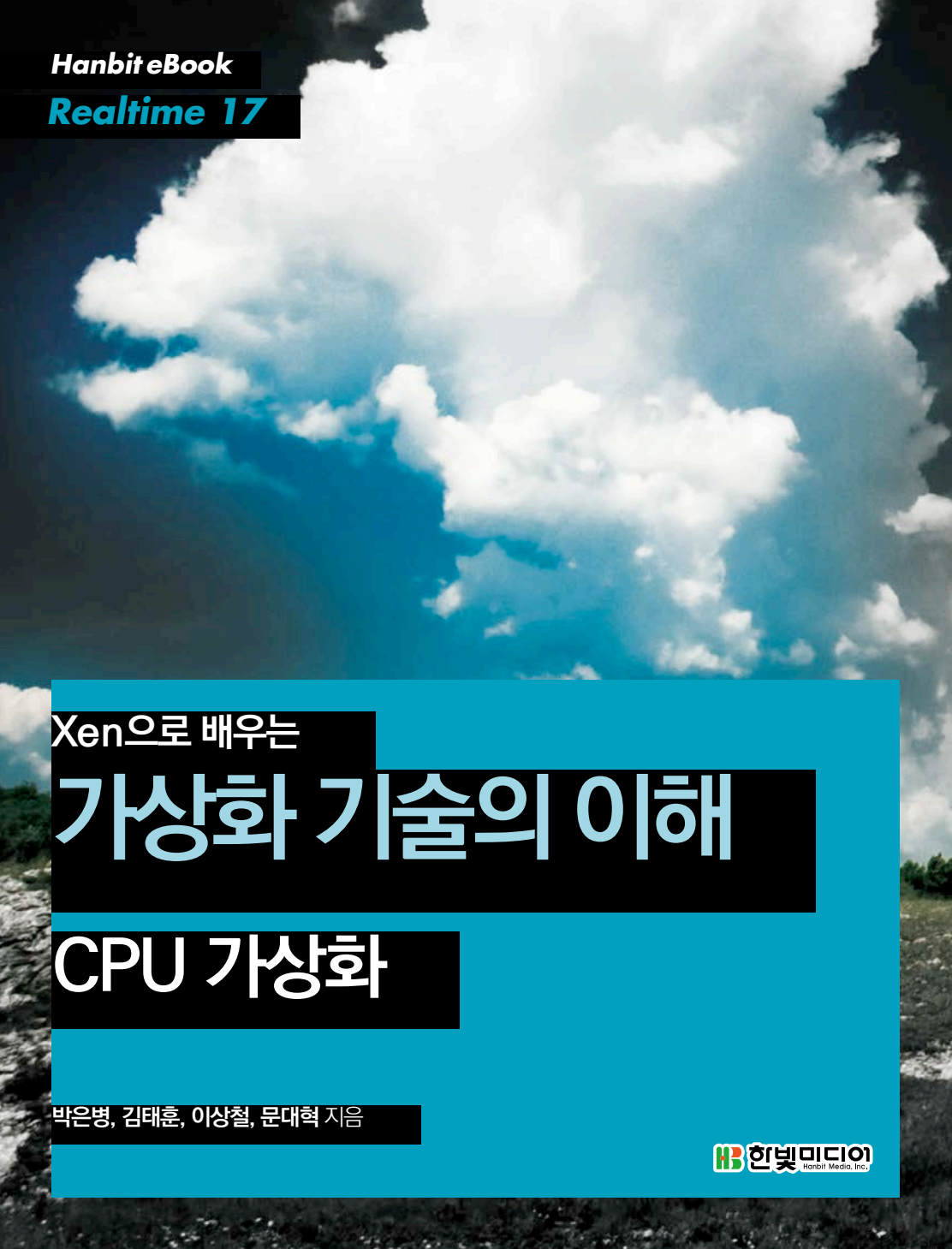

Hanbit eBook
Realtime 17
Xen으로 배우는
가상화 기술의 이해
CPU 가상화
박은병, 김태훈, 이상철, 문대혁 지음
한빛미디어
Hanbit Media, Inc.

Xen으로 배우는

가상화 기술의 이해
CPU 가상화

Xen으로 배우는 **가상화 기술의 이해 – CPU 가상화**

초판발행 2013년 2월 25일

지은이 박은병, 김태훈, 이상철, 문대혁 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-604-3 15560 / **정가** 9,900원

책임편집 배용석 / **기획** 김창수 / **편집** 이중민, 이세진
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanb.co.kr / **이메일** ask@hanb.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanb.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

지은이_ **박은병**

서울대학교에서 석사 학위를 받았으며, 현재 University of Toronto에서 컴퓨터 공학 박사 과정을 공부하고 있다. 석사 과정을 공부하면서 Xen을 이용해 가상화 관련 연구를 진행했다. 시스템 소프트웨어 전반에 관심이 있으며, 현재 기계학습 관련 응용 분야에 흥미를 느껴 공부 중이다.

지은이_ **김태훈**

임베디드, 커널, 가상화, 네트워크, 디바이스 드라이버를 주로 다루는 시스템 프로그래머이다. (주)WIZnet 재직 시절에 개발한 W5300 네트워크 드라이버가 리눅스 커널에 포함되었다. 오픈 소스와 해커 문화를 동경하며, 특히 리누스 토발즈가 우상이다. 현재는 DINOS라는 고성능 ARM 아키텍처를 타깃으로 하는 운영체제를 개발 중이다.

