Hanbit eBook

Realtime 21

Xen으로 배우는

가상화 기술의 이해

메모리 가상화

박은병, 김태훈, 이상철, 문대혁 지음

한빛미디어 Hanbit Media, Inc.

Xen으로 배우는

가상화 기술의 이해
메모리 가상화

Xen으로 배우는 **가상화 기술의 이해 – 메모리 가상화**

초판발행 2013년 3월 15일

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 978-89-6848-605-0 15000 / **정가** 9,900원

책임편집 배용석 / **기획** 김창수 / **편집** 이중민, 이세진
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 박진희
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 프로그래머다. (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 메모리 가상화는 시리즈 두 번째 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 라인별로 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. 가상화 환경에서 가상 주소를 물리 주소 변환하는 대표적인 두 가지 방식인 섀도 페이징과 CPU에서 지원하는 HAP(Hardware Assisted Paging)이 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.

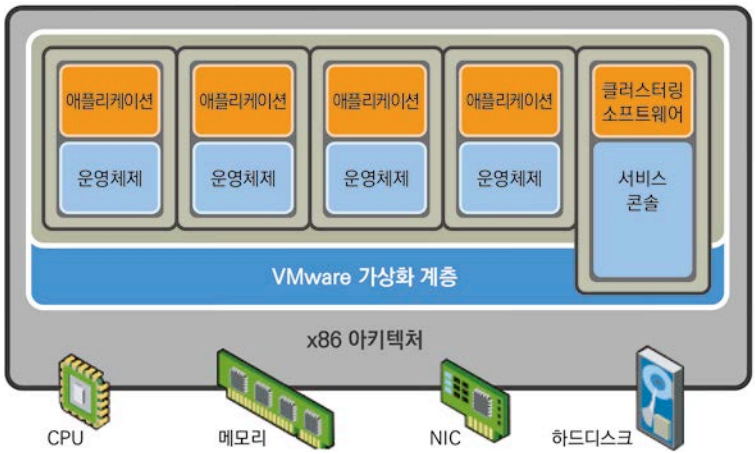
차례

1 | 가상 머신 모니터는 무엇인가?

Xen 내부 동작을 알아보기 전에는 가상 머신 모니터 혹은 하이퍼바이저Hypervisor[01]에 대한 개념과 메모리 가상화에 대한 전반적인 내용을 알아야 한다. 1장과 2장은 메모리 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)



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

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

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.

웹 서버, 데이터베이스 서버, DNS 서버 등 여러 서버를 하나의 머신 위에 동작시켜도 된다. 하지만 운영체제 위에서 여러 개의 서버 프로세스가 동작하게 되므로 서버 프로세스 사이에 많은 영향을 미친다는 치명적인 문제점이 있다.

또 다른 예로 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.

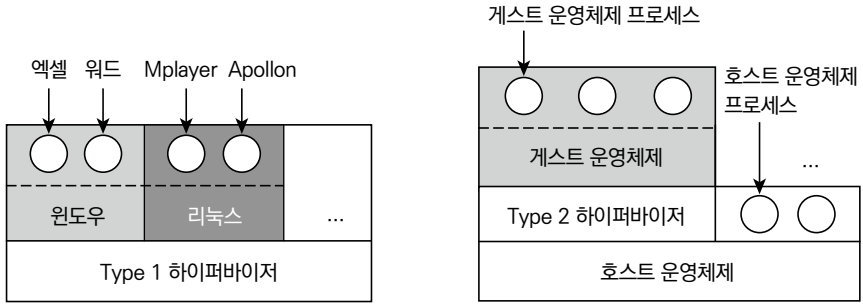
이 외에도 다양한 클라우드 서비스들이 가상화 환경 아래에서 서비스되는 추세며, 앞으로도 수많은 서비스와 애플리케이션이 등장할 것이다.

---

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. 버추얼박스[VirtualBox]나 KVM, 그리고 VMware의 데스크톱을 위한 제품군인 워크스테이션[workstation] 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 | 메모리 가상화

2장에서는 메모리 가상화의 개념을 알아본다. 여기서 말하는 메모리 가상화란 가상 메모리Virtual Memory를 가상화한다는 말과 같다. 현대 운영체제의 대부분은 가상 메모리를 사용한다. 따라서 메모리 가상화를 이해하려면 가상 메모리의 개념을 반드시 깊게 이해해 두어야 한다.

