

10th EDITION

운영체제

Operating System Concepts

박민규 옮김



ABRAHAM SILBERSCHATZ
PETER BAER GALVIN
GREG GAGNE

운영체제 Operating System Concepts

10th EDITION

발행일 : 2020년 2월 28일 1쇄 발행

지은이 : Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

옮긴이 : 박민규

발행인 : 손영만

발행처 : 퍼스트 북(First Book)

등록 : 제312-2009-000053호

주소 : 서울시 서대문구 연희동 193-7 영화 B/D 502호

전화 : 02) 3141-2953(주문 및 고객지원)

팩스 : 02) 3141-2955

홈페이지 : www.firstbook.kr

E-mail : first_book@naver.com

ISBN 979-11-85475-57-8 (93560)

값 : 39,000원

이 도서는 퍼스트북과 John Wiley & Sons사와의 한국어판에 대한 출판, 판매, 독점계약에 의해
발행한 것이므로 내용, 사진, 그림 등의 전부나 일부의 무단 복제 및 무단 전사를 일절 금합니다.

Copyright 2020 © John Wiley & Sons, Inc. All rights reserved.

AUTHORIZED TRANSLATION OF THE EDITION PUBLISHED BY JOHN
WILEY & SONS, New York, Chichester, Brisbane, Singapore AND Toronto.
No part of this book may be reproduced in any form without the written
permission of John Wiley & Sons Inc.

저자 서운 *Preface*

운영체제는 모든 컴퓨터 시스템의 필수적인 부분이다. 비슷하게, 운영체제 과목은 모든 컴퓨터 과학 교육의 필수적인 부분이다. 이 분야는 컴퓨터가 자동차의 내장형 장치에서부터 정부나 다국적 기업을 위한 대단히 복잡한 기획 도구와 같은 거의 모든 응용에 이용되면서 빠른 속도로 변화해 왔다. 그렇지만 기본 개념은 비교적 분명히 변하지 않고 있으며 이러한 개념들에 기반을 두고 이 책을 기술하였다.

이 책은 학사과정 3학년 또는 4학년, 또는 대학원 1년 차 수준의 운영체제 개론 교과를 위한 교재를 목표로 하고 있다. 또한 실무 종사자들에게도 이 책이 유용하기를 바란다. 이 책은 운영체제의 근간을 이루고 있는 개념에 대한 명확한 설명을 제공한다. 이 책을 읽기 위해 독자는 기본 자료구조, 컴퓨터 구성, 그리고 C 또는 Java와 같은 고급 언어를 이해하고 있어야 한다. 운영체제를 이해하기 위해 필요한 하드웨어에 관한 내용은 1장에서 다루고 있다. 1장에서는 또한 대부분의 운영체제에서 많이 사용되고 있는 기본적인 자료구조에 대한 개괄도 제공한다. 코드의 범례에는 주로 C 언어를 사용하고 상당량의 Java를 함께 사용한다. 그러나 독자들은 이들 언어에 대한 깊은 이해 없이도 알고리즘들을 이해할 수 있을 것이다.

개념들은 직관적인 설명을 사용하였으며, 중요한 이론적 결과들도 다루어지지만 정형적인 증명은 대부분 생략하였다. 각 장 마지막에 위치한 문헌 노트는 이러한 결과들이 처음 제시되고 증명된 연구 논문들에 대한 포인터 및 추가적인 탐구를 위한 최근 자료들에 대한 참고문헌을 포함하고 있다. 증명을 제공하는 대신 그 결과들이 올바르다고 예상되는 이유를 그림과 예를 사용하여 제시하였다.

이 책에서 다루고 있는 기본 개념과 알고리즘들은 공개 소스 운영체제와 상용 운영체제에서 자주 사용되는 것들에 기반을 두고 선정하였다. 우리의 목표는 이들 개념과 알고리즘들을 특정 운영체제 환경에 종속되지 않는 일반적인 운영체제의 입장에서 제시하는 것이다. 우리는 가장 대중적이면서 혁신적인 운영체제에 국한된 많은 실례를 제공한다. 이 운영체제에는 Linux, Microsoft Windows, Apple macOS 원래 이름인 OS X는 2016년에 다른 Apple 제품의 명명 체계와 일치하도록 변경되었다)와 Solaris 등이 포함된다. 또한 현재 가장 대표적인 모바일 운영체제인 Android와 iOS의 실례도 제시한다.

이 책의 구성은 저자들의 다년간의 운영체제 강의 경험을 반영한다. 이전 판의 독자들과 현재 및 이전 학생들로부터 받은 많은 의견과 제안과 함께 이 책의 검토자들이 제공한 피드백에 대해서도 고려했다. 이 10판은 IEEE Computing Society와 Association for Computing Machinery (ACM)에서 발행한 컴퓨터 과학 학사 학위 프로그램에 대한 가장 최근의 커리큘럼 지침인 *Computer Science Curricula 2013*의 운영체제 영역에서 대부분의 커리큘럼 지침을 반영한다.

이 판의 새로운 내용

10판에서는 학생들의 비용을 낮추고 학습 과정에 더 효과적으로 참여시키며 강사에 대한 지원을 강화하기 위한 개정 및 개선에 초점을 맞추었다.

출판업계에서 가장 신뢰받는 시장조사기관인 Outsell에 따르면 2015년은 교재 사용의 전환점을 나타냈다. 처음으로 학생들이 디지털 학습 자료에 대한 선호도가 인쇄본 보다 높았고, 그 이후로 디지털에 대한 선호도의 증가가 가속화되고 있다.

교육학 도구로서 많은 학생에게 인쇄본이 여전히 중요한 것으로 남아 있지만 10판은 디지털 자료를 통한 학습 지원을 강조하는 형태로 제공되고 있다. 우리가 제공하는 모든 양식은 9판과 비교하여 학생들에게 비용을 획기적으로 절감한다. 이러한 형태는 다음과 같다.

- **대폭 향상된 독립형 전자 텍스트.** 10판의 전자 텍스트 형식은 주요 섹션의 끝에 새로운 자체 평가 문제, 주요 용어에 대한 정의 숨기기/표시 및 여러 애니메이션 그림을 추가한다. 또한 장마다 해답이 제공되는 추가적인 “연습문제”, 정규 연습문제, 프로그래밍 문제 및 프로젝트, “추가 자료” 섹션, 완전한 용어집 및 레거시 운영체제에 대한 4개의 부록이 포함된다.
- **인쇄물이 함께 제공되는 전자 텍스트.** 얼마 되지 않는 추가 비용을 내면, 요약된 인쇄물이 함께 제공되는 전자 텍스트가 가능하다. 이 요약본은 본 주요 본문 내용, 장의 끝에 있는 “연습문제” 및 “추가 자료”를 포함한다.

우리는 모든 강사 및 학생들이 전자 텍스트 버전의 비용, 내용 및 학습 이점을 활용할 것을 적극적으로 권장하지만, 강사는 Wiley 계정 관리자와 협력하여 맞춤 인쇄판을 만들 수 있다.

이러한 옵션을 자세히 살펴보거나 다른 옵션에 대해 논의하려면 Wiley 계정 관리자 (<http://www.wiley.com/go/whosmyrep>)에게 문의하거나 wiley.com에서 이 교재에 대한 제품 정보 페이지를 방문하시오.

책의 자료

이 책은 21개의 장과 4개의 부록으로 구성되어 있다. 각 장과 부록에는 본문과 다음 개선 사항 중 일부 또는 전부가 포함되어 있다.

- 해답을 포함한 일련의 연습문제
- 정규 연습문제
- 일련의 프로그래밍 문제
- 일련의 프로그래밍 프로젝트
- 추가 자료 섹션
- 중요한(파란색) 용어의 팝업 정의
- 중요한 용어의 용어집
- 특정 핵심 개념을 설명하는 애니메이션

인쇄물이 함께 제공되는 전자 텍스트의 본문 하드카피는 전자 버전과 동일하다. 그러나 부록, 정규 연습, 연습문제에 대한 해답, 프로그래밍 문제, 프로그래밍 프로젝트 및 전자 텍스트에서 볼 수 있는 기타 개선 사항은 포함되지 않는다.

이 책의 내용

이 교재는 10개의 주요 부분으로 구성되어 있다.

- **개관.** 1장과 2장은 우리는 운영체제란 무엇인지, 어떤 일을 하는지, 어떻게 설계되고 구축되는지를 설명한다. 이 두 장은 운영체제의 공통 특징이 무엇인지, 운영체제가 사용자를 위해 무엇을 하는지를 설명한다. 전통적인 PC와 서버 운영체제 및 모바일 장치를 위한 운영체제를 설명한다. 이 장에서는 현실적으로 동기를 부여하고 전반적인 설명을 위주로 주제를 제시하였으며, 제시된 아이디어가 내부적으로 어떻게 구현되었는지에 대한 논의는 피하였다. 그러므로 이 장들은 자세한 내부 알고리즘 공부 없이 운영체제가 무엇인지 배우기를 원하는 전공 지식수준이 낮은 학생이나 독자에 적합하다.
- **프로세스 관리.** 3장부터 5장까지는 현대의 운영체제의 심장부로서 프로세스 개념과 병행성을 다룬다. **프로세스**는 시스템의 작업의 단위로서 시스템은 **병행하게** 실행되는 프로세스의 집합으로 구성되는데 일부는 시스템 코드를 실행하고 나머지는 사용자 코드를 실행한다. 이 장들은 프로세스 스케줄링, 프로세스 간 통신을 위한 방법들을 제시한다. 또한 이 장들에서는 스레드에 관한 논의뿐 아니라 다중 코어 시스템과 병렬 프로그래밍과 관련된 주제에 대한 검토가 이루어진다.