지은이_ **이상철**

하드웨어 개발부터 시작해 시스템 소프트웨어 개발로 차츰 업무를 변경해왔다. 주로 임베디드 시스템 프로그램과 디바이스 드라이버를 개발했으며, 리눅스 커널 관련 업무 또한 담당했다. 현재는 알티캐스트에서 보안 관련 모듈을 개발 중이다.

지은이_ **문대혁**

한양대학교를 휴학하고 사이냅소프트에서 문서 처리 관련 프로그램을 개발 중이다. 시스템 소프트웨어를 포함해 컴퓨터 공학과 연관이 있다. 우연히 본 스터디 모집공고를 계기로 뛰어난 개발자들과 함께 Xen을 분석하는 기회를 가지게 되었다.

저자 서문

이 책은 Xen 하이퍼바이저를 통해 가상 머신 모니터를 설명한 책입니다. Iamroot 라는 스터디 그룹을 통해 1년 반 정도 같이 매주 토요일에 오프라인 모임에서 Xen 하이퍼바이저를 공부했고, 다른 저자분들의 뜻을 모아 공부한 내용을 정리하자는 의도에서 기획했습니다. 저자분들 모두 글 쓰는 데 익숙하지 않은 관계로 아는 지식을 글로 풀어내기가 쉽지 않았지만, 많은 퇴고를 거쳐 마침내 출판하기에 이르게 되었습니다. 이 책이 출판으로 이어지기까지 고생해주신 저자 분들과 한빛미디어 관계자분께 다시 한번 감사의 말씀을 드립니다.

이 책은 기본적인 가상 머신 모니터의 원리부터 Xen 하이퍼바이저의 소스 코드까지 설명해, 초보자부터 실제 Xen 하이퍼바이저를 사용하거나 개발하시는 분 모두에게 유용한 내용으로 구성했습니다. 운영체제와 컴퓨터 아키텍처에 대한 기본 지식만 있으면 읽을 수 있게 쓰였으나, 리눅스 커널에 대한 사전 지식이 있으면 이해하기 더욱 쉬울 것으로 생각합니다. CPU 가상화, 메모리 가상화, I/O 가상화 이렇게 크게 3가지 파트로 나누어서 설명했고, 이 책의 주제인 CPU 가상화는 시리즈 첫 책입니다. Xen 하이퍼바이저에 관심이 많다면 앞으로 출간될 책을 순서대로 읽기를 권장합니다. 전체적인 구조는 각 파트의 앞부분에 기본적인 개념을 설명하고, 뒤의 소스 코드 분석 부분에서 어떻게 실제로 구현되어 있는가를 소스 라인 별로 상세히 설명했습니다. 또한 Xen 하이퍼바이저는 리눅스 커널과 매우 밀접한 관련이 있으므로, 필요한 부분에는 리눅스 커널의 소스 코드도 추가로 설명했는데, 모든 소스 코드를 분석하고 책에 담을 수 없었기에 저자가 생각하는 중요한 부분의 소스 코드를 담았습니다. 따라서 앞부분 개념을 먼저 이해하고, 실제 소스 코드를 내려받아 책의 설명과 함께 책에 빠진 부분이나 궁금한 부분의 소스 코드를 직접 찾아가면서 공부할 것을 권장합니다.

사실 책으로 출판하기에 매우 부끄럽습니다. 하지만 가상 머신 모니터를 쉽게 한글로 설명한 책이 많지 않고, 가상 머신 모니터 공부를 시작하시는 분의 수고를 조금이나마 덜어드리고자 하는 마음에 부끄러움을 무릅쓰고 출판하기로 했습니다. 너그러운 자세로 읽어주시고, 수정이나 보완 사항이 필요하다면 추후에 반영토록 하겠습니다.

집필을 마치며

저자 일동

대상 독자 및 도서 구성

초급 | 초중급 | 중급 | **중고급** | 고급

이 책은 점차 활용 분야가 늘어나는 가상화 기술의 이해를 돕기 위해 Xen을 예로 들어 가상화 기술을 설명합니다. 이 책을 통해 가상화 기술의 개념과 Xen의 동작 방식을 이해하는 데 조금이나마 도움이 되기를 바랍니다. 'Xen으로 배우는 가상화 기술의 이해' 시리즈는 가상화 기술을 다음 세 가지 파트로 나눠 설명합니다.

CPU 가상화

Xen에서 전통적으로 사용하는 반가상화 기법과 하드웨어 지원 가상화 기술을 활용하는 전가상화 기법을 다룹니다. 가상 CPU가 실제 CPU를 어떻게 나눠서 사용하는지 가상 머신 스케줄링에서 살펴봅니다.

메모리 가상화

Xen에서 동작하는 가상 머신의 메모리 접근 방식을 설명합니다. 반가상화에서 사용하는 Shadow Paging 방식과 CPU에서 지원하는 HAP(Hardware Assisted Paging) 방식이 어떻게 이루어지는지 설명합니다.

I/O 가상화

가상 머신이 동작할 때 필요한 입·출력 장치에 접근하는 방법을 다룹니다. Xen에서 반가상화 입·출력 방식과 전가상화 입·출력 방식에는 많은 차이가 있습니다. 따라서 반가상화 입·출력 방식과 전가상화 입·출력 방식을 나눠서 설명합니다.