물론 특정 목적을 위한 RTOSReal Time Operating System는 가상 메모리를 사용하지 않을 때도 있다. 하지만 우리가 염두에 두는 운영체제는 윈도우, 리눅스, 유닉스와 같은 가상화 환경에 많이 사용하는 범용 운영체제General Purpose Operating System를 대상으로 한다는 점을 기억하자. 가상 메모리를 가상화한다는 말이 무슨 말인지 의아해하는 독자도 있을 텐데 지금부터 차근차근히 공부해 보도록 하자.

2.1 가상 메모리

가상 메모리란 용어 그대로 메모리를 가상화하는 방법이다. 가상 메모리가 등장한 가장 큰 배경은 물리 메모리Physical Memory가 너무 작았기 때문이었으므로, 마치 메모리가 더 많은 것처럼 보여주려고 사용했다. 그러나 요즘은 물리 메모리의 가격이 굉장히 저렴하고 수십 GB까지 장착할 수 있게 되면서, 메모리 부족 현상 때문에 가상 메모리를 사용한다는 말은 이제 옛이야기가 되어버렸다.

그렇지만 가상 메모리를 사용하는 이유에는 독립 주소 공간, 요구 페이징Demand Paging과 같은 다른 중요한 이유가 있으며, 이에 대한 자세한 설명은 이 책의 범위를 벗어나므로 생략했다.

그림 2-1 가상 메모리

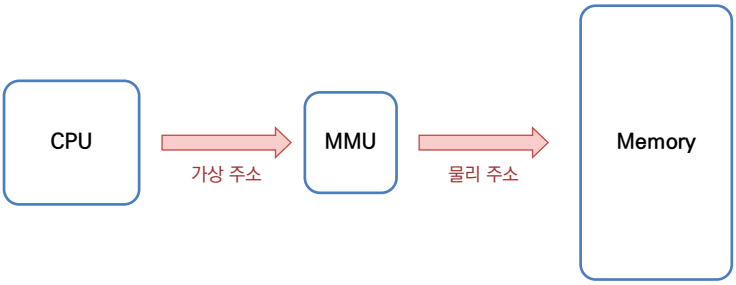


그림 2-1은 가장 기본적인 가상 메모리 구조다. 그림을 살펴보면 가상 메모리를 구현한다는 것은 결국 메모리 주소를 속이는 것으로 볼 수 있다.

CPU가 내보내는 메모리 주소, 즉 운영체제나 애플리케이션이 사용하는 주소는 실제 메모리의 물리 주소Physical Address가 아닌 가상 주소Virtual Address다. 따라서 가상 메모리를 구현하려는 CPU 제조 업체는 저마다 하드웨어 관점에서 가상 주소를 물리 주소로 변환하는 MMUMemory Management Unit라는 것을 제공한다.

가상 주소와 물리 주소와의 매핑을 결정하는 것은 페이지 테이블Page Table이다. 운영체제는 가상 주소와 물리 주소 사이의 매핑 정보를 페이지 테이블에 기록한다. 그런 다음 CPU의 페이지 테이블 베이스 레지스터Page Table Base Register가 페이지 테이블에 기록한 메모리 주소를 등록하면, MMU는 페이지 테이블을 검색해 CPU가 내보내는 가상 주소의 실제 물리 주소를 찾아내 변환하고, 변환한 주소를 통해 메모리에 접근한다. 즉, 페이지 테이블은 가상 주소와 물리 주소 사이의 변환 테이블이다.

