QEMU 에뮬레이터 상에서 MicroC/OS-II를 이용한 XenARM의 성능 측정

*황고운, 신동군
성균관대학교 정보통신공학부
e-mail : *cloudgw@gmail.com, dongkun.shin@gmail.com*

Performance of XenARM on QEMU Emulator Using MicroC/OS-II Operating System

*Go-Woon Hwang, Dong-Kun Shin
School of Information and Communication Engineering
Sungkyunkwan University

Abstract

As embedded systems are more complex in these days, virtualization techniques are important problem. This paper introduces a XenOnARM virtualization technique and measures the performance of this technique. For using XenARM, we use QEMU as an emulator and this is based on Android platform goldfish. The microC/OS-II is adopted as an operating system.

I. 서론

점점 복잡해지는 임베디드 시스템의 추세에 따라 가상화 기술이 큰 주목을 받고 있다. 가상화 기술을 이용하면 하나의 물리적인 자원에서 여러 가지 독립적인 일을 하는 것이 가능하기 때문이다. 그렇기 때문에 가상화 기술을 때로는 간단하게 하나의 하드웨어 자원에서 복수개의 운영체제(OS)를 작동 가능하게 하는 기술이라고 표현하기도 한다. 하나의 물리적인 자원에서 여러 개의 서로 독립적인 환경을 구현하기 위해서는 그 사이에 Virtual Machine Monitor 혹은 하이퍼바이저(hypervisor)가 필요하다. Xen은 이러한 VMM 중에 하나이다. 2003년 캠브리지대학 Computer Laboratory에서 개발된 Xen은 open-source로 제공되며 뛰어난 성능과 보안성 때문에 많은 주목을 받고 있는 기술이다. 본 논문에서는 삼성전자 종합기술원에서 연구 개발한 Secure Xen on ARM(XenARM)을 안드로이드 플랫폼에서 동작하도록 하여 그 성능을 측정해 볼 것이다.

II. 본론

2.1 XenARM

Xen은 부분가상화를 이용한 가상화 기술의 하나이다. Xen은 독립적인 Virtual Machine들을 domain으로 제공하는데, 그 중에서 host OS가 동작하는 domain을 Domain0(dom0)라고 한다. dom0는 Hardware 자원에 접근이 가능한 privileged domain이다. 나머dwguest OS는 dom1, dom2, …… , domU로 표현되며 각각은 hypercallin0(해서 서로 상호▯ (은 hy다. hypercall이란 domain에서 hypervisor로 전달되는 일종의 Software trap이다. 만약 unprivileged domain에서 Hardware에 접근in원할 제에는 dom0의 Back-end device driver에 요청해야만 상호▯ (@enARMivien 기술을 ARM 아키텍처에서 사용할 수 있도록 포팅한 것을 말한다.[1,2]

2.2 MicroC/OS-II

µC/OS-II는 실시간 임베디드 시스템을 위한 RTOS 중에 하나이다. 일부 프로세서 의존적인 부분을 제외하고 대부분이 ANSI C로 작성되었다. 항상 실행 가능한 최고 우선 순위의 태스크를 실행하는 선점형 커널이며 최대 64개의 태스크까지 멀티태스킹을 지원한다. 그 밖에 메일박스, 메시지 큐, 세마포어 등의 서비스를 제공한다.[3]

2.3 포팅

µC/OS-II를 XenARM 아키텍처 상에서 사용하기 위해서는 별도의 포팅 과정이 필요하다. 포팅을 위해서는 전에 µC/OS-II 코드 중에서 프로세서에 의존적인 3개의 코드만 수정하면 된다. 우선 OS_CPU.H 파일에서 #define 상수 값을 설정하고, 10개의 data type과 3개의 macro 함수를 정의해야 한다. OS_CPU_C.C 파일에서 6개의 C언어 함수를 작성하고, OS_CPU_A.ASM 파일에서 4개의 어셈블리 함수를 작성하면 된다.[3,4]

OS_CPU.H 파일에서 #define OS_STK_GROWTH는 스택 포인터의 방향을 나타낸다. 만약 스택이 높은 메모리에서 낮은 메모리 방향으로 자란다면 1, 그 반대의 경우에는 0으로 설정한다. 10개의 data type은 프로세서마다 워드의 길이가 다르기 때문에 정의해서 사용해야 한다. 3개의 macro 함수는 OS_ENTER_CRITICAL(), OS_EXIT_CRITICAL(), OS_TASK_SW()를 의미한다.