예제 테스트 환경

사용 프로그램	버전
리눅스 커널	3.6
Xen 하이퍼바이저	4.1.2

- 리눅스 커널 코드 참고 사이트

  : http://goo.gl/jmbPA

- 리눅스 커널 코드 다운로드

  : ftp://kernel.org/pub/linux/kernel/v3.x/linux-3.6.tar.gz

- Xen 코드 참고 사이트

  : http://goo.gl/PwfEA

- Xen 소스 코드 다운로드

  : http://bits.xensource.com/oss-xen/release/4.1.2/xen-4.1.2.tar.gz

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook 입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내시는 선배, 전문가, 고수분에게는 보다 쉽게 집필하실 기회가 되리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 준비 중이며, 조만간 선보일 예정입니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정한 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나, 저자(역자)와 독자가 소통하면서 보완되고 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여, DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT기기에서 자유롭게 활용하실 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해, 독자 여러분이 언제 어디서 어떤 기기를 사용하시더라도 편리하게 전자책을 보실 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

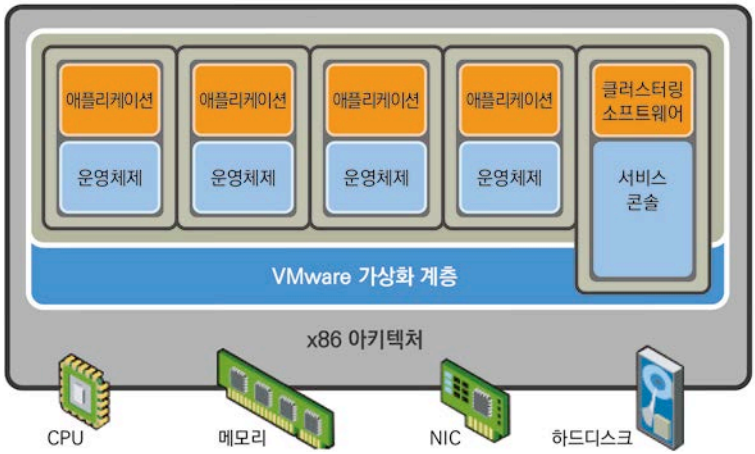
차례

1 | 가상 머신 모니터는 무엇인가?

Xen 내부 동작을 알아보기 전에는 가상 머신 모니터 혹은 하이퍼바이저Hypervisor[01]에 대한 개념과 CPU 가상화에 대한 전반적인 내용을 알아야 한다. 1장은 CPU 가상화 전체를 그려볼 기회이므로 반드시 이해하자.

가상 머신 모니터Virtual Machine Monitor란 글자 그대로 가상 머신을 모니터링하는 소프트웨어 계층을 의미한다. 물론 단순히 모니터링만 하는 것이 아니라 하드웨어 자원 관리, 가상 머신 스케줄링 등 가상 머신을 동작시키는 데 필요한 모든 작업을 담당한다.

그림 1-1 가상 머신 모니터(출처: www.vmware.com)



여기서 말하는 가상 머신이란 실제 머신은 아니지만 마치 물리 머신이 있는 것 같

01 본 책에서는 가상 머신 모니터라는 용어와 하이퍼바이저라는 용어를 섞어서 사용한다. 서로 같은 것을 부르는 말이니 혼동이 없길 바란다.

은 환경을 사용자에게 제공하는 가상화한 머신 환경을 의미한다.[02] 좀 더 쉽게 설명하면 하나의 컴퓨터에 여러 개의 운영체제를 동시에 구동할 수 있게 하는 소프트웨어라고 할 수 있다. 즉, 하나의 컴퓨터에서 가상 머신 모니터는 다수의 가상 머신을 사용자에게 제공하고, 사용자는 하나의 물리 머신 위에 다수의 가상 머신을 가질 수 있게 된다.

1.1 왜 가상화인가?

최근 들어 이곳저곳에서 가상화에 대한 논의가 뜨겁다. 실제로 가상화 기술은 이미 성숙기에 이르렀으며, 많은 기업에서 도입했거나 도입을 고려하는 상황이다. 사실 가상화 기술은 최신 기술이 아니라 1960년대 IBM의 메인프레임에서 처음 구현되었다. 하지만 당시에는 큰 이목을 끌지 못하다가 최근 10여 년 전부터 하드웨어의 발전과 함께 재조명받기 시작했다.

처음 가상화 기술이 재조명받기 시작했을 때, 기업의 가상화 기술 도입 동기는 바로 서버 통합Server Consolidation이었다. 이는 물리 서버 머신 하나가 다수의 CPU와 대량의 메모리를 갖출 수 있게 되면서 많은 자원이 유휴 상태로 남는 경우가 많아, 이렇게 유휴한 상태의 서버를 머신 하나로 통합 관리하자는 요구가 생겼다는 뜻이다. 이때 가상화 기술을 이용하면 물리 서버 하나를 가상 머신 하나로 대체하고, 이 가상 머신 여러 대를 강력한 성능을 가진 물리 머신 하나에서 동작시켜 시스템 자원의 효율성을 극대화할 수 있다.

