임베디드 시스템에서의 실시간 운영체제 가상화 설계 및 구현

양종철*, 김한빛**, 조상준***, 조재일****, 안우현*
*광운대학교 컴퓨터과학과
**KAIST 전산학과
***(주)NHN
****(주)광성전자
shining-soul@hanmail.net, hbkim@camars.kaist.ac.kr, sangjun@kw.ac.kr,
jaeilcool@hanmail.net, whahn@kw.ac.kr

The Design and Implementation of a RTOS Virtualization for Embedded Systems

Jongchul Yang, Hanbit Kim, Sangjun Joe, Jaeil Joe, Woohyun Ahn
*Department of Computer Science, Kwangwoon University
**Division of Computer Science, KAIST
***NHN Corp.
****Kwangsung Corp.

요 약

최근 운영체제 가상화 기술을 통한 이점들로 인해 이에 대한 많은 관심이 대두 되고 있다. 현재 다양한 분야에서의 가상화 연구가 활발히 진행되고 있으며, 범용 운영체제를 위한 상용화 제품도 여러 개 존재한다. 또한 임베디드 시스템에서의 가상화 기술 연구도 큰 관심을 끌고 있지만, 순수 RTOS 가상화의 사례는 없다. 임베디드 시스템에서의 RTOS 가상화가 필요한 예로는 2 CPU - 2 RTOS 구조를 갖는 휴대전화 단말기를 들 수 있는데, 이 경우에 가상화를 적용하면 응용프로그램의 재사용과 생산원가 절감의 효과를 얻을 수 있다. 본 논문에서는 임베디드 시스템에서의 실시간 운영체제 가상화 기법을 제안하고, 이를 위한 인터럽트 가상화, OS간 스케줄링, OS간 통신 등의 기술을 개발하여 실험을 통해 확인한다.

1. 서 론

최근 가상화 기술들에 대한 연구가 해외에서 활발히 진행되고 있고, 여러 연구 결과들이 나오고 있다. 가상화 기술이 관심 받는 이유는 시스템의 물리적 자원을 다수의 사용주체에게 각각 독점적으로 사용되는 것과 같은 환경을 제공함으로써 하드웨어 비용 절감, 높은 시스템 자원 사용의 효율성, 오류의 격리(fault isolation), 기존 어플리케이션의 재사용(Legacy reuse) 등의 이점이 있기 때문이다. 가상화 기술의 적용 분야를 보면 분산시스템이나 서버에서의 가상화 연구가 활발하며, 범용 OS를 위한 상용화 제품도 존재한다. Xen[1], Denali[2], VMWare[3]가 대표적인 예라 할 수 있다. 가상화 기술이 관심 받는 이유는 하지만 실시간 운영체제나 임베디드 시스템에서의 가상화 기술 연구는 아직 부족한 편이다. VirtualLogx[4]처럼 실시간 운영체제(RTOS)와 비실시간 운영체제에 대한 가상화의 경우는 있으나, 순수 RTOS 가상화의 사례는 없다.

임베디드 시스템에 대한 RTOS 가상화 적용이 필요한 예로는 CDMA 단말기를 들 수 있다. 실제 CDMA 휴대전화 단말기의 경우 프로토콜 스택을 처리하기 위한 모뎀 CPU, 어플리케이션 처리를 위한 CPU에서 각각 REX와 Symbian이 탑재되는 2 CPU - 2 RTOS 구조를 갖고 있다. 이 구조에 가상화 기술을 적용하면 기존의 RTOS들을 한 개의 CPU상에서 같이 사용 할 수 있으므로 응용프로그램의 재사용이 가능해지고 생산 원가를 줄일 수 있게 된다.

본 논문에서는 임베디드 시스템 상에서 두 개의 RTOS를 탑재시켜 구동하고, RTOS간의 통신도 가

능한 가상화 기술을 제안한다. 그리고 이를 구현함으로써 RTOS 가상화에 대한 국내 원천기술 개발 가능성을 보인다.

그리고 PXA255 임베디드 보드(embedded board)에서 uC/OS-II와 Nano-Qplus의 두 RTOS를 사용한 실험을 통해, 인터럽트 가상화, OS간 통신, OS간 스케줄링 기법들이 적용된 가상화 기술을 확인한다.