그림 2-2 페이지 테이블 구조

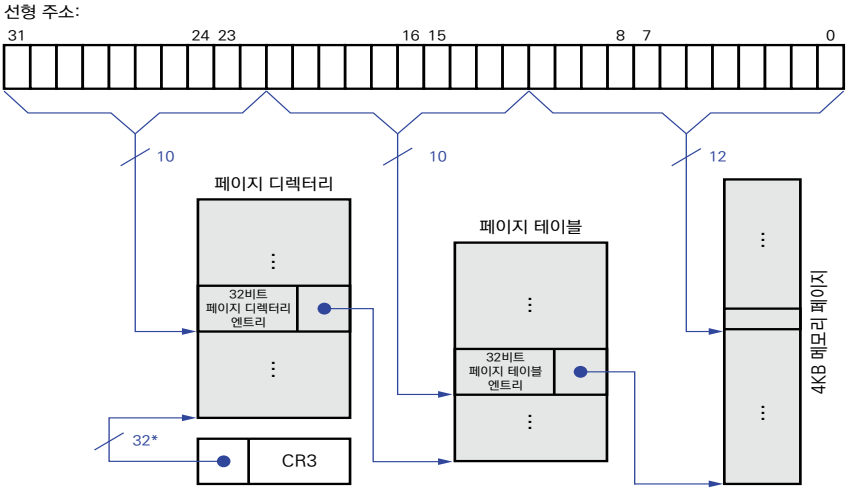


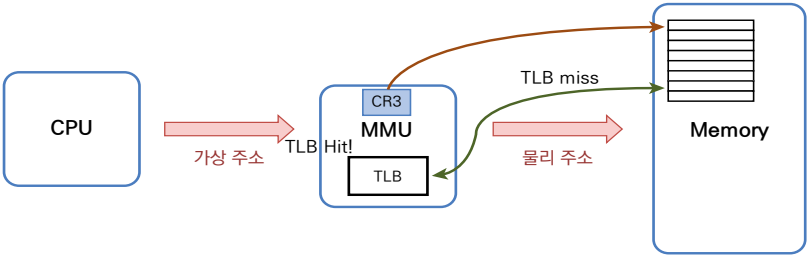
그림 2-2는 x86 아키텍처 32비트 모드의 기본적인 페이지 테이블 구조다. CPU는 32비트 주소를 사용하고, 페이지 디렉터리의 시작 주소는 CR3 레지스터에 저장되어 있다. 그리고 32비트 메모리 주소 중 31~22비트까지의 10비트는 페이지 디렉터리의 인덱스로 사용하고, 해당 인덱스가 가리키는 페이지 디렉터리의 엔트리는 페이지 테이블의 시작 주소를 나타낸다.

마찬가지로 32비트 메모리 주소 중 21~12비트까지는 페이지 테이블의 인덱스로 사용하고, 해당 인덱스가 가리키는 페이지 테이블의 엔트리는 실제 물리 메모리 한 페이지의 시작 주소가 된다. 또한 10비트의 인덱스로 2^{10}인 1,024개의 엔트리를 나타낼 수 있고, 32비트 머신에서 엔트리 하나는 4바이트를 차지하므로 페이지 디렉터리나 페이지 테이블 하나는 각 4KB의 페이지 크기만큼 메모리 공간을 차지하게 된다. 마지막으로 남은 11~0비트까지의 12비트는 4KB 페이지 안의 오프셋을

나타내고, 찾고자 하는 최종 물리 주소가 된다.[01] 12비트로 나타낼 수 있는 주소 범위는 2^{12}인 4KB가 되므로 4KB로 페이지 안의 모든 주소를 가리킬 수 있다.

이렇게 복잡한 다단계 구조로 페이지 테이블을 구성한 까닭은 MMU 구현의 효율성과 페이지 테이블의 크기와 관련 있다. 페이지 테이블 하나를 구성한다면, 1개의 페이지 크기가 4KB인 페이지 테이블은, $2^{32}/2^{12}$인 1,048,576개의 엔트리가 필요하다. 즉, 엔트리 하나의 크기는 4바이트이므로 총 4MB의 공간이 필요하다. 페이지 테이블은 프로세스마다 독립적으로 하나씩 존재하므로 모든 프로세스에 이렇게 4MB의 배열을 메모리에 할당하는 일은 매우 비효율적인 방식이다. 또한 모든 프로세스는 4GB의 주소 공간 중 극히 일부분만을 사용하므로 주소 공간이 필요한 부분만 페이지 테이블을 만들어 할당할 수 있는 다단계 방식을 취했다.