- **프로세스 동기화.** 6장부터 8장까지는 프로세스 동기화 및 교착 상태 처리 방법에 대해 설명한다. 프로세스 동기화의 적용 범위를 늘렸기 때문에 이전 5장(프로세스 동기화)을 6장 동기화 도구 및 7장 동기화 예제의 두 장으로 나누었다.
- **메모리 관리.** 9장과 10장은 프로세스가 실행되는 동안 메인 메모리를 어떻게 관리하는가에 관한 문제를 다룬다. CPU의 효율적인 활용과 사용자에게 빠른 응답 시간을 보장하기 위해 컴퓨터는 한순간에 많은 프로세스를 메모리에 유지해야 하는 것이 필수적이다. 메모리 관리에 관한 다양한 접근 방법을 반영하는 많은 종류의 메모리 관리 기법들이 존재하며 특정 알고리즘의 유효성은 상황에 따라 달라진다.
- **저장장치 관리.** 11장과 12장은 현대 운영체제가 대용량 저장장치와 입출력을 어떻게 처리하는지에 대해 설명한다. 컴퓨터 시스템에 부착되는 입출력 장치들의 종류가 매우 다양하고 응용들이 이러한 장치들의 모든 측면을 관리할 수 있도록 광범위한 기능을 제공해야 한다. I/O 시스템 설계, 인터페이스, 그리고 내부 시스템 구조 및 기능을 포함한 시스템 I/O를 깊이 있게 논의한다. 여러 면에서 입출력 장치는 컴퓨터 시스템의 주요 구성요소 중에서 가장 느리게 동작하는 구성요소이다. 장치들이 성능의 병목 지점으로 작용하기 때문에 입출력 장치들에 관한 성능 문제들도 검토한다.
- **파일 시스템.** 13장부터 15장까지는 최신 컴퓨터 시스템에서 파일 시스템을 처리하는 방법에 대해 설명한다. 파일 시스템은 데이터와 프로그램을 온라인으로 저장하고 접근하기 위한 기법을 제공한다. 이 장들은 저장장치 관리를 위한 고전적인 내부 알고리즘들과 그 구조를 다룬다. 또한 사용되는 알고리즘의 속성, 장점 및 단점에 대한 견고한 실질적인 이해를 제공한다.
- **보안 및 보호.** 16장과 17장은 컴퓨터 시스템의 보안과 보호를 위해 필요한 기법들에 대해 논의한다. 운영체제 내의 다양한 프로세스들은 다른 프로세스들의 활동으로부터 보호되어야 한다. 그런 보호를 제공하기 위해서는 운영체제로부터 적절한 인가를 획득한 프로세스만이 파일, 메모리, CPU 및 다른 자원에 대해 연산을 하도록 보장해야 한다. 보호는 프로그램, 프로세스, 또는 사용자들이 컴퓨터 시스템 자원에 대한 접근을 제어하기 위한 기법이다. 이 기법은 시행될 제어를 지정할 방법과 이들을 강제하려는 방법을 제공해야 한다. 보안은 인가되지 않은 접근, 악의적 파괴 또는 변경, 그리고 비일관성의 우연한 도입으로부터 시스템에 저장된 정보(데이터 및 코드 모두)의 일관성과 시스템의 물리 자원들을 보호한다.
- **고급 주제.** 18장과 19장에서는 가상 머신과 네트워크/분산 시스템에 대해 논의한다. 18장에서는 가상 머신과 그것의 최신 운영체제와의 관계에 대해 개괄적으로 설명한다. 가상화를 가능하게 하는 하드웨어 및 소프트웨어 기술에 대한 일반적인 설명이 포함되어 있다. 19장에서는 인터넷과 TCP/IP를 중심으로 컴퓨터 네트워크 및 분산 시스템에 대한 개요를 제공한다.

- **사례 연구.** 20장과 21장은 두 개의 실제 운영체제인 Linux와 Windows 10에 대한 상세한 사례 연구를 제시한다.
- **부록.** 부록 A는 더는 사용되지 않는 오래된 영향력 있는 운영체제에 대해 설명한다. 부록 B에서 D는 Windows 7, BSD 및 Mach라는 세 가지 이전 운영체제를 자세히 다룬다.

프로그래밍 환경

이 책은 C 언어와 Java 언어로 작성된 여러 예제 프로그램을 제공한다. 이 프로그램들은 다음의 프로그래밍 환경에서 실행되도록 작성되었다.

- **POSIX.** *Portable Operating System Interface*를 의미하는 POSIX는 기본적으로 UNIX 기반 운영체제를 위해 구현된 표준안의 집합을 나타낸다. Windows 시스템 또한 특정 POSIX 프로그램을 실행시킬 수 있지만 POSIX를 설명하는 경우 Linux와 UNIX 시스템에 초점을 맞춘다. POSIX-compliant 시스템은 POSIX 핵심 표준안 (POSIX.1)을 반드시 구현해야 하며 Linux 및 macOS가 대표적인 POSIX-compliant 시스템이다. POSIX는 실시간 확장 (POSIX1.b)와 스레드 라이브러리(POSIX1.c Pthreads로 더 잘 알려져 있음)를 포함하는 여러 확장 표준안도 정의한다. Pthreads와 실시간 프로그래밍을 위한 확장을 포함하여 POSIX 기반 API의 사용법을 설명하는 C 프로그램을 제공한다. 이 예제 프로그램들은 Linux 4.4, macOS 10.11 시스템에서 gcc 컴파일러를 사용하여 검증되었다.
- **Java.** Java는 풍부한 API와 병행 및 병렬 프로그래밍을 위한 기능이 언어 자체에 내장된 널리 사용되는 프로그래밍 언어이다. Java 프로그램은 Java 가상 기계 (JVM)를 지원하는 어떤 운영체제에서도 실행할 수 있다. Java 1.8 Java Development Kit(JDK)을 사용하여 검증된 Java 프로그램들을 통하여 운영체제와 네트워킹의 다양한 개념을 설명한다.
- **Windows 시스템.** Windows 시스템을 위한 기본적인 프로그래밍 환경은 Windows API이다. 이 API는 프로세스, 스레드, 메모리, 그리고 주변 장치들을 관리하기 위해 필요한 모든 함수를 제공한다. 이 API 사용법에 대해서 설명하기 위하여 필요한 C 프로그램들을 제공한다. 이 예제 프로그램들은 Windows 10 시스템에서 검증되었다.

우리가 이 세 프로그래밍 환경을 선정한 이유는 Linux/UNIX와 Windows가 가장 대중적인 운영체제 모델이며 Java 환경 또한 널리 사용되는 프로그래밍 환경이라고 생각하기 때문이다. 대부분의 프로그래밍 예제는 C 언어로 작성되었으며 독자들이 이 언어에 익숙할 것이라고 가정한다. C 언어와 Java 언어에 익숙한 독자들이라면 교재에

나오는 대부분의 프로그램을 쉽게 이해해야 한다.

스레드 생성과 같은 몇몇 예제에서는 특정 개념을 설명하기 위하여 세 프로그래밍 환경의 예제 코드를 모두 보여준다. 이를 통하여 독자들은 동일한 작업을 할 때 세 가지 다른 라이브러리들의 이용을 대조해 볼 수 있을 것이다. 다른 경우에는 개념을 설명하기 위하여 한 가지 API만을 사용하였다. 예를 들면 공유 메모리 개념은 POSIX API만을 사용하여 설명하였고 TCP/IP의 소켓 프로그래밍은 Java API만을 사용하여 설명하였다.

리눅스 가상 머신

학생들이 Linux 시스템에 대해 더 잘 이해할 수 있도록, 우리는 이 교재와 함께 Ubuntu 배포판을 실행하는 Linux 가상 머신을 제공한다. 교재 웹 사이트(<http://www.os-book.com>)에서 다운로드 할 수 있는 가상 머신은 gcc 및 Java 컴파일러를 포함한 개발 환경도 제공한다. Windows API가 필요한 과제를 제외하고 이 가상 머신을 사용하여 이 책에 있는 대부분의 프로그래밍 과제를 완료할 수 있다. 가상 머신은 현재 Windows 10, Linux 및 macOS를 포함한 VirtualBox 가상화 소프트웨어를 실행할 수 있는 모든 호스트 운영체제에서 설치 및 실행될 수 있다.

10판

우리가 이 10판을 쓸 때 운영체제에 영향을 주는 핵심적인 4가지 영역의 지속적인 발전에 따라 개정하였다.

1. 모바일 운영체제
2. 다중 코어 시스템
3. 가상화
4. 비휘발성 메모리 보조저장장치

이러한 주제를 강조하기 위해 우리는 이 새 판 전반에 걸쳐 관련 내용을 통합했다. 예를 들어 Android 및 iOS 모바일 운영체제에 대한 논의 범위와 함께 모바일 기기를 지원하는 ARMv8 아키텍처에 대한 논의 범위를 크게 늘렸다. 또한 병행성과 병렬성을 지원하는 API 논의 범위를 늘리는 등 다중 코어 시스템 논의 범위를 늘렸다. SSD와 같은 비휘발성 메모리 장치는 이제 I/O, 대용량 저장장치 및 파일 시스템에 관해 설명하는 장에서 하드 디스크 드라이브와 동급으로 취급된다.

일부 독자들이 Java 논의 범위의 증가에 대해 지지를 표명했으며 이 판 전반에 걸쳐 추가의 Java 예제를 제공했다.

또한 우리는 거의 모든 장에서 오래된 자료를 최신 자료로 갱신하고 더는 흥미롭지

않거나 관련이 없는 자료를 삭제하여 자료를 다시 썼다. 우리는 많은 장을 재정렬 했으며 경우에 따라 절을 한 장에서 다른 장으로 이동했다. 또한 새로운 그림을 추가했을 뿐만 아니라 기존 그림을 수정하여 삽화를 크게 개선하였다.