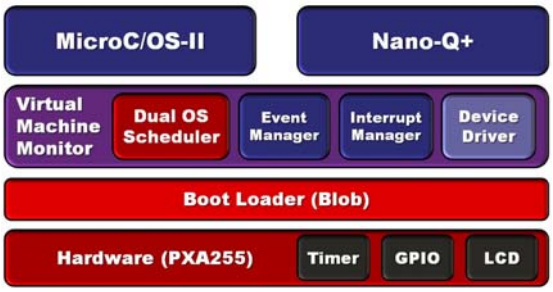
본 논문의 구성은 다음과 같다. 제2장에서는 제안하는 시스템의 개요를 보이고, 제3장에서는 이 시스템의 각 구성요소에 대해 설명하며, 제4장에서 실험결과 보인다. 마지막으로 제5장에서 본 논문의 결론을 맺는다.

2. 시스템 개요

2.1 아키텍처

본 논문에서 제안하는 RTOS 가상화는 그림1에서 보여지는바와 같이 다중 계층 구조로 이루어진다. 하드웨어 위에 부트로더(bootloader)가 탑재되며, 그 위에 핵심적인 가상화 기능을 제공하는 Virtual Machine Monitor(VMM)가 위치한다. VMM에 의해 관리되는 분리 된 영역에 두개의 RTOS가 탑재되며, 각 RTOS는 고유의 응용프로그램을 실행시킨다.

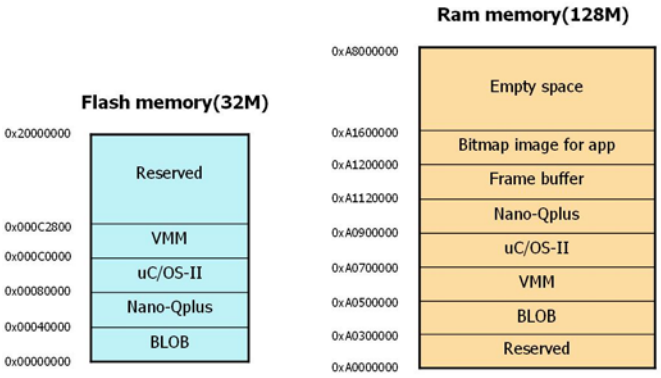
탑재되어 실행되는 Guest OS로는 uC/OS-II Ver2.1과 ETRI에서 개발된 Nano-Qplus를 사용하며, 하드웨어 플랫폼은 ARM 5TE ISA(Instruction Set Architecture)를 기반으로 하는 PXA255-Pro3를 사용한다.



< 그림 1. System Architecture >

2.2 메모리 맵 (Memory Map)

PXA255-Pro3는 플래시 메모리 32MB와 128MB의 SDRAM을 사용한다. 그림 2는 시스템 구성에 사용된 메모리 맵을 나타낸다. 부트로더와 두 RTOS, VMM이 플래시메모리에 저장되고, 부팅 시 메모리 맵에 따라 램에 각각 탑재되어 실행된다. PXA255에 장착되어 있는 LCD 제어에 사용할 프레임버퍼와, 테스트 프로그램에서 사용할 이미지 데이터들도 지정된 램 영역에서 사용된다.



< 그림 2. Memory Map >

3. 시스템 구성 요소

3.1 Virtual Machine Monitor(VMM)

VMM은 하드웨어 플랫폼과 OS들 사이에서 교량역할을 한다. VMM은 3가지 기능을 하는데, 이는 ROTS 인터럽트 가상화, OS간의 문맥교환(context switching), OS간의 통신에 발생하는 이벤트 관리이다.

VMM은 그림3처럼 3가지 요소로 구성된다. 각 요소는 인터럽트 가상화 모듈(module), Dual OS Scheduler(DOSS), Event Manager이며 이에 대한 구체적 설명은 다음 절부터 한다.

