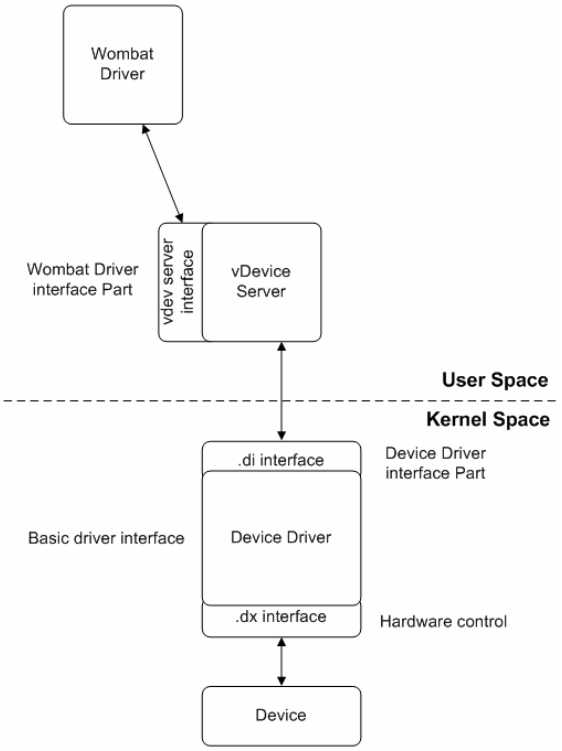

그림 1 드라이버 버전 2 의 구조도

연산을 기본 드라이버에 작성하고 이를 전부 게스트 OS 의 디바이스 드라이버에서 호출하는 방식보다 여러 번 호출될 연산들을 하나로 줄여서 동작시키는 것이 성능을 향상시킬 수 있다.

3.2.1 LCD 드라이버 작성을 위한 설계

임베디드 시스템에서 사용되는 LCD 디바이스 드라이버는 크게 세가지로 구성된다. 화면을 출력하기 위한 LCD 패널을 초기화 하는 부분, 뿌려지는 영상데이터를 제어하기 위해 해상도, 픽셀 클럭, RGB 나 YUV 등을 컬러 스페이스 등을 설정하기 위한 LCD 컨트롤러에 대한 설정 부분 그리고 LCD 패널에 화면을 뿌리기 위해 존재하는 프레임버퍼 부분의 세 가지로 구성되며 이 중 LCD 패널 초기화와 LCD 컨트롤러에 대한 설정들이 실제 하드웨어를 접근하는 부분으로 기본 디바이스 드라이버에서 수행되어야 할 부분이며 프레임버퍼에 대한 부분은 사용자 공간에서 처리하여 LCD 디바이스 드라이버를 작성한다.

4. 디바이스 드라이버의 구현

이번 장에서는 OKL4 에서 제공하는 드라이버 버전 2 프레임워크를 사용하여 타겟 시스템의 디바이스 드라이버를 구현에 대해 논한다. 3 장에서 언급한 LCD 디바이스 드라이버의 작성을 위한 설계를 따라 작성했다.

4.1 LCD 드라이버 구현

KMU 참조 플랫폼의 LCD 패널은 Intel®사의 2700G 그래픽가속기와 연동이 되어 있으며 현재 구현상 해당 기능은 사용하지 않기 때문에 Bypass 모드로 설정하여 사용한다. 이 부분도 2700G 의 내부 레지스터를 설정하기에 기본 디바이스 드라이버 내에 작성한다. SSP 를 사용하여 LCD 패널을 초기화하며 LCD 컨트롤러 레지스터를 설정하여 영상데이터에 대한 제어를 하며 2700G 를 bypass 모드로 설정하는 과정들을 기본 디바이스 드라이버로 작성한다. dx 인터페이스에서는 빈번히 접근할 레지스터에 대한 기술을 하였고 가장 처음에 한번만 수행되는 2700G 의 bypass 모드 설정과 LCD 패널에 대한 초기화는 OKL4 내부 함수로 처리하였다.

하드웨어 관련 초기화가 정상적으로 수행되면 LCD 패널에 영상데이터를 뿌리기 위한 프레임버퍼에 대한 디바이스 드라이버를 작성하며 이는 가상화된 리눅스인 Wombat 영역에서 작성한다. OKL4 에서 제공하는 가상 프레임버퍼를 사용하고 이 때 사용되는 DMA 할당도 OKL4 내부 함수를 통해 작성한다. 할당 받은 DMA 시작 주소를 LCD 컨트롤러에 넘겨주는 루틴을 하드웨어 처리부터 추가함으로 드라이버 버전 2 프레임워크를 사용한 LCD 디바이스 드라이버를 작성한다.

5. 실험

5.1 제약조건

OKL4 의 가상화 기술을 기반으로 구축한 시스템의 결과물이 KMU 보드에 탑재된 모든 장치가 아닌 LCD 디바이스 드라이버가 구현되었으므로 비교대상이 되는 네이티브 리눅스 또한 해당 디바이스들만을 활성화 시켜서 동작시켜 동일한 조건에서 실험한다.