주요 변화

10판의 개신은 내용과 새로운 지원 자료 측면에서 이전 개신보다 훨씬 더 많은 자료를 포함한다. 다음으로 각 장의 주요 내용 변경 사항에 대한 간략한 개요를 제공한다.

- **1장: 서론**에는 다중 코어 시스템의 개신된 논의 범위와 NUMA 시스템과 Hadoop 클러스터의 새로운 논의 범위가 포함된다. 오래된 자료가 업데이트되었으며 운영체제 공부를 해야 하는 새로운 동기가 추가되었다.
- **2장: 운영체제 구조**에서는 운영체제의 설계와 구현에 대해 크게 수정된 논의를 제공한다. Android 및 iOS에 대한 내용을 업데이트했으며 Linux 시스템용 GRUB를 중심으로 시스템 부트 프로세스 논의 범위를 수정했다. Linux-용 Windows 서브시스템에 대한 새로운 내용도 포함되어 있다. 링커와 로더에 대한 새로운 절을 추가했으며 이제 응용 프로그램이 운영체제에 따라 달라지는 이유에 관해 설명한다. 마지막으로 BCC 디버깅 툴셋에 대한 논의를 추가했다.
- **3장: 프로세스**는 이제 CPU 스케줄링 문제만 포함하게 하여 스케줄링 논의를 단순화한다. C 프로그램의 메모리 레이아웃, Android 프로세스 계층 구조, Mach 메시지 전달 및 Android RPC에 대해 새로운 설명을 추가하였다. 또한 전통적인 UNIX/Linux 초기화 프로세스의 논의를 `systemd`의 논의로 대체했다.
- **4장: 스레드와 병행성**(이전 스레드)은 API 및 라이브러리 수준에서 병행 및 병렬 프로그래밍에 대한 지원에 대한 논의를 늘렸다. Java 스레드에 대한 절을 개정하여 이제 미래를 포함하도록 하고 Apple의 Grand Central Dispatch에 대한 논의를 업데이트 하여 Swift를 포함하였다. 새로운 절에서는 Java의 포크-조인 프레임워크를 사용하는 포크-조인 병렬 처리와 Intel의 스레드 벌딩 블록에 대해 논의한다.
- **5장: CPU 스케줄**(이전 6장)은 다단계 큐 및 다중 코어 처리 스케줄링의 논의를 개정한다. 이 스케줄링이 부하 균등화에 미치는 영향을 포함하여 NUMA-인지 스케줄링 문제의 논의를 통합했다. 또한 Linux CFS 스케줄러의 관련 수정 사항에 대해서도 설명한다. 새로운 논의 범위는 라운드 로빈과 우선순위 스케줄링, 이기종 다중 처리 및 Windows 10 스케줄링에 대한 설명을 결합한다.
- **6장: 동기화 도구**(이전 5장 프로세스 동기화의 일부)는 프로세스 동기화를 위한 다양한 도구에 초점을 맞춘다. 중요한 새로운 논의는 명령 순서 변경 및 버퍼에 대한 지연된 쓰기와 같은 아키텍처 문제에 관해 설명한다. 이 장에서는 CAS (Compare-and-Swap) 명령어를 사용하는 잠금 없는 알고리즘도 소개한다. 특정 API가 제공

되지 않지만 대신 이 장에서는 경쟁 조건에 대한 소개와 데이터 경쟁을 방지하는 데 사용할 수 있는 일반적인 도구를 제공한다. 세부 사항에는 메모리 모델, 메모리 장벽 및 라이브러리에 관한 새로운 논의가 포함된다.

- **7장: 동기화 예제**(이전 5장 프로세스 동기화의 일부)에서는 전통적인 동기화 문제를 소개하고 이러한 문제를 해결하는 해결책 설계를 위한 특정 API 지원에 관해 설명 한다. 이 장에는 조건 변수뿐만 아니라 POSIX 명명 및 익명 세마포의 새로운 논의가 포함된다. Java 동기화에 대한 새로운 절도 추가되었다.
- **8장: 교착 상태**(이전 7장)는 라이브러리에 대한 새로운 절과 라이브러리 위험의 예로서 교착 상태에 관한 논의를 약간 수정한다. 이 장에는 Linux lockdep 및 BCC 교착 상태 탐지기 도구의 새로운 논의와 스레드 덤프를 사용한 Java 교착상태 감지의 논의가 포함되어 있다.
- **9장: 메인 메모리**(이전 8장)에는 최신 컴퓨터 시스템의 메모리 관리와 관련하여 이 장을 최신으로 업데이트하는 몇 가지 개정이 포함된다. ARMv8 64비트 아키텍처의 새로운 논의를 추가하고 동적 링크 라이브러리의 논의를 업데이트했으며 스와핑 논의는 이제 프로세스 스와핑이 아닌 페이지 스와핑 페이지에 초점을 맞추도록 변경하였다. 또한 세그먼트에 관한 논의를 삭제하였다.
- **10장: 가상 메모리**(이전 9장)에는 NUMA 시스템의 메모리 할당에 대한 업데이트된 논의와 가용 프레임 리스트를 사용한 전역 할당을 포함하여 여러 개정이 포함되어 있다. 새로운 논의에는 압축 메모리, 주요/사소한 페이지 폴트 및 Linux 및 Windows 10의 메모리 관리가 포함된다.
- **11장: 대용량 저장장치 구조**(이전의 10장)는 플래시 및 반도체 메모리 디스크와 같은 비휘발성 메모리 장치에 대한 논의를 추가한다. 하드 드라이브 스케줄링은 현재 사용되는 알고리즘만 표시하도록 단순화시켰다. 또한 클라우드 저장장치에 대한 새로운 절, 업데이트된 RAID 논의 및 객체 저장소에 대한 새로운 토론도 포함된다.
- **12장: I/O**(이전 13장)는 기술 및 성능 수치의 논의를 업데이트하고 동기/비동기 및 봉쇄/비봉쇄 I/O의 논의를 확장하며 벡터화 된 I/O에 대한 절을 추가한다. 또한 모바일 운영체제의 전원 관리 논의를 확장한다.
- **13장: 파일 시스템 인터페이스**(이전 11장)는 현재 기술에 대한 정보로 업데이트되었다. 특히, 디렉터리 구조의 논의가 향상되었으며 보호 논의가 업데이트되었다. 메모리 맵드 파일 절이 확장되었으며 공유 메모리에 대한 토론에 Windows API 예제가 추가되었다. 주제 순서는 13장과 14장의 주제의 순서가 재구성되었다.
- **14장: 파일 시스템 구현**(이전 12장)은 현재 기술을 다루면서 업데이트되었다. 이 장에는 이제 TRIM과 Apple File System에 대한 설명이 포함되어 있다. 또한 성능

에 대한 논의가 업데이트되었으며 저널링 논의가 확대되었다.

- **15장: 파일 시스템 내부**는 새로운 내용이며 이전 11장과 12장의 업데이트된 정보가 포함되어 있다.
- **16장: 보안**(이전 15장)이 이제 보호 장보다 먼저 나온다. 랜섬웨어 및 원격 액세스 도구를 포함하여 현재 보안 위협 및 솔루션에 대한 개정 및 업데이트된 용어가 포함된다. 최소 권한의 원칙이 강조된다. 코드 주입 취약성 및 공격에 대한 논의가 개정되었으며 이제 코드 예제가 포함된다. 암호화 기술에 대한 논의는 현재 사용되는 기술에 초점을 맞추기 위해 업데이트되었다. (암호 및 기타 방법에 의한) 인증의 논의는 유용한 힌트와 함께 업데이트되고 확장되었다. 또한 주소 공간 레이아웃 무작위화에 대한 논의와 보안 방어의 새로운 요약이 추가된다. Windows 7 예제가 Windows 10으로 업데이트되었다.
- **17장: 보호**(이전 14장)에는 주요 변경 사항이 포함된다. 보호 링과 레이어에 대한 설명이 업데이트되었으며 이제 Bell-LaPadula 모델을 참조하고 TrustZone의 ARM 모델과 Secure Monitor Call을 살펴본다. 강제적 액세스 제어의 논의와 함께 꼭 알아야 할 원칙의 논의가 확장된다. Linux 기능, Darwin 자격, 보안 무결성 보호, 시스템 콜 필터링, 샌드박싱 및 코드 서명에 대한 하위 절이 추가되었다. 스택 검사 기법을 포함하여 Java의 런타임 기반 강제시행 논의도 추가되었다.
- **18장: 가상 컴퓨터**(이전 16장)에는 하드웨어 지원 기술에 대한 추가 정보가 포함되어 있다. 컨테이너, 영역, 도커 및 Kubernetes를 포함하여 응용 프로그램 격리에 대한 주제도 확장되었다. 새로운 절에서는 unikernels, 라이브러리 운영체제, 하이퍼바이저 분할 및 분리 하이퍼바이저를 포함하여 진행 중인 가상화 연구에 대해 논의한다.
- **19장: 네트워크 및 분산 시스템**(이전 17장)은 상당히 업데이트되었으며 이제는 컴퓨터 네트워크와 분산 시스템의 논의를 통합한다. 이 자료는 최신 컴퓨터 네트워크와 분산 시스템과 관련하여 최신 정보를 제공하도록 개정되었다. TCP/IP 모델의 논의를 강화하였고, 클라우드 스토리지에 대한 논의가 추가되었다. 네트워크 토플로지 절은 삭제되었다. 이를 변환의 논의가 확장되고 Java 예제가 추가되었다. 이 장에는 Google 파일 시스템 위에 구현된 MapReduce, Hadoop, GPFS 및 Lustre를 비롯한 분산 파일 시스템에 대한 새로운 내용도 포함되어 있다.
- **20장: Linux 시스템**(이전 18장)이 Linux 4.i 커널을 다루도록 업데이트되었다.
- **21장: Windows 10 시스템**은 Windows 10의 내부를 다루는 새로운 장이다.
- **부록 A:** 본문에서 더는 다루지 않는 장의 내용을 포함하도록 영향력 있는 운영체제가 업데이트되었다.