< 그림 3. Virtual Machine Monitor >

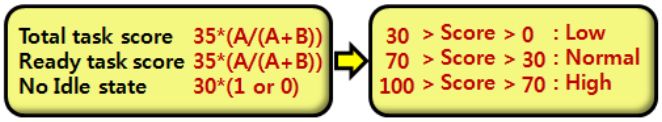
3.1.1 Dual OS Scheduler (DOSS)

DOSS는 두 RTOS의 우선순위를 조정하여 OS간 문맥교환을 제어하는 역할을 한다. 주기적으로 발생하는 타이머 틱(timer tick)이 발생 할 때마다 DOSS는 두 OS로부터 우선순위 결정에 필요한 정보들을 취합하고, 정책에 따른 우선순위 산출 방법을 통해 순위를 갱신 시키며, 이를 바탕으로 문맥교환을 수행하게 된다.

스위칭 수행 시점에 적용되는 스케줄링 정책은, 두 OS가 동일한 우선순위라면 라운드로빈 스케줄링을 사용하고, 두 OS의 우선순위가 다르다면 선점 스케줄링(preemptive scheduling)을 사용함으로써 OS 단위의 실시간성을 최대한 보장해준다.

OS의 우선순위는 RTOS의 실시간성을 감안하면서도 최대한 공평하게 CPU자원을 배분 할 수 있는 정책을 사용하여 결정된다. OS의 우선순위 산출에 사용되는 요소들은 총 태스크의 개수, ready 태스크의 개수, idle 태스크의 동작 여부, 이렇게 3가지이며, 각각 일정 가중치가 적용 된다. 이 요소들과 가중치는 처리할 작업이 많은 OS가 높은 우선순위를 갖도록 하기 위해 선택 되었다. 총 태스크의 수와 ready 태스크의 수가 많을수록 OS의 작업량도 늘어나기 때문에 이 두 가지 요소에 각 35%씩의 높은 가중치를 적용하여 우선순위 결정에 사용한다.

그림 4는 우선순위의 계산 공식을 나타낸다. 100을 기준으로 했을 때, 총 태스크와 ready태스크 항목은 각각 35%의 가중치를 주고, OS의 idle 상태 여부에 30%의 가중치를 둔다. 가중치를 적용시킨 각 항목을 더한 값에 따라 High, Normal, Low의 우선순위 중 한 개가 적용된다.



A : 현재 수행되고 있는 OS의 ready 태스크 또는 총 태스크의 수
B : 다른 OS의 ready 태스크 또는 총 태스크의 수
A+B : 전체 시스템에서의 ready 태스크 또는 총 태스크의 수

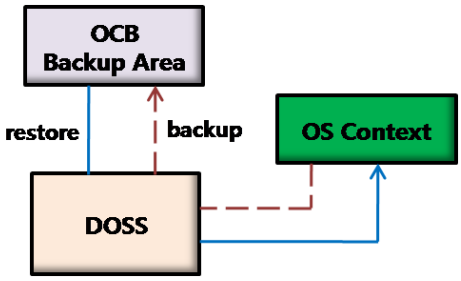
< 그림 4. OS 우선순위 계산 >

타이머 인터럽트 발생 시 스케줄러에 의해 두 OS의 우선순위가 새로 계산되고 갱신 된다. 스케줄러는 갱신된 우선순위를 근거로 문맥교환 여부를 판단하여 문맥교환을 수행한다. OS간의 문맥교환 시 DOSS는 그림5에서 보여지는 OS Control Block(OCB)을 교체한다. OCB는 그림6과 같은 과정에서 문맥교환 되어 나가는 OS가 차후 동작을 재개할 때 필요로 하는 모든 정보를 담고 있다. OCB는 범용 레지스터, 모드별 스택포인터, 인터럽트 마스킹 상태, 프로그램 상태 레지스터를 포함하며 84바이트로 구성된다.

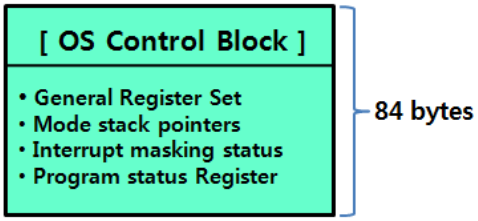
DOSS는 문맥교환 시 지정된 메모리 영역에 현재 OS의 OBC를 백업하고, 기존에 백업된 이전 OS의 OBC를 복구함으로써 OS가 기존에 동작하던 환경으로 설정한 후 OS 수행을 재개 시킨다.