그림 2-3 Translation Lookaside Buffer



CPU 안의 레지스터 혹은 캐시 메모리 접근과 메모리 접근 속도의 차이는 엄청나게 크다. 따라서 매번 가상 주소를 물리 주소로 변환하려고 메모리의 페이지 테이블에 접근하는 일은 시스템 성능을 상당히 저하시킨다. 또한 운영체제나 애플리케이션이 특정 순간 접근하는 메모리 주소는 일정 범위로 한정된 경우가 많으므로, 한 번 변환했던 주소는 캐시 메모리에 저장했다가 다음번에 같은 주소 변환 요청

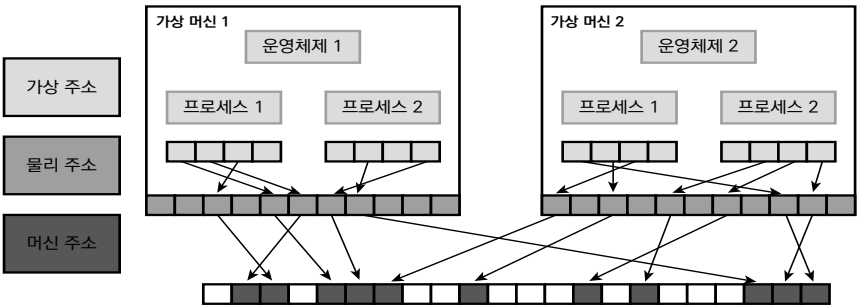
01 페이지 테이블을 구성하는 메모리 주소의 비트 구조는 아키텍처와 페이지 모드마다 다르다.

이 일어나면 메모리에 접근하지 않고 캐시 메모리에 저장한 주소를 가져다가 사용한다. 이것을 TLB[Translation Lookaside Buffer], 변환 색인 버퍼라고 부르며, 실제 메모리 주소 변환은 TLB를 통해서 이루어진다. 그림 2-3은 기본 페이지 테이블 구조에 TLB가 추가된 그림이다. TLB에 CPU가 요청한 가상 주소에 대한 엔트리가 이미 있다면 TLB에 있는 엔트리를 그대로 사용하고, 그렇지 않다면 메모리에 접근해 페이지 테이블 엔트리를 가져와서 TLB에 저장한 후, 해당 엔트리를 가져와서 사용한다.

2.2 가상화 환경의 메모리 주소 체계

비가상화 환경에서 물리 주소와 가상 주소라는 두 단계 주소 체계를 가진다면, 가상화 환경에서는 한 단계가 더 추가된다. 가상 주소, 물리 주소, 머신 주소[Machine Address]라는 세 단계로 부르거나 가상 주소, 가상 물리 주소[Virtual Physical Address], 물리 주소의 세 단계로 부르기도 한다.

그림 2-4 가상 머신 메모리 주소 체계



가상 머신에서 동작하는 게스트 운영체제는 가상화 환경에서 동작한다는 사실을 모른다. 그리고 가상 머신은 가상화되지 않은 환경에서 동작할 때와 마찬가지로 가상 주소와 물리 주소를 그대로 사용한다. 하지만 가상 머신이 아는 물리 주소는 실

제 물리 주소가 아니며, 하이퍼바이저의 중재 아래 실제 물리 주소로 변환해서 사용해야 한다. 그림 2-4는 가상 머신 두 개가 동작한다. 운영체제 내부의 각 프로세스는 각각의 가상 주소 공간을 사용하며 운영체제가 관리하는 페이지 테이블에 의해 물리 주소로 변환된다. 하지만 운영체제가 생각하는 물리 주소는 실제 물리 주소가 아니며, 하이퍼바이저가 중재한 실제 메모리에 접근할 수 있는 머신 주소로 변환된다.

아직 이해하지 못할 수 있는 독자를 위해 반대로 가상 머신이 사용하는 물리 주소를 그대로 사용한다고 생각해보자. 기본적으로 운영체제는 할당된 메모리가 0번지부터 시작한다고 생각해 동작한다. 그러면 모든 가상 머신은 너도나도 0번지부터 시작하는 메모리를 사용할 것이고, 모두 같은 영역의 메모리를 사용하므로 충돌이 일어나서 정상적으로 동작할 수 없을 것이다. 따라서 하이퍼바이저의 적절한 중재가 있어야 한다. 예를 들어 512MB의 메모리를 가진 가상 머신 두 개를 동작시킨다고 하면, 첫 번째 가상 머신에는 실제 메모리의 0MB~512MB 영역을 할당하고, 두 번째 가상 머신에는 513MB~1,024MB 영역을 할당한다. 그리고 하이퍼바이저는 두 번째 가상 머신의 물리 주소 0번지를 실제 물리 주소, 즉 머신 주소 513MB 영역의 번지로 변환해야 한다.