지원 웹사이트

여러분이 이 교재를 지원하는 웹 사이트 www.wiley.com/college/silberschatz를 방문하면 다음과 같은 자원들을 다운로드 할 수 있다.

- Linux 가상 머신
- C와 Java 소스코드
- 모든 그림과 삽화
- FreeBSD와 Mach, Windows 7 사례 연구
- 오류 수정
- 참고문헌

강의자를 위한 정보

운영체제 개론 및 고급 운영체제론 과목에서 이 책을 활용하기 위한 다양한 접근 예를 제안하는 견본 강의 개요를 이 책을 위한 웹 사이트를 통해 제공한다. 일반적으로 우리는 독자들이 이 책을 순서대로 통독하기를 권장한다. 왜냐하면 이 방법이야말로 운영체제를 가장 완벽하게 공부할 방법이기 때문이다. 그러나 견본 강의 요강을 이용하여 각 장 또는 각 장의 절들을 임의의 순서대로 선택하여 공부할 수 있다.

이 판에서 많은 새로운 연습문제, 새로운 프로그래밍 문제와 프로젝트를 추가하였다. 새로운 프로그래밍 과제는 프로세스, 스레드, 프로세스 스케줄링, 프로세스 동기화와 메모리 관리와 관련되어 있다. 몇몇 과제는 Linux 시스템에 새로운 커널 모듈을 추가하는 작업을 필요로 하는데 이 과제를 수행하기 위해서는 이 교재와 함께 제공되는 Linux 가상 머신 또는 적절한 Linux 배포판을 이용해야 한다.

연습문제와 프로그래밍 과제에 대한 해답은 이 책을 이용하여 운영체제 수업을 진행하는 강의자에게만 제공된다. 이러한 제한된 보조 자료를 얻으려면 John Wiley & Sons의 지역 출판에 연락하면 된다. 지역 출판은 <http://www.wiley.com/college> 사이트에서 “Who's my rep?” 링크를 클릭하면 알 수 있다.

학생들을 위한 정보

여러분이 각 장의 마지막에 나오는 실습문제를 충분히 활용하기를 권장한다. 우리는 또한 공부 가이드를 정독하기를 바란다. 이 가이드는 수강생 중의 한 학생이 준비한 것이다. 마지막으로 UNIX와 Linux 시스템이 생소한 학생들은 지원 사이트에서 제공하고 있는 Linux Virtual Machine을 다운로드 하여 설치해보기를 권장한다. 이러한 작업은

여러분에게 새로운 컴퓨팅 경험을 제공할 뿐 아니라 Linux가 공개 소스 소프트웨어이기 때문에 이 대중적인 운영체제의 내부 상세 사항을 쉽게 탐구할 수 있게 한다. 운영체제를 공부하는 동안 행운이 짓들기를 기원한다.

저자와의 연락

우리는 이 새로운 판의 오자, 버그 등의 오류를 고치려고 시도하였다. 그러나 새로운 소프트웨어의 공표와 마찬가지로 대부분의 버그가 그대로 남아있을 것이다. 가장 최신의 오류 수정 목록은 책의 지원 사이트에서 얻을 수 있도록 하겠다. 현재 수정 목록에는 등록되지 않은 어떠한 오류 또는 빠진 오류들을 우리에게 알려준다면 매우 고맙게 생각할 것이다.

책을 개선할 수 있는 제안 또한 기쁘게 받아들일 것이다. 프로그래밍 연습문제, 프로젝트 제안, 온라인 실습과 지도 및 강의 조언과 같은 다른 독자들에 도움이 될 만한 웹사이트 개선 사항 또한 환영한다. 이 모든 사항을 os-book-authors@cs.yale.edu로 보내주기를 바란다.

감사의 말

많은 사람이 이 10판과 그것이 파생된 이전 9판에서 우리를 도왔다.

제 10 판

- Rik Farrow는 기술 편집자로서 전문가 조언을 제공했다.
- Jonathan Levin은 모바일 시스템, 보호 및 보안에 대한 지원을 제공했다.
- Alex Ionescu는 이전 Windows 7장을 업데이트하여 21장: Windows 10을 제공했다.
- Sarah Diesburg는 19장: 네트워크 및 분산 시스템을 수정했다.
- Brendan Gregg는 BCC 툴셋에 대한 지침을 제공했다.
- Richard Stallman (RMS)은 무료 및 공개 소스 소프트웨어의 설명에 대한 피드백을 제공했다.
- Robert Love는 20장: Linux 시스템에 대한 업데이트를 제공했다.
- Michael Shapiro는 저장장치 및 I/O 기술 세부 사항을 지원했다.
- Richard West는 가상화 연구 분야에 대한 통찰력을 제공했다.
- Clay Breshears는 Intel 스레드 빌딩 블록을 다루는 데 도움을 주었다.
- Gerry Howser는 운영체제에 대한 연구의 동기 부여에 대한 피드백을 제공했으며 새로운 자료를 그의 강의에서 시험했다.

- Judi Paige는 그림 생성과 슬라이드 표현에 도움을 주었다.
- Jay Gagne와 Audra Rissmeyer는 이 판의 새로운 삽화를 준비했다.
- Owen Galvin은 11장 및 12장에 대한 기술 편집을 제공했다.
- Mark Wogahn은 이 책을 제작하는 소프트웨어(LaTeX 및 글꼴)가 제대로 작동하는지 확인했다.
- Ranjan Kumar Meher는 이 새로운 텍스트 제작에 사용된 일부 LaTeX 소프트웨어를 다시 작성했다.

이전 판

- **처음 세 판.** 이 책은 이전 판에서 파생된 것이며 그중 처음 세 판은 James Peterson과 공동 저술한 것이다.
- **전반적인 기여.** 이전 버전에서 우리를 도와준 다른 사람들로는 Hamid Arabnia, Rida Bazzi, Randy Bentson, David Black, Joseph Boykin, Jeff Brumfield, Gael Buckley, Roy Campbell, PC Capon, John Carpenter, Gil Carrick, Thomas Casavant, Bart Childs, Ajoy Kumar Datta, Joe Deck, Sudarshan K. Dhall, Thomas Doeppner, Caleb Drake, M. Rasit Eskicioğlu, Hans Flack, Robert Fowler, G. Scott Graham, Richard Guy, MaxHailperin, Rebecca Hartman, Wayne Hathaway, Christopher Haynes, Don Heller, Bruce Hillyer, Mark Holliday, Dean Hougen, Michael Huang, Ahmed Kamel, Morty Kewstel, Richard Kieburtz, Carol Kroll, Morty Kwestel, Thomas LeBlanc, John Leggett, Jerrold Leichter, Ted Leung, Gary Lippman, Carolyn Miller, Michael Molloy, Euripides Montagne, Yoichi Muraoka, Jim M. Ng, Banu Özden, Ed Posnak, Boris Putanec, Charles Qualline, John Quarterman, Mike Reiter, Gustavo Rodriguez-Rivera, Carolyn JC Schauble, Thomas P. Skinner, Yannis Smaragdakis, Jesse St. Laurent, John Stankovic, Adam Stauffer, Steven Stepanek, John Sterling, Hal Stern, Louis Stevens, Pete Thomas, David Umbaugh, Steve Vinoski, Tommy Wagner, Larry L. Wear, John Werth, James M. Westall, J. S. Weston 및 Yang Xiang 등이다.

• 특별한 기여

- Robert Love는 본문 전체에 걸쳐 20장과 Linux 논의를 모두 업데이트했으며 많은 Android 관련 질문에 답하였다.
- 부록 B는 Dave Probert가 작성했으며 운영체제 개념의 8판 22장에서 파생되었다.

- Jonathan Katz는 16장에 기고했다. Richard West는 18장에 정보를 제공했다. Salahuddin Khan은 Windows 7 보안에 대한 새로운 논의를 제공하기 위해 16.7절을 업데이트했다.
- 19장의 일부는 Levy and Silberschatz [1990]의 논문에서 파생되었다.
- 20장은 Stephen Tweedie의 출판되지 않은 원고에서 파생되었다.
- Cliff Martin은 FreeBSD를 다루기 위해 UNIX 부록을 업데이트하는 데 도움을 주었다.
- 일부 연습문제 및 해답은 Arvind Krishnamurthy가 제공했다.
- Andrew DeNicola는 웹 사이트에서 제공되는 학생 학습 안내서를 준비했다. 그 슬라이드 중 일부는 Marilyn Turnamian이 준비했다.
- Mike Shapiro, Bryan Cantrill 및 Jim Mauro는 Solaris 관련해 많은 질문에 답했으며 Sun Microsystems의 Bryan Cantrill은 ZFS 논의를 도왔다. Josh Dees와 Rob Reynolds는 Microsoft의 .NET에 대한 논의에 기여했다.
- Owen Galvin은 18장 판 편집을 도와주었다.

책 제작

편집장은 Don Fowley였다. 선임 프로덕션 편집자는 Ken Santor였다. 프리랜서 개발 편집자는 Chris Nelson이었다. 보조 개발 편집자는 Ryann Dannelly였다. 표지 디자이너는 Tom Nery였다. 카피 에디터는 Beverly Peavler였다. 프리랜서 교정자는 Katrina Avery였다. 프리랜서 색인 작성자는 WordCo, Inc였다. Aptara LaTex 팀은 Neeraj Saxena와 Lav kush로 구성되었다.

개인 메모

Avi는 이 책을 개정하는 동안 Valerie의 사랑, 인내 및 지원에 대해 감사의 말을 전한다.

Peter는 그의 아내 Carla와 그의 자녀 Gwen, Owen, Maddie에게 감사의 말을 전한다.