5.2 실험 대상

OKL4 가상화 기술을 적용하여 구축한 시스템에 현재 드라이버 버전 2 로 작성한 드라이버는 LCD 이며 구축한 시스템의 전체성능을 LMbench 벤치마크 프로그램을 사용하여 측정한다.

5.3 실험 방법

네이티브 리눅스에 비해 OKL4 의 Wombat 리눅스의 성능이 떨어질 것으로 예상되며 그 이유는 직접 하드웨어에 접근하는 것이 아니라 추상화된 하드웨어 계층을 통해 접근하므로 같은 연산이 2 번 소모되는 오버헤드가 발생하기 때문이다. 이에 대한 성능을 측정하기 위해 데이터가 실제 장치에 쓰여질 때까지 소모된 시간을 측정하며 이를 통하여 성능을 비교한다. 네이티브 리눅스에서 소모된 시간을 T(n)이라고 하며 Wombat 리눅스에서 소모된 시간을 T(w)라고 한다. 이 때 측정된 소모된 시간에 관한 결과를 나누어서 성능을 비교한다.

5.4 실험 결과

5.4.1 LCD 디바이스 드라이버

LCD 디바이스 드라이버의 성능을 측정하기 위해서 가장 중요하게 고려되어야 할 것이 영상 데이터를 프레임버퍼에 전송할 때 사용되는 DMA 의 속도이며 이에 대한 성능을 측정하기 위해 DMA 전송이 시작된 순간부터 전송이 끝나는 순간까지의 시간을 측정한다. 실험을 위해 MPEG2 동영상을 사용하였으며 영상 프레임의 수와 해상도를 다르게 하였다. 표 1은 실험에 사용된 동영상의 사양을 보여준다.

표 1 MPEG2 동영상 사양

	해상도	포맷	프레임 수
a.mpg	320 x 256	RGB565	60
b.mpg	720 x 576	RGB565	88
c.mpg	160 x 28	RGB565	198

평가 기준은 해상도와 프레임 수에 따른 성능 비교로써 Y 축은 전송 시 소모된 시간을 보여준다. 그림 2은 실험을 통해 나온 결과이며 OKL4 의 디바이스 드라이버의 성능이 네이티브 리눅스보다 떨어짐을 보인다. 이유는 하드웨어 추상화 계층으로 인한 DMA 전송을 위한 영상 데이터 전송이 네이티브 리눅스 보다 한번 더 발생하기 때문이다.

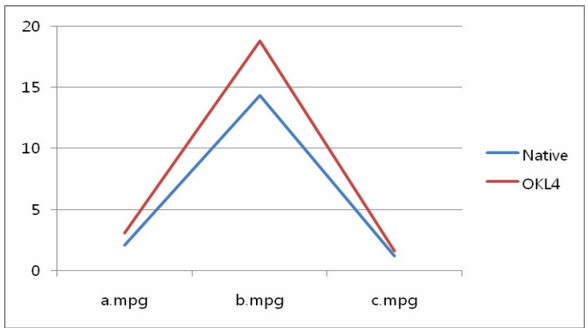

그림 2 DMA 성능 측정

우선 프레임 개수에 따른 성능의 차이는 거의 없음을 알 수 있으며 이보다 해상도에 따라 성능 차이가 큼을 알 수 있다. c.mpg 의 경우 a.mpg 보다 프레임 수가 3 배 많지만 더 빠른 전송 속도를 보인다.

5.4.2 전체 시스템

시스템 콜에 대한 성능을 측정한 결과 네이티브 리눅스가 OKL4 보다 더 좋은 성능을 보였다. 그림 3은 read/write/open/close 와 같은 시스템콜에 대한 성능측정을 보여주는 그래프이다.

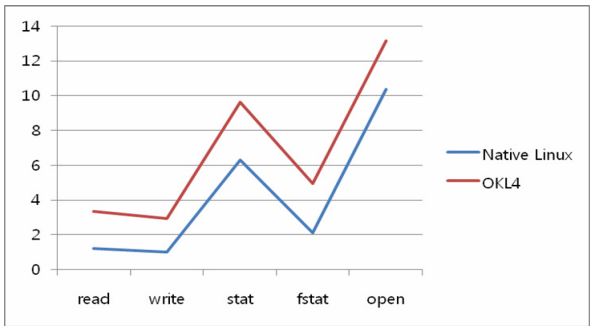

그림 3 Systemcall 성능 측정

실험 방법에서 언급했듯이 시스템 콜 호출은 해당 요청에 대한 처리를 위해 사용자 영역에서 커널 영역으로 문맥전환이 발생하는데 가상화된 Wombat 리눅스에서는 이 문맥전환이 한 번 더 발생하기 때문에 성능 저하를 보인다

그림 4는 IPC 에 관련된 성능 측정 결과이며 같은 주소 공간을 사용하는 사용자 영역에서 동작하는 프로세스간 통신이기에 L4 가 네이티브 리눅스보다 더 좋은 성능을 보인다.