서버 통합과 더불어 가상화 기술이 가진 또 하나의 장점은 가상 머신의 격리Isolation다. 사실 서버 통합을 하는데 가상화 기술이 반드시 필요한 것은 아니다. 예를 들어 웹 서버, 데이터베이스 서버, DNS 서버 등 여러 서버를 하나의 머신 위에 동작시켜도 된다. 하지만 운영체제 위에서 여러 개의 서버 프로세스가 동작하게 되므로 서

02 JVM(Java Virtual Machine)과 같은 가상 머신을 떠올리는 독자도 있을 것이다. 이 책에서는 JVM과 같이 프로세스 레벨의 가상화가 아닌 시스템 레벨의 가상 머신에 대해 다룬다.

버 프로세스 사이에 많은 영향을 미친다는 치명적인 문제점이 있다.

또한 자신이 홈페이지 하나를 운영하고 있고, 호스팅 업체에 웹 호스팅을 요청했다고 가정하자. 홈페이지가 다른 고객과 서버를 함께 사용해서 성능에 영향을 미친다면 당신은 해당 업체에 호스팅을 의뢰하고 싶지 않을 것이다. 가상화는 각 고객에게 독립된 가상 머신 하나를 제공하며, 가상 머신 사이에서 완벽하게 격리된 환경을 보장해 주므로 고객 각각의 요구사항을 만족시킬 수 있다.[03]

또 다른 장점은 관리의 용이성이다. 가상화되지 않은 환경에서 서버를 점검하거나 업그레이드한다면, 동작 중인 서버의 전원을 끄고 점검 및 업그레이드를 한 뒤 다시 서버 전원을 켜야 한다. 하지만 가상화된 환경에서는 단순히 가상 머신을 다른 물리 서버 머신으로 이주하고 해당 물리 서버를 점검하면 되므로, 서비스 중단없이 원하는 작업을 마무리할 수 있다.

이처럼 처음에는 서버 통합과 관련한 기업 요구 때문에 재조명을 받았던 가상화 기술이지만, 최근에는 클라우드 컴퓨팅이라는 거부할 수 없는 흐름에 맞춰 가상화 기술이 탄생한 이래로 가장 큰 호황을 누리고 있다. IaaSInfrastructure as a Service라는 클라우드 서비스로 분류하는 아마존 EC2Elastic Compute Cloud는 바로 지금 설명하는 가상화 기술을 기반으로 구축되어 사용자는 웹 사이트에서 단순한 클릭 몇 번만으로 수 분 안에 서버에 생성한 가상 머신을 사용할 수 있다.

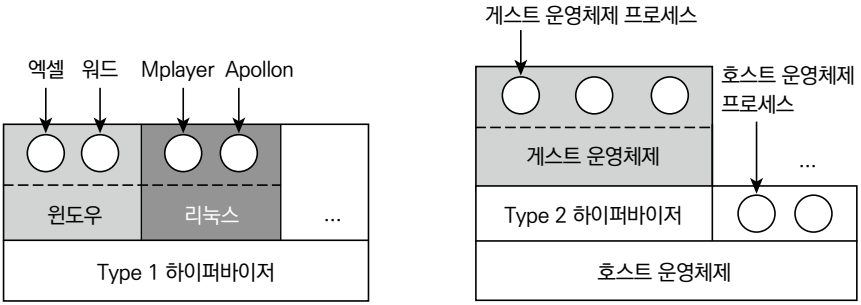
이 외에도 다양한 클라우드 서비스들이 가상화 환경 아래에서 서비스되는 추세며, 앞으로도 수많은 서비스와 애플리케이션이 등장하게 될 것이다.

---

03 물론 운영체제 스스로 프로세스 사이의 간섭을 줄이는 방법들이 있다. 하지만 가상 머신만큼 완벽히 격리된 환경을 제공하지 못한다.

1.2 하이퍼바이저 종류

그림 1-2 Type 1과 Type 2 하이퍼바이저



가상 머신 모니터는 시스템 위치에 따라 크게 Type 1[native or bare-metal]과 Type 2[hosted]로 분류한다. Type 1 하이퍼바이저는 하드웨어 바로 위의 소프트웨어 계층으로 존재하며 그 위에 다양한 게스트 운영체제가 동작하는 방식이다. Xen이나 VMware의 서버용 하이퍼바이저 제품군 등이 여기에 해당한다. Type 1은 하드웨어 전원이 들어오면 가장 먼저 하이퍼바이저가 부팅을 시작하게 하고, 부팅을 완료하면 관리자가 가상 머신을 생성해서 여러 개의 가상 머신이 동작한다.

Type 2는 이와 조금 다르게 호스트 운영체제가 존재하고, 그 위에서 하이퍼바이저가 동작하며, 다시 그 위에 게스트 운영체제가 동작한다. 버추얼박스VirtualBox나 KVM, 그리고 VMware의 데스크톱을 위한 제품군인 워크스테이션workstation 계열이 여기에 속한다. 동작 방식도 조금 다른데 우선 호스트 운영체제를 가장 먼저 부팅해 실행하고, 그 위에서 사용자가 하이퍼바이저를 실행시킨다. 그런 다음 실행된 하이퍼바이저가 여러 개의 가상 머신을 생성하고 동작하게 한다.

어떤 종류의 하이퍼바이저가 좀 더 좋고 효율적인지는 여러 가지 이견들이 있으니 관심 있는 독자는 개별적으로 찾아보기 바란다.