Greg은 가족, 즉 그의 아내 Pat과 아들 Thomas와 Jay의 지속적인 지원에 감사의 말을 전한다.

Abraham Silberschatz, New Haven, CT

Peter Baer Galvin, Boston, MA

Greg Gagne, Salt Lake City, UT

역자 서문

이 책은 운영체제 분야의 대표적인 교재 중 하나인 Silberschatz, Galvin과 Gagne의 저서인 *Operating Systems Concepts* 10판을 번역한 것이다. 시대의 흐름에 맞추어 기본적으로 전자책 버전으로 출판된 책이며, 필요에 따라 인쇄본을 제공하는 방식으로 변경한 첫 번째 판이다. 역자들이 번역한 책은 이 책의 Asia 판을 번역하였다.

미국 내에서 판매되는 버전과 Asia 버전의 가장 큰 차이점은 실습 과제 및 프로젝트 과제가 빠져 있다는 것이다. 운영체제 학습에 있어 실제로 소스 코드를 분석하고 수정하여 기능을 변경하거나 추가하는 것이 매우 중요한 과정임을 생각하면 아쉬운 점이다. 독자들은 이 점을 감안하여 교재 관련 웹 사이트에서 제공하는 여러 추가 자료를 같이 공부할 것을 권한다. 교재 관련 웹 사이트에 관한 정보는 교재의 서문을 참고하기 바란다.

역자는 가능한 한 원저의 설명을 그대로 전달하려고 노력하였으나 일부 설명에 대해서는 이해를 쉽게 할 수 있도록 의역하였다. 용어는 될 수 있으면 영어 용어를 그대로 사용하거나 나란히 적으려고 노력하였다. 특히 여러 역자가 공동으로 번역한 이전 버전을 기초로 하였기 때문에 일부 용어에 대해서 우리말 용어가 약간 다를 수도 있을 것이다. 또한 실제로 사용되는 용어 위주로, 그리고 정확한 의미 전달이 어려운 경우 로마자 표기법으로 용어를 번역하려고 하였음을 양지하기 바란다. 앞으로 이 책의 내용을 개정할 때 많은 용어를 통일하고 더 적합한 용어를 사용할 수 있기를 기대한다.

역자들의 노력에도 불구하고 미처 발견하지 못한 오류들이 곳곳에 존재할 것이다. 오류를 발견한 독자들은 역자들이나 출판사에 알려주어 책의 품질을 높이는 데 함께 해주기 바란다.

마지막으로 이 책의 출판을 위해 노력해준 퍼스트북 출판사와 우일미디어의 경영진과 편집진에 깊이 감사드린다.

2020년 2월
박민규



자료 Contents

저자 서문 *iii*

역자 서문 *xvii*

Part 1 ▶ 개관 1

Chapter 1 시론 3

- | | |
|-------------------------|---------------------|
| 1.1 운영체제가 할 일 4 | 2.9 운영체제 빌딩과 부팅 100 |
| 1.2 컴퓨터 시스템의 구성 7 | 2.10 운영체제 디버깅 105 |
| 1.3 컴퓨터 시스템 구조 16 | 2.11 요약 110 |
| 1.4 운영체제의 작동 23 | 연습 문제 112 |
| 1.5 자원 관리 28 | 추가 자료 112 |
| 1.6 보안과 보호 35 | |
| 1.7 가상화 36 | |
| 1.8 분산 시스템 38 | |
| 1.9 커널 자료구조 39 | |
| 1.10 계산 환경 43 | |
| 1.11 무료 및 공개 소스 운영체제 50 | |
| 1.12 요약 56 | |
| 연습 문제 57 | |
| 추가 자료 58 | |

Chapter 2 운영체제 구조 61

- | | |
|------------------------------|------------------------------------|
| 2.1 운영체제 서비스 62 | 3.1 프로세스 개념 118 |
| 2.2 사용자와 운영체제 인터페이스 64 | 3.2 프로세스 스케줄링 123 |
| 2.3 시스템 콜 68 | 3.3 프로세스에 대한 연산 128 |
| 2.4 시스템 서비스 82 | 3.4 프로세스 간 통신 137 |
| 2.5 링커와 로더 84 | 3.5 공유 메모리 시스템에서의
프로세스 간 통신 139 |
| 2.6 응용 프로그램이 운영체제마다 다른 이유 86 | 3.6 메시지 전달 시스템에서의
프로세스 간 통신 141 |
| 2.7 운영체제 설계 및 구현 88 | 3.7 IPC 시스템의 사례 146 |
| 2.8 운영체제 구조 90 | 3.8 클라이언트 서버 환경에서 통신 161 |

Part 2 ▶ 프로세스 관리 115

Chapter 3 프로세스 117

- | | |
|------------------------------------|------------|
| 3.1 프로세스 개념 118 | 3.9 요약 169 |
| 3.2 프로세스 스케줄링 123 | 연습 문제 171 |
| 3.3 프로세스에 대한 연산 128 | 추가 자료 172 |
| 3.4 프로세스 간 통신 137 | |
| 3.5 공유 메모리 시스템에서의
프로세스 간 통신 139 | |
| 3.6 메시지 전달 시스템에서의
프로세스 간 통신 141 | |
| 3.7 IPC 시스템의 사례 146 | |
| 3.8 클라이언트 서버 환경에서 통신 161 | |

Chapter 4 스레드와 병행성 175

- 4.1 개요 176
- 4.2 다중 코어 프로그래밍 178
- 4.3 다중 스레드 모델 182
- 4.4 스레드 라이브러리 185
- 4.5 암묵적 스레딩 194
- 4.6 스레드와 관련된 문제들 206
- 4.7 운영체제 사례 213
- 4.8 요약 215
- 연습 문제 216
- 추가 자료 217

Chapter 5 CPU 스케줄링 219

- 5.1 기본 개념 220
- 5.2 스케줄링 기준 225
- 5.3 스케줄링 알고리즘 226
- 5.4 스레드 스케줄링 239
- 5.5 다중 처리기 스케줄링 242
- 5.6 실시간 CPU 스케줄링 250
- 5.7 운영체제 사례들 259
- 5.8 알고리즘의 평가 268
- 5.9 요약 274
- 연습 문제 276
- 추가 자료 278

*Part 3 ▶ 프로세스 동기화 281**Chapter 6 동기화 도구들 283*

- 6.1 배경 283
- 6.2 임계구역 문제 286
- 6.3 Peterson의 해결안 288
- 6.4 동기화를 위한 하드웨어 지원 291
- 6.5 Mutex Locks 297
- 6.6 세마포 299
- 6.7 모니터 303
- 6.8 라이브러리 310
- 6.9 평가 312
- 6.10 요약 315
- 연습 문제 316
- 추가 자료 316

Chapter 7 동기화 예제 319

- 7.1 고전적인 동기화 문제들 319
- 7.2 커널 안에서의 동기화 327
- 7.3 POSIX 동기화 330
- 7.4 Java에서의 동기화 334
- 7.5 대체 방안들 342
- 7.6 요약 345
- 연습 문제 346
- 추가 자료 346

Chapter 8 교착 상태 349

- 8.1 시스템 모델 350
- 8.2 다중 스레드 응용에서의 교착 상태 351
- 8.3 교착 상태 특성 353
- 8.4 교착 상태 처리 방법 358
- 8.5 교착 상태 예방 360
- 8.6 교착 상태 회피 363
- 8.7 교착 상태 탐지 371
- 8.8 교착 상태로부터 회복 376
- 8.9 요약 378
- 연습 문제 378
- 추가 자료 380

*Part 4 ▶ 메모리 관리 383**Chapter 9 메모리 예오리 385*

- 9.1 배경 385
- 9.2 연속 메모리 할당 393
- 9.3 페이지 397
- 9.4 페이지 테이블의 구조 409
- 9.5 스와핑 414
- 9.6 사례: Intel 32비트와 64비트 구조 417
- 9.7 사례: ARM 구조 421
- 9.8 요약 423
- 연습 문제 424
- 추가 자료 426

Chapter 10 가상 메모리 427

- 10.1 배경 428
- 10.2 요구 페이징 431

- 10.3 쓰기 시 복사 438
- 10.4 페이지 교체 440
- 10.5 프레임의 할당 454
- 10.6 스파싱 461
- 10.7 메모리 압축 467
- 10.8 커널 메모리의 할당 469
- 10.9 기타 고려 사항 473
- 10.10 운영체제의 예 480
- 10.11 요약 483
 - 연습 문제 484
 - 추가 자료 487

Part 5 ▶ 저장장치 관리 489

Chapter 11 대용량 저장장치 구조 491

- 11.1 대용량 저장장치 구조의 개관 491
- 11.2 디스크 스케줄링 500
- 11.3 NVM 스케줄링 504
- 11.4 오류 감지 및 수정 505
- 11.5 저장장치 관리 506
- 11.6 스왑 공간 관리 511
- 11.7 저장장치 연결 513
- 11.8 RAID 구조 517
- 11.9 요약 530
 - 연습 문제 532
 - 추가 자료 533

Chapter 12 입출력 시스템 535

- 12.1 개관 535
- 12.2 입출력 하드웨어 536
- 12.3 응용 입출력 인터페이스 548
- 12.4 커널 입출력 서브시스템 556
- 12.5 입출력 요구를 하드웨어 연산으로 변환 566
- 12.6 STREAMS 568
- 12.7 성능 570
- 12.8 요약 574
 - 연습 문제 575
 - 추가 자료 575

Part 6 ▶ 파일 시스템 577

Chapter 13 파일 시스템 인터페이스 579

- 13.1 파일 개념 579
- 13.2 접근 방법 590
- 13.3 디렉터리 구조 593
- 13.4 보호 603
- 13.5 메모리 사상 파일 608
- 13.6 요약 613
 - 연습 문제 614
 - 추가 자료 615