3.1.2 인터럽트 가상화

발생하는 인터럽트를 처리할 OS를 정확히 분별하여 해당 OS의 인터럽트의 처리 루틴을 호출하기 위



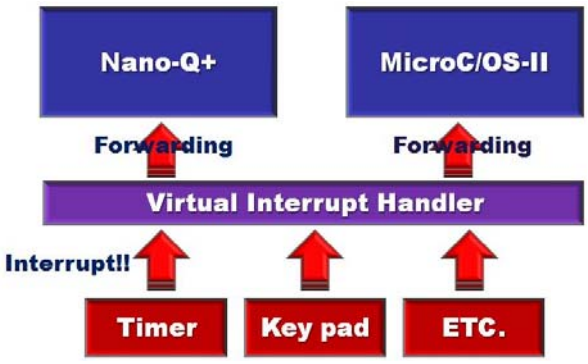
< 그림 5. Backup/Restore OCB >



< 그림 6. OS Control Block >

해 인터럽트 가상화가 필요하다.

VMM에서 관리하는 인터럽트 가상화는 부트로더의 인터럽트 벡터 테이블에 VMM의 가상 인터럽트 핸들러를 등록함으로써 이루어진다. 그림7에서 보여지는바와 같이, 모든 인터럽트는 VMM의 가상 인터럽트 핸들러로 전달된다. VMM은 상황에 따라 직접 처리하거나, 인터럽트의 대상이 되는 OS를 판별하여 해당 OS의 인터럽트 처리 루틴을 호출한다.



< 그림 7. Virtual Interrupt Handling >

본 논문에서는 두 가지 인터럽트만 사용한다. 하나는 타이머 인터럽트고, 다른 하나는 키 인터럽트다. 타이머는 시스템에서 한 개만 사용하도록 설계 했다. 즉, VMM과 OS들이 같은 타이머를 이용하기 때문에 타이머 인터럽트 발생 시, 두 단계에 거쳐 처리된다. 타이머 인터럽트가 발생하면 1차적으로 VMM에서 OS간 스케줄링 동작을 위해 처리된다. 그 다음 현재 수행 중인 OS의 타이머 인터럽트 핸들러를 호출함

으로써 OS내에서 타이머 인터럽트가 다시 한 번 처리되도록 한다.

키 인터럽트는 PXA255 실험 보드의 키패드를 사용하여 발생 된다. 키패드의 특정키를 통해 OS를 선택한 후 일반 키들을 누르면 해당 OS로 인터럽트를 보낼 수 있다. 즉, OS 선택키를 통해 현재 동작중인 OS뿐 아니라 다른 OS를 목표로 하여 인터럽트를 보낼 수도 있다. 키 인터럽트 처리는 Bottom half 방식으로 처리된다. 인터럽트 발생 시, 이에 대한 이벤트를 각 OS의 이벤트 리스트에 저장만 하고, 이에 대한 처리는 인터럽트 핸들 루틴 종료 후, OS 수행 시간에 이루어진다.

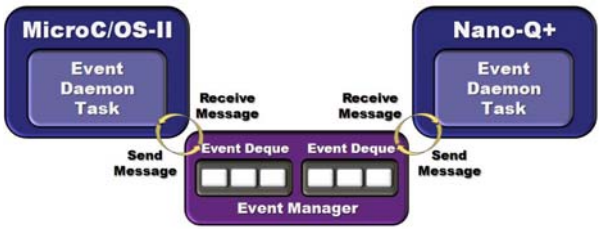
3.1.3 Event Manager

특정 목적을 위해 여러 개의 OS를 동시에 사용하는 경우 OS간 통신 방법이 필요하게 된다. 2CPU 휴대폰을 예로 들면, 전화가 걸려올 때 모뎀 CPU가 어플리케이션 CPU에 송신자 정보를 전달해 주어 발신 정보가 표시된다. 이와 같은 경우가 OS간의 통신 방법이 필요한 예이다.