2.3 섀도 페이지 테이블

가상화 환경의 메모리 주소 체계에서는 게스트 운영체제가 생각하는 물리 주소를 실제 머신 주소로 변환해서 사용해야 하는데, 어떻게 이러한 일이 가능한지 알아보자. 일반적으로 하이퍼바이저에서는 섀도 페이지 테이블Shadow Page Table이라는 가상의 페이지 테이블을 사용한다. 운영체제가 사용하는 페이지 테이블이 가상 주소에서 물리 주소로 변환하는 정보를 저장한다면, 섀도 페이지 테이블은 가상 주소에서 머신 주소로 변환하는 정보를 저장하는 페이지 테이블이다. 하이퍼바이저는 머신 주소 정보를 가지므로, 섀도 페이지 테이블을 구축하고 페이지 테이블 베이스

레지스터에 섀도 페이지 테이블의 시작 주소를 등록한다. 그러면 MMU가 참조하는 페이지 테이블은 섀도 페이지 테이블이 되고, CPU가 생성하는 가상 주소를 머신 주소로 변경해 사용할 것이다.

그림 2-5 섀도 페이지 테이블

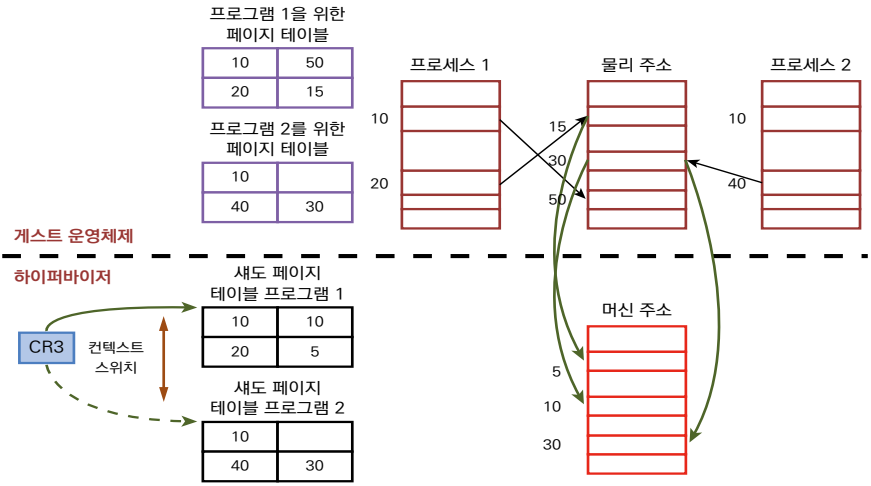


그림 2-5를 살펴보면, 현재 가상 머신 하나가 동작 중이고, 그 위에 프로세스 두 개가 동작한다. 그리고 운영체제는 프로세스마다 페이지 테이블을 운영한다. 또한 하이퍼바이저는 각 프로세스에 해당하는 섀도 페이지 테이블을 유지하며, 페이지 테이블 베이스 레지스터인 CR3는 섀도 페이지 테이블을 가리킨다.

현재 프로세스 1의 가상 주소 10에 해당하는 물리 주소는 50이다. 따라서 게스트 운영체제가 관리하는 페이지 테이블에는 가상 주소 10의 엔트리에 50이라고 쓰여 있다. 하지만 실제 물리 주소 50은 머신 주소 10에 매핑되었고, 프로세스 1의 가상 주소 10에 해당하는 섀도 페이지 테이블 엔트리에는 10이라고 쓰여있다. 페이지 테이블 베이스 레지스터는 섀도 페이지 테이블을 가리키므로, MMU는 가상 주소

10을 머신 주소 10으로 변환할 것이다.

또한 현재 프로세스 1이 동작하므로 페이지 테이블 베이스 레지스터는 섀도 페이지 테이블 1을 가리킨다. 그런데 여기서 게스트 운영체제가 프로세스 2로 컨텍스트 스위치Context Switch한다면 어떻게 될까? 게스트 운영체제는 컨텍스트 스위치할 때, 페이지 테이블 베이스 레지스터에 프로세스 2의 페이지 테이블을 가리키도록 수정한다.