Chapter 14 파일 시스템 구조 617

- 14.1 파일 시스템 구조 618
- 14.2 파일 시스템 구현 620
- 14.3 디렉터리 구현 623
- 14.4 할당 방법 625
- 14.5 가용 공간의 관리 634
- 14.6 효율과 성능 637
- 14.7 복구 642
- 14.8 예: WAFL 파일 시스템 646
- 14.9 요약 650
 - 연습 문제 651
 - 추가 자료 652

Chapter 15 파일 시스템 내부구조 655

- 15.1 파일 시스템 655
- 15.2 파일 시스템 마운팅 657
- 15.3 파티션과 마운팅 659
- 15.4 파일 공유 660
- 15.5 가상 파일 시스템 662
- 15.6 원격 파일 시스템 664
- 15.7 일관성의 의미 668
- 15.8 NFS 669
- 15.9 요약 676
 - 연습 문제 677
 - 추가 자료 677

Part 7 ▶ 보안과 보호 679**Chapter 16 보안 681**

- 16.1** 보안 문제 681
- 16.2** 프로그램 위협 685
- 16.3** 시스템과 네트워크 위협 696
- 16.4** 보안 도구로서 암호 기법 699
- 16.5** 사용자 인증 712
- 16.6** 보안 방어의 구현 717
- 16.7** 예: Windows 10 727
- 16.8** 요약 730
- 추가 자료 731

Chapter 17 보호 733

- 17.1** 보호의 목표 733
- 17.2** 보호의 원칙 734
- 17.3** 보호 링 735
- 17.4** 보호의 영역 738
- 17.5** 접근 행렬 742
- 17.6** 접근 행렬의 구현 746
- 17.7** 접근 권한의 취소 749
- 17.8** 역할 기반 액세스 제어 751
- 17.9** 강제적 접근 제어 752
- 17.10** 자격-기반 시스템 753
- 17.11** 기타 보호 개선 방법 755
- 17.12** 언어 기반의 보호 758
- 17.13** 요약 764
- 추가 자료 765

Part 8 ▶ 진보된 주제 767**Chapter 18 가상 머신 769**

- 18.1** 개요 769
- 18.2** 역사 771
- 18.3** 장점 및 특징 772
- 18.4** 빌딩 블록 775
- 18.5** VM 유형 및 구현 781
- 18.6** 가상화와 운영체제 구성요소 789
- 18.7** 사례 796
- 18.8** 가상화 연구 798
- 18.9** 요약 799
- 추가 자료 801

Chapter 19 네트워크 및**분산 시스템 803**

- 19.1** 분산 시스템의 장점 803
- 19.2** 네트워크 구조 805
- 19.3** 통신 구조 808
- 19.4** 네트워크 및 분산 운영체제 820
- 19.5** 분산 시스템의 설계 문제 824
- 19.6** 분산 파일 시스템 829
- 19.7** DFS 명명 및 투명성 833
- 19.8** 원격 파일 액세스 836
- 19.9** 분산 파일 시스템에 대한 최종 생각 840
- 19.10** 요약 841
- 연습 문제 842
- 추가 자료 843

Part 9 ▶ 사례 검토 845**Chapter 20 Linux 시스템 847**

- 20.1** Linux 역사 847
- 20.2** 설계 원칙 853
- 20.3** 커널 모듈 856
- 20.4** 프로세스 관리 860
- 20.5** 스케줄링 864
- 20.6** 메모리 관리 870
- 20.7** 파일 시스템 880
- 20.8** 입/출력 887
- 20.9** 프로세스 간 통신 890
- 20.10** 네트워크 구조 891
- 20.11** 보안 894
- 20.12** 요약 897
- 연습문제 898
- 추가 자료 898

Chapter 21 윈도우 10 901

- 21.1** 역사 901
- 21.2** 설계 원칙 906
- 21.3** 시스템 구성요소 920
- 21.4** 터미널 서비스와 빠른 사용자 교체 960
- 21.5** 파일 시스템 961
- 21.6** 네트워킹 968
- 21.7** 프로그래머 인터페이스 973

21.8 요약 986

연습 문제 987

추가 자료 987

Part 10 ▶ 부록 989

*Appendix A 영향력 있는
운영체제 991*

A.1 기능 전이 991

A.2 초기 시스템 992

A.3 Atlas 1000

A.4 XDS-940 1001

A.5 THE 1002

A.6 RC 4000 1002

A.7 CTSS 1004

A.8 MULTICS 1004

A.9 IBM OS/360 1005

A.10 TOPS-20 1006

A.11 CP/M과 MS/DOS 1007

A.12 Macintosh 운영체제와 Windows 1008

A.13 Mach 1008

A.14 자격-기반 시스템—Hydra 및 CAP 1010

A.15 기타 시스템들 1013

추가 자료 1013

찾아보기 1017

Appendix B, C, D와 Additional excercises는 퍼스트북 홈페이지 → 자료실 → 데이터파일에 올려 놓았으며, 아래 사이트에 접속하시면 확인하실 수 있습니다.

<http://firstbook.kr/>

Part 1

개관 Overview

운영체제는 컴퓨터 사용자와 컴퓨터 하드웨어 사이에서 중개자 역할을 한다. 운영체제의 목적은 사용자가 프로그램을 편리하고 효율적으로 수행할 수 있는 환경을 제공하는 데 있다.

운영체제는 컴퓨터 하드웨어를 관리하는 소프트웨어이다. 하드웨어는 컴퓨터 시스템의 정확한 동작을 보장하고 프로그램이 시스템의 정상적인 동작을 방해하지 않도록 하는 적합한 메커니즘을 제공해야 한다.

운영체제는 다양한 발전 방향을 따라 구성되었기 때문에 운영체제의 내부 구조는 매우 다양하다. 새 운영체제의 설계는 매우 어려운 작업이고, 시스템의 목표는 설계가 시작되기 전에 명확하게 정의되어야 한다는 것이 중요하다.

운영체제는 매우 덩치가 크고 복잡하므로 부분별로 생성되어야 한다. 이 하나의 부분은 전체 시스템 윤곽에 잘 맞는 일부여야 하며 이 부분들의 입력과 출력, 동작은 주의를 기울여 정의해야 한다.

서론

Introduction



운영체제(operating system)는 컴퓨터 하드웨어를 관리하는 소프트웨어이다. 운영체제는 또한 응용 프로그램을 위한 기반을 제공하며 컴퓨터 사용자와 컴퓨터 하드웨어 사이에서 중재자 역할을 수행한다. 운영체제의 놀라운 점은 광범위한 컴퓨팅 환경에서 이러한 일들을 매우 다양한 방법으로 수행한다는 것이다. 운영체제는 “사물 인터넷(Internet of Things)”을 포함하는 자동차와 홈 기기에서 스마트폰, 개인용 컴퓨터, 대형 컴퓨터 및 클라우드 컴퓨팅 환경까지 어느 곳에나 존재한다.

현대 컴퓨팅 환경에서 운영체제의 역할을 탐구하기 위하여 먼저 컴퓨터 하드웨어의 구성과 구조를 이해하는 것이 중요하다. 이러한 지식에는 CPU, 메모리 및 입출력 장치와 저장장치가 포함된다. 운영체제의 기본적인 책임은 이러한 자원들을 프로그램에 할당하는 것이다.

운영체제는 덩치가 매우 크고 복잡하므로 부분별로 생성되어야 한다. 이 하나의 부분은 전체 시스템 윤곽에 잘 맞는 일부여야 하며 이 부분들의 입력과 출력, 동작은 주의를 기울여 정의해야 한다. 이 장에서는 현대 컴퓨터 시스템의 주요 구성요소와 운영체제가 제공하는 기능에 대한 일반적인 개관을 제공한다. 또한 이 책의 나머지 부분을 위한 무대를 설정하는 데 필요한 다수의 주제, 운영체제에 사용되는 자료구조, 계산환경 및 공개 소스 및 무료 운영체제를 다룬다.

이 장의 목표

- 컴퓨터 시스템의 일반적인 구성과 인터럽트의 역할을 기술한다.
- 현대 다중 처리기 컴퓨터 시스템의 구성요소에 관해 기술한다.
- 사용자 모드에서 커널 모드로의 전환에 대해 설명한다.
- 다양한 컴퓨팅 환경에서 운영체제가 어떻게 사용되는지 논의한다.
- 무료 및 공개 소스 운영체제의 예를 제공한다.

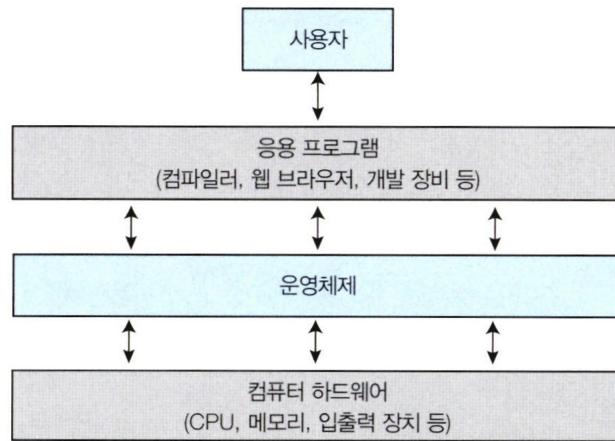


그림 1.1 컴퓨터 시스템 구성요소에 대한 개략적 구성도

1.1 운영체제가 할 일 *_What Operating Systems Do*

전체 컴퓨터 시스템에서 운영체제가 수행하는 역할에 대해 살펴보는 것으로 논의를 시작한다. 컴퓨터 시스템은 대개 네 가지 구성요소인 하드웨어, 운영체제, 응용 프로그램 및 사용자로 구분할 수 있다(그림 1.1).