OS_CPU_C.C 파일에서 정의하는 6개의 함수는 OSTaskStkInit(), OSTaskCreatHook(), OSTaskDelHook(), OSTaskSwHook(), OSTaskStatHook(), OSTimeTickHook()을 의미하고 이 중에서 실제로 중요한 함수는 OSTaskStkInit() 하나이다. 나머지는 반드시 정의해야 하는 함수이지만 내용은 중요하지 않다.

OS_CPU_A.ASM 파일에서 정의하는 4개의 어셈블리 함수는 OSStartHighRdy(), OSCtxSw(), OSIntCtxSw(), OSTickISR()이다. 만약 인라인 어셈블리 기능을 지원하는 컴파일러를 사용한다면 이 함수들은 OS_CPU_C.C 코드 내에 포함하는 것이 가능하다.[5]

본 논문에서는 XenARM 홈페이지에서[2] XenARM을 위해 포팅되어진 ucos_ii_xen_on_arm을 다운받아 사용하였다.

Ⅲ. 실험

XenARM의 성능을 알아보기 위해서 3개의 task를 수행시켜보고 각각의 사이클 수와 명령어 수를 측정하였다. 운영체제로 Fedora 8을 사용했으며 성능 측정을 위해 별도로 수정된 QEMU 에뮬레이터에 XenARM을 실행시켰다. 도메인으로는 dom0에 MiniOS, dom1에 MicroC/OS-II를 사용하였다.

표 1. 3개의 task에 대한 성능 측정 결과

	Task1	Task2	Task3
cycle	323172	725382	1798492
inst.	119808	262144	704917
cycle/inst.	2.7	2.77	2.74

표1에서 cycle은 해당 task가 수행되는 데 총 소요된 사이클의 수를 의미하고, inst.는 총 명령어의 수를 의미한다. 따라서 cycle/inst.는 명령어의 평균적인 사이클 수를 의미한다. Task의 복잡도는 task3으로 갈수록 커진다.

Ⅳ. 결론 및 향후 연구 방향

표1에서 task3으로 갈수록 사이클의 수와 명령어의 수가 증가하는 것을 볼 수 있다. 따라서 task가 복잡해질수록 해당 task를 수행하기 위해 더 많은 시간이 소요됨을 알 수 있다. 그러나 하나의 명령어에 대한 평균 사이클 수는 비슷하게 나타났다. 결과 data를 살펴보면 하나의 명령어가 30 사이클 이상이 걸려 수행되는 경우도 있지만 평균적으로 약 2.7 사이클이 소요됐다는 것을 알 수 있다.

XenARM의 성능 파악을 위해 3종류의 task에 대한 사이클 수와 명령어 수를 실험해보았다. 그러나 이 수치만으로 XenARM의 성능을 파악하는 데는 부족함이 많다. 본 논문에서는 XenARM을 QEMU 에뮬레이터 위에서만 동작시켰지만, 직접적으로 하드웨어에서 동작시켜 수치를 비교하는 것도 하나의 방법이다. 그러나 이를 위해서는 XenARM을 하드웨어에 적합하도록 포팅하는 과정이 별도로 필요할 것이다.

참고문헌

[1] Prabhakar Chaganti, Xen Virtualization: A fast and practical guide to supporting multiple operating systems with the Xen hypervisor, Packt Publishing, 2007

[2] http://wiki.xensource.com/xenwiki/XenARM

[3] Jean J. Labrosse, MicroC/OS-II The Real-Time Kernel, Osborne/McGraw-Hill, 2002

[4] www.ucos-ii.com

[5] 김대홍, uC/OS-II 뛰어넘기(VI), Embedded World, 2004.04