이미 언급한 바와 같이 Xen 하이퍼바이저는 Type 1에 해당하며 다른 여러 하이퍼바이저와 비교해 완성도가 매우 높고 성능도 뛰어나다고 알려졌다. 또한 아마존 웹 서비스Amazon Web Service에서 Xen을 기본 하이퍼바이저로 채택해 사용하는 만큼, 안정성도 이미 검증되었다고 볼 수 있다.

2 | CPU 가상화

2장에서는 컴퓨터 시스템 전체를 가상화하는데 필수 요소 중 하나인 CPU 가상화를 알아본다. ARM, Sparc, 그리고 PowerPC 등 CPU의 종류는 많다.

하지만 여기에서는 가상화 시스템에서 가장 많이 사용하는 x86 아키텍처를 중심으로 설명한다. x86은 가상화와 클라우드에 맞추어 서버 시장에서 주목하는 아키텍처다.

2.1 에뮬레이션과 직접 실행

CPU 가상화를 이해하려면 시스템을 가상화하는 두 가지 방식의 차이점을 알아야 한다. 일반적으로 사람들은 가상화를 에뮬레이션Emulation 기법으로 많이 생각한다. 예를 들어 게임기 에뮬레이터는 게임기를 소프트웨어적으로 완벽하게 모방하여 PC에서 그대로 실행시킬 수 있다.

또한 이와 비슷한 예로 안드로이드 SDK에 포함된 에뮬레이터를 생각해 보자. 안드로이드 개발자는 실제 디바이스가 없더라도 자신이 만든 프로그램을 에뮬레이터를 통해 PC에서 실행할 수 있다.

그렇다면 ARM 아키텍처를 기반으로 만들어진 안드로이드 플랫폼을 어떻게 x86 아키텍처인 PC에서 실행할 수 있을까? 에뮬레이터는 안드로이드 ARM 바이너리 이미지를 스캔해 역어셈블 과정으로 ARM 명령어를 추출해 낸다. 그런 다음 추출해 낸 ARM 명령어를 x86에서 동작할 수 있도록 번역Translation하여, 소프트웨어로 만들어진 가상 머신 위에서 동작하게 한다.

그림 2-1 에뮬레이터(좌: MAME, 우: 안드로이드)

에뮬레이션은 위와 같이 번역의 과정을 거치므로 당연히 실제 머신이나 디바이스에서 실행하는 것보다 성능이 저하된다. 하지만 만약 에뮬레이션해야 하는 대상 아키텍처와 실제 구동시킬 머신 혹은 디바이스의 아키텍처가 같을 때도 이런 불필요한 번역 과정이 필요할까?

이런 단점을 극복하고자 사용하는 방식이 바로 직접 실행Direct Execution 방식이다. 바이너리 프로그램을 번역 과정 없이[01] 직접 CPU에서 실행하는, 즉 가상화 환경이 아닌 것처럼 바이너리 프로그램을 CPU가 직접 실행하게 만드는 것이다. 이렇게 하면 불필요하게 번역할 필요가 없으므로 가상화하지 않은 환경과 같은 성능을 보장하는 동시에 가상화를 통한 다양한 이점을 누릴 수 있다. 그러나 직접 실행 방식으로 하이퍼바이저를 구현할 때는 여러 가지 해결해야 할 문제점이 있다. 다음 장부터는 이 문제점을 설명하고 어떻게 문제점을 해결하는지 알아본다.

2.2 특권 모드와 비특권 모드

직접 실행 방식으로 하이퍼바이저를 구현할 때 발생하는 문제점을 알아보기 전, 특권 모드Privileged mode와 비 특권 모드Non privileged mode를 알아보기로 하자. 요즘 운영체제는 모두 특권 모드라고 불리는, 특권을 갖는 실행 모드가 있다(비특권 모드

01 물론 번역 과정이 전혀 없는 것은 아니다. 디바이스 에뮬레이션과 같이 부분적으로 에뮬레이션을 사용한다.

에서 실행되는 프로그램은 특권 모드의 프로그램과 데이터에는 접근할 수 없다). 일반적으로 응용 프로그램은 비특권 모드에서 실행되고, 운영체제 커널은 특권 모드에서 실행된다. 즉, 애플리케이션은 운영체제가 사용하는 데이터나 코드에 접근할 수 없으므로 운영체제는 시스템의 중요한 자원을 애플리케이션으로부터 보호할 수 있다. 프로그래밍할 때 흔히 볼 수 있는 에러 메시지인 'Segmentation Fault'가 바로 여기에 해당하며, 곧바로 애플리케이션이 종료되므로 전체 시스템에는 큰 영향을 미치지 않는다.[02]