하드웨어는 중앙 처리 장치(CPU), 메모리 및 입출력(I/O) 장치로 구성되어, 기본 계산용 자원을 제공한다. **응용 프로그램**인 워드 프로세서, 스프레드시트, 컴파일러, 그리고 웹 브라우저 등은 사용자의 계산 문제를 해결하기 위해 이들 자원이 어떻게 사용될지를 정의한다. 운영체제는 다양한 사용자를 위해 다양한 응용 프로그램 간의 하드웨어 사용을 제어하고 조정한다.

또한 우리는 컴퓨터 시스템이 하드웨어, 소프트웨어 및 데이터로 구성되어 있다고 볼 수 있다. 운영체제는 컴퓨터 시스템이 동작할 때 이들 자원을 적절하게 사용할 수 있는 방법을 제공한다. 운영체제는 정부(government)와 유사하다. 운영체제는 정부처럼 그 자체로는 유용한 기능을 수행하지 못한다. 운영체제는 단순히 다른 프로그램이 유용한 작업을 할 수 있는 환경을 제공한다. 운영체제의 역할을 좀 더 완전히 이해하기 위하여 사용자와 시스템 두 관점에서 살펴본다.

1.1.1 사용자 관점 *_User View*

컴퓨터에 대한 사용자의 관점은 사용되는 인터페이스에 따라 달라진다. 많은 컴퓨터 사용자는 랙톱 혹은 모니터, 키보드, 마우스로 구성된 PC 앞에서 작업한다. 이러한 시스템은 한 사용자가 자원을 독점하도록 설계되었으며 목표는 사용자가 수행하는 작업(또는 놀이)을 최대화하는 것이다. 이러한 경우 운영체제는 대부분 **사용의 용이성**을 위해 설계되고 성능에 약간 신경을 쓰고 다양한 하드웨어와 소프트웨어 자원이 어떻게 공유되느냐의 **자원의 이용**에는 전혀 신경을 쓰지 않는다.

점점 더 많은 사용자가 스마트폰 및 태블릿과 같은 모바일 장치(일부 사용자의 데스크톱 및 랩톱 컴퓨터 시스템을 대체하는 장치)와 상호 작용한다. 이러한 장치는 일반적으로 셀룰러 또는 기타 무선 기술을 통해 네트워크에 연결된다. 모바일 컴퓨터용 사용자 인터페이스는 일반적으로 사용자는 실제 키보드와 마우스를 사용하지 않고 화면에서 손가락을 누르고 스와이프하여 시스템과 상호 작용하는 **터치스크린**이 특징이다. 또한 많은 휴대 기기에서 사용자가 Apple의 **Siri**와 같은 **음성 인식** 인터페이스를 통해 상호 작용할 수 있다.

일부 컴퓨터는 사용자 관점이 존재하지 않거나 매우 작은 예도 있다. 예를 들면, 가전제품이나 자동차 내의 **내장형 컴퓨터**는 숫자 키패드를 가지고, 상태를 보이기 위해 표시등을 켜고 끌 수 있지만 이들 컴퓨터나 운영체제와 응용 프로그램은 사용자의 개입 없이 작동하도록 설계되어 있다.

1.1.2 시스템 관점 *System View*

컴퓨터의 관점에서 운영체제는 하드웨어와 가장 밀접하게 연관된 프로그램이다. 따라서 우리는 운영체제를 **자원 할당자**(resource allocator)로 볼 수 있다. 컴퓨터 시스템은 문제를 해결하기 위해 요구되는 여러 가지 자원들(하드웨어와 소프트웨어), 즉 CPU 시간, 메모리 공간, 저장장치 공간, 입출력 장치 등을 가진다. 운영체제는 이를 자원의 관리자로서 동작한다. 자원에 대해 서로 상충할 수도 있는 많은 요청이 있으므로, 운영체제는 컴퓨터 시스템을 효율적이고 공정하게 운영할 수 있도록 어느 요청에 자원을 할당할지를 결정해야 한다.

운영체제에 대한 다소 다른 관점은 여러 가지 입출력 장치와 사용자 프로그램을 제어할 필요성을 강조한다. 운영체제는 **제어 프로그램**(control program)이다. 제어 프로그램은 컴퓨터의 부적절한 사용을 방지하기 위해 사용자 프로그램의 수행을 제어한다. 운영체제는 특히 입출력 장치의 제어와 작동에 깊이 관여한다.

1.1.3 운영체제의 정의

이제 독자들은 **운영체제**가 많은 역할과 기능을 수행한다는 것을 알았을 것이다. 이는 부분적으로는 컴퓨터의 설계와 용도가 수없이 많기 때문이다. 컴퓨터는 토스터, 자동차, 선박, 우주여행선, 가정 및 산업체에 존재한다. 컴퓨터는 게임기, 케이블 TV 수신기 및 산업체어 시스템의 기반이 된다.

이러한 다양성을 설명하기 위하여 컴퓨터의 역사를 살펴볼 수 있다. 컴퓨터의 역사는 짧지만 급격히 발전해 왔다. 컴퓨팅은 처음 무엇을 할 수 있을지 알기 위한 실험으로 시작했으나 곧바로 암호 깨기와 탄도 계산과 같은 군사용 및 인구조사 계산과 같은 정부 업무등의 고정 목적 시스템으로 전용되었다. 이를 초기의 컴퓨터들이 범용의 다기능 대형컴퓨터로 발전하였으며 그즈음 운영체제가 탄생하였다. 1960년대에 **무어의 법칙**(Moore's Law)이 집적회로의 트랜지스터 수가 18개월마다 배가할 것이라고 예측했으

며 이 법칙은 지켜져 왔다. 컴퓨터는 기능이 확대되고 크기가 작아졌으며 용도가 다양해졌고 다양한 운영체제가 등장하였다. (운영체제의 자세한 역사는 부록 A를 보라.)

그렇다면 운영체제가 무엇인지 어떻게 정의할 수 있는가? 일반적으로 운영체제에 대한 적합한 정의는 없다. 운영체제는 유용한 컴퓨팅 시스템을 만드는 문제를 해결할 수 있는 합리적인 방법을 제공하기 때문에 존재한다. 컴퓨터 시스템의 기본 목표는 프로그램을 실행하고 사용자 문제를 더욱 쉽게 해결할 수 있게 하는 것이다. 컴퓨터 하드웨어는 이 목표를 가지고 구성된다. 오로지 하드웨어만으로는 사용하기가 쉽지 않으므로 응용 프로그램이 개발된다. 이러한 프로그램에는 입출력 장치 제어와 같은 특정 공통 작업이 필요하다. 자원을 제어하고 할당하는 일반적인 기능은 운영체제라는 하나의 소프트웨어로 통합된다.

또한 운영체제에 포함되는 요소에 보편적인 정의는 없다. 단순한 관점은 “운영체제”를 주문할 때 공급 업체가 제공하는 모든 것을 포함한다는 것이다. 그러나 포함된 기능은 시스템마다 크게 다르다. 일부 시스템은 메가바이트 미만의 공간을 차지하고 전체 화면 편집기가 없는 반면, 다른 시스템은 기가바이트의 공간이 필요하며 그래픽 윈도 시스템을 기반으로 한다. 더욱 일반적인 정의와 우리가 지지하는 것은 운영체제가 컴퓨터에서 항상 실행되는 프로그램(일반적으로 **커널**이라고 함)이다. 커널과 함께 두 가지 다른 유형의 프로그램이 있다. 운영체제와 관련되어 있지만 반드시 커널의 일부일 필요는 없는 **시스템 프로그램**과 시스템 작동과 관련되지 않은 모든 프로그램을 포함하는 응용 프로그램이다.

개인용 컴퓨터가 널리 보급되고 운영체제가 점점 정교해짐에 따라 운영체제의 구성 요소가 무엇인지가 점점 더 중요해졌다. 1998년 미국 법무부는 Microsoft가 운영체제에 너무 많은 기능을 포함하여 응용 프로그램 공급 업체의 경쟁을 막았다고 주장하면서 Microsoft에 대해 소송을 제기했다. (예를 들어, 웹 브라우저는 Microsoft 운영체제의 필수 요소이다.) 결과적으로 Microsoft는 운영체제의 독점적 위치를 사용하여 경쟁을 제한하였다고 유죄를 선고받았다.

그러나 오늘날 모바일 기기의 운영체제를 살펴보면 운영체제를 구성하는 기능의 수가 다시 증가하고 있음을 알 수 있다. 모바일 운영체제에는 종종 핵심 커널뿐만 아니라 **미들웨어**(응용 프로그램 개발자에게 추가 서비스를 제공하는 일련의 소프트웨어 프레임워크)도 포함된다. 예를 들어, Apple의 iOS 및 Google의 Android와 같이 가장 유명한 두 가지 모바일 운영체제 각각에는 데이터베이스, 멀티미디어 및 그래픽을 지원하는 미들웨어와 함께 핵심 커널이 포함되어 있다(몇 가지만 언급하면).

요약하자면, 운영체제에는 항상 실행 중인 커널, 응용 프로그램 개발을 쉽게 하고 기능을 제공하는 미들웨어 프레임워크 및 시스템 실행 중에 시스템을 관리하는 데 도움이 되는 시스템 프로그램이 포함된다. 본 교재의 대부분은 범용 운영체제의 커널과 관련 있지만 운영체제 설계 및 연산을 모두 설명하는 데 필요한 다른 구성요소에 대해서도 설명한다.

왜 운영체제를 공부하는가?

컴퓨터 과학에 종사하는 사람은 많지만 운영체제를 만들거나 수정하는 데는 소수만이 참여한다. 그렇다면 왜 운영체제와 그들의 작동방식을 공부하는가? 단순하게 거의 모든 코드가 운영체제 위에서 실행되므로 운영체제 작동방식에 대한 지식은 적절하고 효율적이며 효과적이며 안전한 프로그래밍에 중요하기 때문이다. 운영체제의 기본 지식, 컴퓨터 하드웨어 구동 방식 및 응용 프로그램에 제공하는 내용을 이해하는 것은 운영체제를 작성하는 사람들에게 필수적일 뿐만 아니라 그 위에서 프로그램을 작성하고 운영체제를 사용하는 사람들에게도 매우 유용하다.