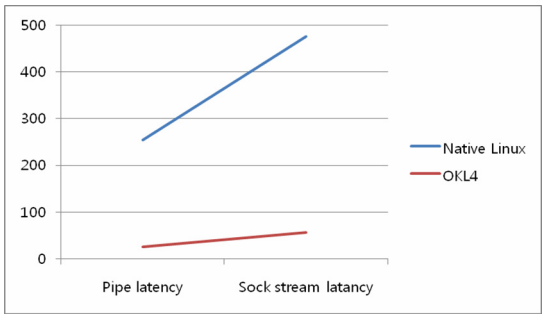

그림 4 IPC 성능 측정

마지막으로 프로세스간 문맥전환 시 소모되는 시간을 측정하였다. 대부분 OKL4 가 네이티브 리눅스에 비해 좋은 성능을 보인다. 그림 5는 생성된 프로세스의 크기가 8kb 일 때 발생하는 문맥전환 소모시간을 측정한 것이다. 네이티브 리눅스는 프로세스의 생성 개수와는 상관없이 꾸준한 성능을 보이는 반면 L4 의 문맥전환 소모시간은 프로세스의 개수가 많아짐에 따라 추가적인 오버헤드가 발생하지만 네이티브 리눅스보다 뛰어난 성능을 보인다.

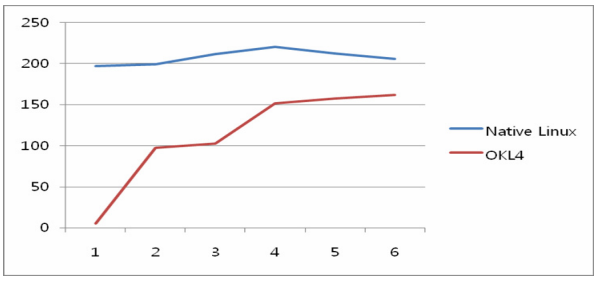

그림 5 문맥전환 성능 측정

6. 결론 및 향후 계획

전체적인 성능은 네이티브 리눅스에 비해 약 70~80%의 성능을 보이는 것을 실험 결과를 통하여 알 수 있었다. L4 가 동일 주소 공간에서 발생하는 IPC 나 문맥전환과 같은 연산에 우수성을 보이는 것은 L4 의 IPC 성능이 뛰어나다는 것을 알 수 있으며 또한 L4 는 동일한 주소 공간에서 발생하는 문맥 전환 시 XScale 기반 프로세서에서 지원하는 FASS 기능을 지원하기 때문에 좋은 성능을 보이지만 시스템 콜이나 디바이스 드라이버와 같이 반드시 커널 공간을 거쳐야 하는 연산을 수행하는 경우에는 네이티브 리눅스가 L4 보다 뛰어난 성능을 보인다. 그리고 프로세스가 더욱 많아지고 사이즈가 커지면 Cache flush 와 같은 추가적인 오버헤드가 발생하며 이로 인해 L4 의 문맥전환 지연시간이 커지는 것을 알 수 있다.

그러나 L4 의 디바이스 드라이버 프레임워크를 이용하여 작성한 드라이버는 시스템의 악영향을 미치는 요인을 최소화함으로 안정성과 신뢰성을 보장한다.

향후 계획으로는 KMU 참조 플랫폼에 미 구현된 디바이스 드라이버(DMB, AUDIO, KEYPAD, MTD 등)를 작성하여 시스템의 성능을 평가하는 하는 것이며 다수의 운영체제가 동작하는 시스템을 구축하여 응용 프로그램이 특성에 맞는 운영체제에서 동작할 수 있는 시스템을 구축한다.

참고문헌

[1] Ben Lesile, Carl van Schaik and Gernot Heiser, “Wombat A portable User-Mode Linux for Embedded Systems,” proceedings of the 6th Linux.Conf.Au, Canberra, April, 2005.
[2] Gernot Heiser, “Do microkernel suck,” 9th Linux.Conf.Au, Melbourne, January, 2008.
[3] Open Kernel Labs, inc., “Writing a Device Driver”, White Paper, April, 2008.
[4] VMware, inc. “Understanding Full Virtualization, Paravirtualization, and Hardware Assist,” White Paper, Nov. 2007.
[5] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, Chul-Ryun Kim, ”Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones,” 5th IEEE Consumer Communications and Networking Conference, 2008.
[6] Gernot Heiser, “Hypervisors for Consumer Electronics,” Proceedings of the 6th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, USA, January, 2009.
[7] Gernot Heiser, Kevin Elphinstone, Ihor Kuz, Gerwin Klein and Stefan M. Petters, "Towards trustworthy computing systems: taking microkernels to the next level," ACM Operating Systems Review, 41(4), 3–11, (July, 2007).
[8] Open Kernel Labs, inc., "Virtualization for Embedded Systems," White Paper, Nov, 2007
[9] OK-Labs, http://www.ok-labs.com/
[10] Xen, http://xen.org/