그림 2-2 특권 모드와 비특권 모드

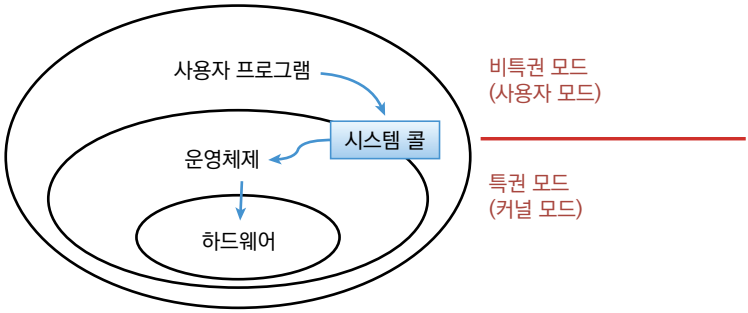


그림 2-2는 시스템 콜을 통해 애플리케이션이 운영체제에 접근하는 방법을 보여준다. 운영체제는 시스템 전체를 관장하며, 하드웨어를 보호하고 관리한다. 다시 말해 운영체제는 애플리케이션이 하드웨어에 직접 접근할 수 없게 하고, 원하는 서비스를 시스템 콜을 통해 요청하도록 규정한다. 예를 들어 애플리케이션이 파일에서 데이터를 읽으려면 프로그래머는 read()라는 시스템 콜을 호출하며, 운영체제는 디스크에서 직접 데이터를 읽어 애플리케이션에 전달한다. 따라서 애플리케이션이 직접 디스크에 접근할 수 있다면, 여러 개의 프로그램이 동시에 아무런 조정 없이 디스크에 접근할 것이고, 시스템 오류를 일으킬 것이다. 또한 아무런 제약 없

02 과거 MS-DOS는 이런 특권 모드를 운영체제가 지원하지 않았으므로 사용자가 조금만 프로그램을 잘못 작성하더라도 운영체제의 데이터에 쉽게 접근할 수 있어 시스템 전체가 다운되는 현상이 있었다.

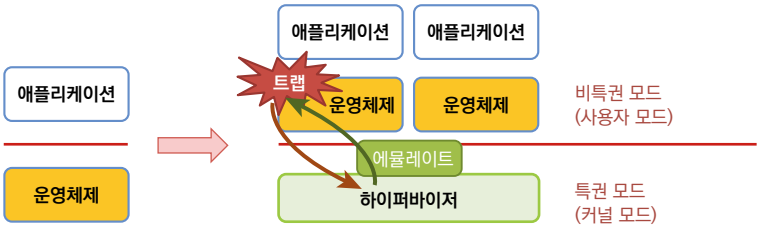
이 운영체제가 사용하는 메모리 영역이나 다른 프로세스가 사용하는 메모리 영역에 접근한다면 어떻게 될까? 역시 시스템 오류를 일으킬 것이다. 즉, 특권 모드와 비특권 모드를 명확하게 구분하지 않으면 악의적인 프로그램 사용 혹은 프로그래밍 실수 때문에 시스템 전체의 안정성을 위협할 것이다.

2.3 특권 명령 및 트랩

특권 명령은 특권 모드에서만 실행 가능한 명령어를 의미하며, 트랩trap은 0으로 나누기나 디버거를 위한 중단점break point 등과 같이 특정 이벤트를 만나면 프로그램의 제어권이 트랩 핸들러(트랩 사건을 처리하는 루틴)로 넘어가는 것을 의미한다. 실제로 비특권 모드에서 특권 명령을 실행할 때 시스템에 트랩이 발생한다. 예를 들어 x86 아키텍처의 (입·출력) 관련 명령어는 특권 명령이므로 사용자 프로세스(비특권 모드)가 해당 명령어를 실행하면 트랩이 발생해 운영체제(특권 모드)의 트랩 핸들러로 제어권이 넘어가고 프로세스는 에러가 발생한다. 따라서 사용자 프로세스는 직접 입·출력 명령어를 사용해서 디바이스를 제어할 수 없고, 위에서 설명한 시스템 콜을 보내 운영체제에서 명령어를 실행하게 한다.

2.4 전통적인 하이퍼바이저 구현 방법

그림 2-3 트랩과 에뮬레이트



그럼 어떻게 하이퍼바이저를 구현할 수 있을까? 전통적인 하이퍼바이저는 그림 2-3과 같은 트랩과 에뮬레이트trap and emulate 방식을 통해 구현한다. 그리고 하이퍼

바이저는 하드웨어 하나에 여러 개의 운영체제를 동작하게 하는 것이 주된 의무이므로 운영체제보다 더 높은 특권을 가져야 한다. 운영체제와 하이퍼바이저가 모두 특권 모드에서 동작한다고 생각해보자. 여러 개의 운영체제가 동시에 특권 명령으로 하드웨어 자원에 접근할 수 있어 시스템 오류를 가져올 것이다. 이런 단점을 극복하려고 하이퍼바이저는 특권 모드에서, 운영체제는 비특권 모드에서 실행한다. 즉, 비특권 모드에서 동작하는 운영체제가 특권 명령을 실행할 때마다 트랩이 발생해 하이퍼바이저로 제어권이 넘어가며, 하이퍼바이저는 적절히 관련 루틴을 에뮬레이트하고 다시 제어권을 운영체제로 넘겨 준다.