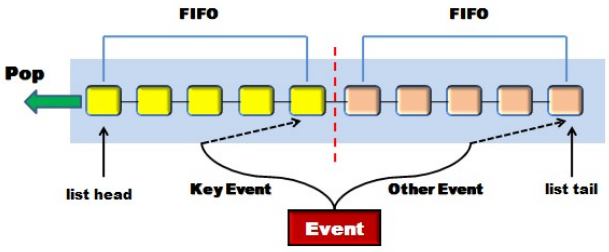
본 논문에서는 OS간 통신을 위해 이벤트 방식을 사용했다. 그림8처럼 이벤트들은 VMM의 이벤트 매니저를 통해 관리된다. 사용되는 이벤트 종류는, 8비트 비트 마스크에 따라 최대 8가지가 사용 될 수 있다. 그리고 각 OS에는 통신에 사용 될 이벤트 리스트와 이벤트 검사 데몬이 추가되어 사용된다. 이벤트 리스트는 발생된 이벤트들이 처리될 때까지 모아두는 역할을 하고, 이벤트 데몬은 주기적으로 이 리스트를 검사하여 이벤트를 처리한다.

이벤트는 OS간의 통신 외에도, VMM의 bottom half 방식의 키 인터럽트 처리에도 사용된다. 키 인터럽트 발생 시, 가상 인터럽트 핸들러는 이벤트 매니저를 통해 처리한다. 이 때, 이벤트 매니저는 키 이벤트를 생성 하여 특정 OS의 이벤트 리스트에 추가시킨다.

각 OS는 이벤트 리스트에 있는 이벤트를 처리하기 위해, 이벤트 매니저가 제공하는 함수를 사용하여 리스트에 접근할 수 있다. OS간 통신에 사용되는 이벤트에 대해서는 전형적인 FIFO형식으로 리스트를 사용한다. 하지만 OS간의 메시지보다 키 이벤트 메시지를 먼저 처리하는 정책에 따라, 그림9와 같이 키 이벤트 FIFO와 다른 이벤트 FIFO를 연결한 복합 FIFO 방식을 사용한다.



< 그림 8. Inter OS Communication >



< 그림 9. 이벤트 리스트의 구조 >

4. 실험

MicroC/OS-II 와 Nano-Qplus는 LCD화면을 각각 절반씩 왼쪽과 오른쪽을 사용한다. 각 RTOS는 서로 다른 메뉴 화면으로 구성되어 있다. Asterisk('*')와 Sharp('#')버튼은 MicroC/OS-II와 Nano-Qplus를 선택하는 기능을 한다. RTOS가 선택된 상황에서 2, 4, 6, 8번 버튼을 이용하여 메뉴선택 커서를 위, 오른쪽, 아래쪽, 왼쪽으로 이동 시킨다. 이것은 키 인터럽트가 선택한 RTOS에게 보내져 처리됨을 보여주고, 또한 두 RTOS가 LCD 디바이스를 공유하여 사용하는 것을 보여준다. 이 어플리케이션은 눈으로 확인하기 힘든 가상화 기술을 가시화시켜 확인 가능케 하는 것을 목표로 한 간단한 구조이다.

그림10은 초기화면, 그림 11과 그림12는 초기화면에서 차례로 MicroC/OS-II와 Nano-Qplus를 선택했을 때의 메뉴 화면을 보여준다.

< 그림 10. 어플리케이션 초기 화면 >

< 그림 11. MicroC/OS-II 선택 화면 >

< 그림 12. Nano-Qplus 선택 화면 >

5. 결론 및 향후 연구

본 논문에서는 임베디드 시스템에서의 RTOS 가상화가 가능함을 구현을 통해 입증하였다. 이 시스템은 복수의 CPU에서 복수의 RTOS를 운영하는 임베디드 시스템을 목표로 하였으며, 특히 휴대 단말기기의 생산비용 절감 및 소형화에 기여할 수 있을 것 이라 기대되며 RTOS 가상화에 대한 국내 원천 기술 개발 가능성을 보여준다.

본 논문에서는 OS 단위의 우선순위 정책을 사용했지만, 향후 OS들의 태스크를 한데 묶어 우선순위를 부여하고 태스크의 스케줄링에 따라 효율적으로 OS를 스위칭 시키는 연구를 지속 할 예정이다.

참고문헌

[1] Ian Pratt, Keir Fraser, Steven Hand, Christain Limpach, Andrew Warfield, Xen 3.0 and the Art of Virtualization, Linux Symposium, Ottawa, 2005.

[2] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the Denali isolation kernel. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 195.210, Boston, MA, USA, Dec. 2002.

[3] VMWare. VMWare home page. Web site: http://www.vmware.com

[4] VirtualLogix. VirtualLogix home page. Web site: http://www.virtuallogix.com