하이퍼바이저는 페이지 테이블 베이스 레지스터를 변경하는 것을 감지[02]할 수 있으므로, 제어권을 넘겨받아 게스트 운영체제 프로세스 2의 페이지 테이블 주소가 아닌, 프로세스 2의 섀도 페이지 테이블을 페이지 테이블 베이스 레지스터에 설정한다. 즉, 위 과정을 반복하면 하이퍼바이저는 게스트 운영체제 모르게 적절히 섀도 페이지 테이블을 사용해 시스템 전체의 메모리 가상화를 구현할 수 있다.

섀도 페이지 테이블의 더 자세한 내용은 6장에서 자세히 살펴보겠다. 6장에는 하이퍼바이저가 섀도 페이지 테이블을 구축하는 방법, 게스트 운영체제의 페이지 테이블과 동기화하는 방법, 좀 더 효율적인 알고리즘 등 재미있는 주제들이 많으므로 Xen 소스 코드 분석과 함께 좀 더 깊이 있게 다루도록 하겠다.

2.4 직접 페이지 테이블 접근

섀도 페이지 테이블은 메모리 가상화에 아주 적합해 보이는 방법이지만 어쩔 수 없이 오버헤드를 가진다. 예를 들면 게스트 운영체제 모든 프로세스의 페이지 테이블을 중복해서 가지며, 항상 게스트 운영체제의 페이지 테이블과 동기화해야 하는 등 성능 저하를 발생시키는 많은 요인이 있다.

그런데 반가상화 기법을 사용하면 게스트 운영체제의 소스 코드를 수정할 수 있으

02 앞서 설명한 바이너리 변환 방법이나 하이퍼 콜, 혹은 Vt-x 같은 하드웨어 지원을 받아서 해당 명령을 실행할 때는 하이퍼바이저에게 제어권을 넘겨준다.

므로 섀도 페이지 테이블을 사용하지 않고도 메모리 가상화를 할 수 있다. Xen에서는 게스트 운영체제를 수정해 게스트 운영체제가 직접 페이지 테이블을 올바르게 수정하도록 변경할 수 있다. 또한 하이퍼 콜로 하이퍼바이저가 페이지 테이블을 수정하도록 요청할 수도 있고, 뒤에 설명하겠지만 직접 페이지 테이블을 수정한 뒤 페이지 테이블이 올바르게 수정되었는지를 하이퍼바이저가 나중에 확인하도록 할 수도 있다.

그림 2-6 직접 페이지 테이블 접근

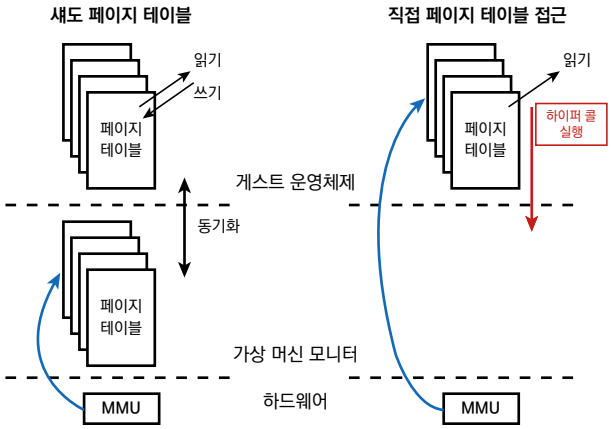


그림 2-6의 왼쪽은 섀도 페이지 테이블을 보여주며, 오른쪽은 직접 페이지 테이블 접근 방식을 보여준다. 섀도 페이지 테이블은 앞서 설명했듯이 MMU가 게스트 운영체제의 페이지 테이블을 참조하는 것이 아니라 하이퍼바이저가 유지하는 섀도 페이지 테이블을 참조한다. 그러나 직접 페이지 테이블 접근 방식에서는 MMU가 직접 게스트 운영체제의 페이지 테이블을 참조한다.

그렇다면 가상 주소와 머신 주소 사이의 주소 변환을 어떻게 반영하는 것일까? 답은 간단하다. 게스트 운영체제는 물리 주소와 머신 주소 사이의 매핑 정보를 가진

다는 점이다. 따라서 직접 페이지 테이블에 주소를 저장할 때, 물리 주소가 아닌 머신 주소를 저장한다. Xen의 경우에는 하이퍼 콜을 통해서 하이퍼바이저에 페이지 테이블 수정을 요청한다. 그러면 하이퍼바이저는 유효한 요청인지 검사한 후 요청을 수락하게 된다.