과거 프로세서 아키텍처는 위 방식으로 쉽게 구현했다. IBM VM/370이 이에 해당하며 현대 하이퍼바이저의 토대가 되었다. 그런데 트랩 그 자체로는 시스템 오버헤드가 매우 컸고 무엇보다 x86 아키텍처는 트랩과 에뮬레이트 방식만으로 하이퍼바이저를 구현하기가 불가능했다. 특권 명령과 비특권 명령의 경계가 모호한 명령어가 있었기 때문이다. 대표적인 예가 POPF 명령어인데, ALU 플래그(예: Zero 플래그)와 시스템 플래그(예: 인터럽트 금지 및 활성화)를 동시에 수정할 수 있는 명령어다. 사용자 프로세스가 함부로 시스템 플래그를 수정하면 안 되므로, 이 명령어는 비특권 모드에서 실행하면 시스템 플래그는 변하지 않고 ALU 플래그만 변하게끔 설계했다. 예를 들어 게스트 운영체제가 인터럽트를 금지하려고 POPF 명령어를 실행한다고 가정하자. 그럼 비특권 모드에서 실행 중인 게스트 운영체제는 인터럽트 금지 명령을 무시해 트랩을 발생하지 않고 실행을 지속한다. 그러면 운영체제 스스로 오류에 빠져 하이퍼바이저가 운영체제를 적절히 동작시킬 수 없게 된다. 이 명령어 외에도 약 17개 명령어가 트랩과 에뮬레이트 방식으로 하이퍼바이저를 구현하기 어렵게 만든다고 알려졌다.[03]

---

03 Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, John Scott Robin, 2000

2.5 바이너리 변환과 하이퍼 콜

위에서 제기한 문제를 해결하는 데는 두 가지 방법이 있다. 하나는 VMware가 제시한 바이너리 변환Binary Translation이라는 기술이고 다른 하나는 Xen에서 사용하는 하이퍼 콜Hypercall이다. 바이너리 변환은 운영체제의 코드 이미지를 하이퍼바이저가 스캔해 위와 같이 문제의 소지가 있는 명령어를 찾아내고 검출한다. 그리고 해당 명령어를 적절히 수정하여 잘 동작할 수 있는 코드를 만들어 실행한다. 이는 CPU에서 직접 실행하는 방식이지만 중간에 하이퍼바이저가 번역하는 과정이 추가된다.

하이퍼 콜은 반가상화 기술의 일종이며 Xen에서 사용하는 방식으로 번거롭지만 직접 운영체제의 소스를 수정하여 문제의 소지가 있는 명령어를 제거한다. 그리고 하이퍼바이저로 호출할 수 있는 하이퍼 콜로 바꾸어서 문제의 소지가 있는 명령어가 실행될 시점에 하이퍼바이저로 제어권이 넘어가게 한다. 하이퍼바이저는 게스트 운영체제의 요청을 받아서 적절히 처리한 다음 다시 게스트 운영체제로 제어권을 넘겨준다.

2.6 하드웨어 지원

트랩과 에뮬레이트 방식이 불가능한 아키텍처의 근본적인 성질 때문에 바이너리 변환이나 하이퍼 콜과 같은 여러 가지 소프트웨어 솔루션이 등장했다. 하지만 가상화에 대한 요구가 점점 증가하면서 벤더는 저마다 가상화 기능을 CPU에 추가시켰다. 이를 인텔은 VT-x라고 부르고, AMD는 SVMSecure Virtual Machine이라고 부른다. 최근에는 임베디드 시스템에서 주로 사용하는 ARM 프로세서에도 가상화 기능이 추가되었다. 벤더마다 조금씩 아키텍처가 다른 점은 있겠으나, 전체적인 개념은 대부분 비슷하다. 즉, 게스트 운영체제의 소스 코드를 수정하거나 바이너리 변환 방법 없이도 트랩과 에뮬레이트 방식으로 하이퍼바이저를 구현할 수 있도록 하드웨어적인 기능을 추가했다.

그림 2-4 인텔 VT-x

그림 2-4는 인텔 VT-x의 핵심 부분을 도식화한 그림이다. 살펴보면 기존 특권 모드와 비특권 모드 외에 추가로 하이퍼바이저를 위한 root 모드와 게스트 운영체제를 위한 non-root 모드가 등장했다. 즉, 게스트 운영체제는 non-root 모드에서 동작하고 하이퍼바이저는 root 모드에서 동작한다. 또한 게스트 운영체제는 non-root 모드의 특권 모드에서 동작하고, 애플리케이션은 non-root 모드의 비특권 모드에서 동작한다. 그리고 이미 설명했던, 문제의 소지가 있는 명령어가 non-root 모드에서 실행되면 트랩이 발생하고 root 모드로 변경되어 하이퍼바이저로 제어권이 넘어가도록 하드웨어적으로 구현했다. 따라서 POPF 같은 명령어는 자동으로 트랩이 발생하며, 바이너리 변환이나 하이퍼 콜 등의 도움 없이 해당 명령어를 적절히 처리할 수 있다.

3 | Paravirt Operation과 하이퍼 콜

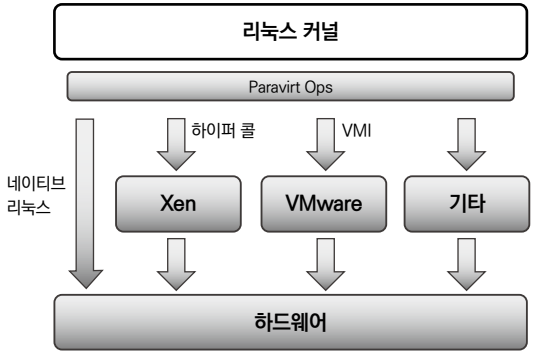
3.1 반가상화

반가상화Para Virtualization 기법은 운영체제의 소스 코드를 수정해서 가상화를 구현한다. 이를 통해 운영체제와 하이퍼바이저 사이의 협력을 극대화해 가상화 시스템의 고성능화를 실현한다. Xen은 하이퍼 콜 인터페이스를 통해서 가장 먼저 반가상화 기법을 널리 알렸으며 이런 흐름에 발맞추어 VMware에서도 VMI 인터페이스를 제공한다. 최근에는 KVM도 반가상화 인터페이스를 제공한다.