이렇게 게스트 운영체제의 페이지 테이블은 가상 주소와 물리 주소 사이의 변환 테이블이 아니라 가상 주소와 머신 주소 사이의 변환 테이블을 가진다. 따라서 MMU가 직접 게스트 운영체제의 페이지 테이블을 참조해도 정상적으로 동작하는 것이다. 이는 직접 소스 코드를 수정해야 하는 번거로움이 있다. 하지만 섀도 페이지 테이블과 같이 중복해서 페이지 테이블을 유지할 필요가 없기 때문에 성능상의 이점을 얻을 수 있다.

지금까지 메모리 가상화를 위한 소프트웨어적 솔루션 두 가지를 가볍게 살펴보았다. 두 방법 모두 좋은 접근 방식이기는 하나, 여전히 성능상의 문제점이 있다. 많은 수의 페이지 폴트[03]를 일으키고, 게스트 운영체제와 하이퍼바이저 사이의 컨텍스트 스위치가 자주 발생하게 되므로 이에 따른 성능 저하도 무시할 수 없기 때문이다. 따라서 이런 점을 극복하고자 하드웨어적 접근 방식이 나타나게 되었다.

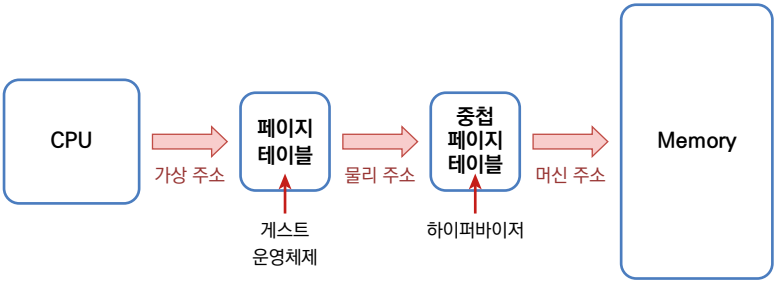
2.5 중첩 페이지 테이블

중첩 페이지 테이블은 AMD가 관련 기능을 먼저 지원하기 시작했고, 인텔도 이에 발맞춰 관련 기능을 추가하기 시작했다. AMD는 중첩 페이지 테이블Nested Page Table, 인텔은 확장 페이지 테이블Extended Page Table이라고 부르는데, 명칭과 세부 기능만 조금 다를 뿐 하는 일은 거의 같다. 기존의 가상 메모리가 가상 주소에서 물리 주소로 변환하는 기능을 하드웨어적으로 구현한 것이라면, 중첩 페이지 테이블은 물리 주소에서 머신 주소로 변환하는 기능을 하드웨어적으로 구현한 것이다.

---

03 가상 메모리에서 필요한 페이지가 메모리에 상주 하지 않는 상태를 말한다.

그림 2-7 중첩 페이지 테이블



즉, CPU가 내보내는 가상 주소는 게스트 운영체제가 구축한 페이지 테이블을 통해서 물리 주소로 변경하고, 변경한 물리 주소는 다시 하이퍼바이저가 구축한 중첩 페이지 테이블을 통해서 머신 주소로 변경해 메모리에 접근한다. 이렇게 되면 게스트 운영체제는 페이지 테이블을 구축할 때 섀도 페이지 테이블이나 직접 접근 방식에서 발생하던 페이지 폴트 같은 부가적인 오버헤드 없이 페이지 테이블을 구축할 수 있다. 즉, 하이퍼바이저가 게스트 운영체제를 생성할 때 할당해준 메모리의 머신 주소 정보만 중첩 페이지 테이블에 적어주면 쉽게 메모리 가상화를 할 수 있다.

또한 섀도 페이지 테이블은 모든 게스트 운영체제의 프로세스에 페이지 테이블을 중복해서 유지해야 하지만, 중첩 페이지 테이블은 가상 머신 당 중첩 페이지 테이블 하나만 있으면 되므로 메모리 공간의 활용 효율성 측면에서도 매우 뛰어나다.