3.2 Paravirt Operation

이렇듯 가상화 솔루션 벤더마다 반가상화 인터페이스를 따로 만들어 리눅스 커널을 수정하다 보니, 소스 코드의 복잡성과 관리에 문제가 생겼다. 그렇다고 표준 인터페이스를 만들 수 있는 상황도 아니므로, 리눅스에서 공통 API를 제공하고 각 하이퍼바이저 벤더는 여기에 맞춰서 인터페이스를 구현하도록 했다. 이것이 바로 Paravirt Operation[Pv-ops]이다.

그림 3-1 Paravirt Operation



리눅스 커널의 가상화는 필요한 루틴에 공통 API를 만들어, 여기에 하이퍼 콜 혹은 VMI 인터페이스를 연결하는 방식으로 구현했다. 최초에 Xen은 리눅스 2.6.18 커널 버전을 대폭 수정하여 Xen Linux를 만들었고, 이를 리눅스 paravirt operation에 맞게 포팅했다. 제러미 피츠하딩Jeremy Fitzharding과 콘라드 윌크Konrad Rzeszutek Wilk가 구현한 이 커널을 Pv-ops 커널이라고 부른다. 현재 많은 기능 대부분이 리눅스 커널의 mainline에 포함되어 있다.[01]

코드 3-1 Paravirt operation 자료구조 〈($LINUX)/arch/x86/include/asm/paravirt_types.h〉

```
...
struct pv_cpu_ops {
    unsigned long (*get_debugreg)(int regno);
    void (*set_debugreg)(int regno, unsigned long value);

    void (*clts)(void);

    unsigned long (*read_cr0)(void);
    void (*write_cr0)(unsigned long);

    unsigned long (*read_cr4_safe)(void);
    unsigned long (*read_cr4)(void);
    void (*write_cr4)(unsigned long);

    ...
};

...
struct pv_irq_ops {
    struct paravirt_callee_save save_fl;
    struct paravirt_callee_save restore_fl;
```

01 http://wiki.xen.org/xenwiki/XenParavirtOps

```
    struct paravirt_callee_save irq_disable;
    struct paravirt_callee_save irq_enable;

    void (*safe_halt)(void);
    void (*halt)(void);
#ifdef CONFIG_X86_64
    void (*adjust_exception_frame)(void);
#endif
};

…
struct pv_mmu_ops {
    unsigned long (*read_cr2)(void);
    void (*write_cr2)(unsigned long);

    unsigned long (*read_cr3)(void);
    void (*write_cr3)(unsigned long);

    void (*activate_mm)(struct mm_struct *prev, struct mm_struct *next);
    void (*dup_mmap)(struct mm_struct *oldmm, struct mm_struct *mm);
    void (*exit_mmap)(struct mm_struct *mm);

    void (*flush_tlb_user)(void);
    void (*flush_tlb_kernel)(void);
    void (*flush_tlb_single)(unsigned long addr);
    void (*flush_tlb_others)(const struct cpumask *cpus,
                             struct mm_struct *mm,
                             unsigned long start,
                             unsigned long end);

    …
};
```

```
...
struct paravirt_patch_template {
    struct pv_init_ops pv_init_ops;
    struct pv_time_ops pv_time_ops;
    struct pv_cpu_ops pv_cpu_ops;
    struct pv_irq_ops pv_irq_ops;
    struct pv_apic_ops pv_apic_ops;
    struct pv_mmu_ops pv_mmu_ops;
    struct pv_lock_ops pv_lock_ops;
};

...
```

paravirt_patch_template이라는 구조체는 기능별로 세부 ops 구조체를 가지며 이중 pv_cpu_ops 구조체는 CPU와 직접적인 관련이 있는 명령의 모임이다. 예를 들어 read/write cr0, cr4 같은 명령어는 Paravirt operation으로 대체할 수 있다. CR0와 CR4 레지스터는 x86 아키텍처에서 주요한 시스템 플래그를 모아놓은 콘트롤 레지스터Control Register이고, 페이징 활성화, 캐시 활성화, 세그멘테이션 활성화 등 주요 설정을 할 수 있는 레지스터다. pv_irq_ops 구조체는 인터럽트와 관련이 있는 작업operation 집합이다. 게스트 운영체제 마음대로 인터럽트를 활성화/비활성화하면 안 되므로, 이 역시 Paravirt operation으로 대체되어야 한다. 또한 halt와 같이 시스템을 중지하는 명령어 역시 여기에 해당한다. pv_mmu_ops는 메모리와 관련된 작업 집합으로, CR2 레지스터는 페이지 폴트가 발생하면 페이지 폴트를 발생시킨 명령어의 주소를 저장한다. 나중에 페이지 폴트를 처리한 뒤 돌아갈 명령어의 주소를 알아낼 때 사용한다. CR3 레지스터는 페이지 테이블 베이스 레지스터로 하이퍼바이저는 반드시 페이지 테이블 베이스 레지스터의 변경을 알아야 하므로 역시 Paravirt operation으로 대체해야 한다. 이외에도 TLB를 플러쉬하는 명령어도 Paravirt operation으로 대체된다.