1.2 컴퓨터 시스템의 구성 *Computer-System Organization*

현대의 범용 컴퓨터 시스템은 하나 이상의 CPU와 구성요소와 공유 메모리 사이의 액세스를 제공하는 공통 **버스**를 통해 연결된 여러 장치 컨트롤러로 구성된다(그림 1.2). 각 장치 컨트롤러는 특정 유형의 장치(예: 디스크 드라이브, 오디오 장치 또는 그래픽 디스플레이)를 담당한다. 컨트롤러에 따라 둘 이상의 장치가 연결될 수도 있다. 예를 들어 하나의 시스템 USB 포트는 여러 장치를 연결할 수 있는 USB 허브에 연결할 수 있다. 장치 컨트롤러는 일부 로컬 버퍼 저장소와 특수 목적 레지스터 집합을 유지 관리한다. 장치 컨트롤러는 제어하는 주변 장치와 로컬 버퍼 저장소 간에 데이터를 이동한다.

일반적으로 운영체제에는 각 장치 컨트롤러마다 **장치 드라이버**가 있다. 이 장치 드라이버는 장치 컨트롤러의 작동을 잘 알고 있고 나머지 운영체제에 장치에 대한 일관된 인터페이스를 제공한다. CPU와 장치 컨트롤러는 병렬로 실행되어 메모리 사이클을 놓고 경쟁한다. 공유 메모리를 질서 있게 액세스하기 위해 메모리 컨트롤러는 메모리에 대한 액세스를 동기화한다.

다음 절에서는 시스템의 세 가지 주요 측면에 중점을 두어 이러한 시스템의 작동방식의 기본 사항에 대해 설명한다. CPU의 조치가 필요한 이벤트에 대해 경고하는 인터럽트부터 살펴볼 것이다. 그런 다음 저장장치 구조 및 입출력 구조에 대해 설명한다.

1.2.1 인터럽트 *Interrupts*

일반적인 컴퓨터 작업(입출력을 수행하는 프로그램)을 고려하자. 입출력 작업을 시작하기 위해 장치 드라이버는 장치 컨트롤러의 적절한 레지스터에 값을 적재한다. 그런 다음 장치 컨트롤러는 이러한 레지스터의 내용을 검사하여 수행할 작업을(예: “키보드에서 문자 읽기”) 결정한다. 컨트롤러는 장치에서 로컬 버퍼로 데이터 전송을 시작한다. 데이터 전송이 완료되면 장치 컨트롤러는 장치 드라이버에게 작업이 완료되었음을 알린다. 그런 다음 장치 드라이버는 읽기 요청이면 데이터 또는 데이터에 대한 포인터를 반환하며 운

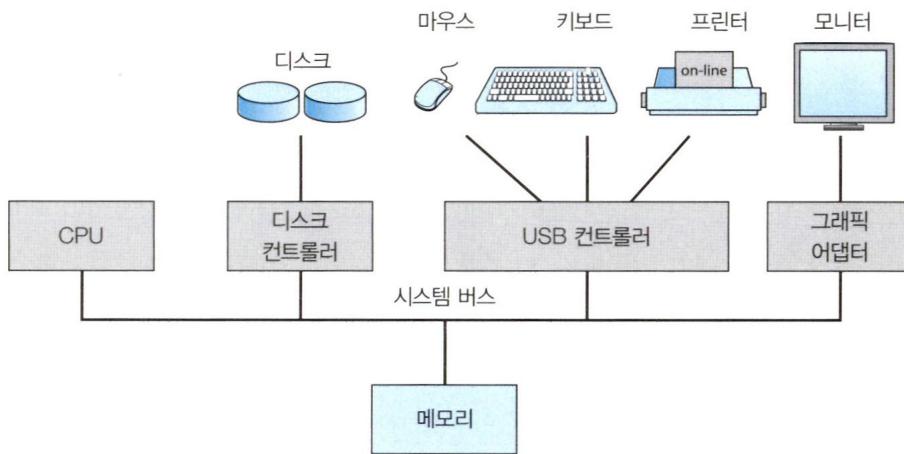


그림 1.2 통상적인 PC 컴퓨터 시스템

영체제의 다른 부분에 제어를 넘긴다. 다른 작업의 경우 장치 드라이버는 “쓰기 완료” 또는 “장치 사용 중”과 같은 상태 정보를 반환한다. 그러나 컨트롤러는 장치 드라이버에게 작업을 완료했다는 사실을 어떻게 알리는가? 이는 **인터럽트**를 통해 이루어진다.

1.2.1.1 개요 *Overview*

하드웨어는 어느 순간이든 시스템 버스를 통해 CPU에 신호를 보내 인터럽트를 발생시킬 수 있다. (컴퓨터 시스템에는 많은 버스가 있을 수 있지만 시스템 버스는 주요 구성요소 사이의 주요 통신 경로이다.) 인터럽트는 다른 많은 목적으로도 사용되며 운영체제와 하드웨어의 상호 작용 방식의 핵심 부분이다.

CPU가 인터럽트 되면, CPU는 하던 일을 중단하고 즉시 고정된 위치로 실행을 옮긴다. 이러한 고정된 위치는 일반적으로 인터럽트를 위한 서비스 루틴이 위치한 시작 주소를 가지고 있다. 그리고 인터럽트 서비스 루틴이 실행된다. 인터럽트 서비스 루틴의 실행이 완료되면, CPU는 인터럽트 되었던 연산을 재개한다. 이러한 연산의 시간 일정 (time line)이 그림 1.3에 있다. 이 그림과 관련된 애니메이션을 실행하려면 여기를 클릭하라.

인터럽트는 컴퓨터 구조의 중요한 부분이다. 각 컴퓨터 설계는 자신의 인터럽트 메커니즘을 가지고 있으며, 몇 가지 기능은 공통적이다. 인터럽트는 적절한 서비스 루틴으로 제어를 전달한다. 이러한 전달을 관리하는 직선적인 방법은 인터럽트 정보를 조사하는 일반적인 루틴을 호출하는 방법이다. 이 루틴은 이어 인터럽트 고유의 핸들러(handler)를 호출한다. 그러나, 인터럽트는 매우 빈번하게 발생하기 때문에 빠르게 처리되어야 한다. 필요한 속도를 제공하기 위해 인터럽트 루틴에 대한 포인터들의 테이블을 대신 이용할 수 있다. 이 경우 중간 루틴을 둘 필요 없이, 테이블을 통하여 간접적으로 인터럽트 루틴이 호출될 수 있다. 일반적으로 포인터들의 테이블은 하위 메모리에 저장된다(첫

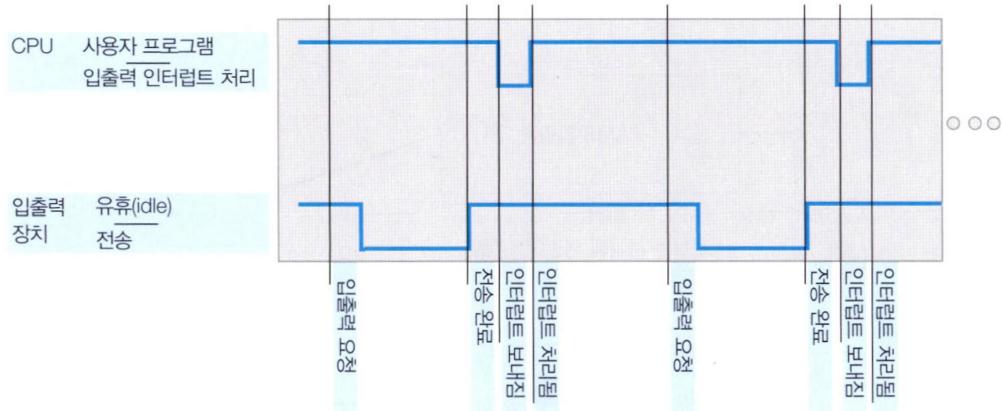


그림 1.3 출력을 수행하고 있는 단일 프로세스에 대한 인터럽트 시간 일정

100개 정도의 위치). 이들 위치에는 여러 장치에 대한 인터럽트 서비스 루틴의 주소가 들어 있다. 인터럽트가 요청되면, 인터럽트를 유발한 장치를 위한 인터럽트 서비스 루틴의 주소를 제공하기 위해 이 주소의 배열, 즉 **인터럽트 벡터**가 인터럽트 요청과 함께 주어진 고유의 유일한 장치 번호로 색인된다. Windows나 UNIX 같은 서로 다른 운영체제가 이러한 방법으로 인터럽트를 처리한다.

인터럽트 구조는 또한 인터럽트된 모든 정보를 저장해야 인터럽트를 처리한 후 이 정보를 복원할 수 있다. 만약 인터럽트 루틴이 처리기의 상태를 변경할 필요(예를 들어, 레지스터의 값을 변경하여)가 있다면, 인터럽트 루틴은 반드시 명시적으로 현재의 상태를 저장하여야 하며, 복귀하기 전에 상태를 복원해야 한다. 인터럽트를 서비스한 후, 저장되어 있던 복귀 주소를 프로그램 카운터에 적재하고, 인터럽트에 의해 중단되었던 연산이 인터럽트가 발생되지 않았던 것처럼 다시 시작된다.

1.2.1.2 구현 *Implementation*

기본 인터럽트 메커니즘은 다음과 같이 작동한다. CPU 하드웨어에는 **인터럽트 요청 라인**(interrupt request line)이라는 선이 있는데, 이는 하나의 명령어의 실행을 완료할 때마다 CPU가 이 선을 감지한다. CPU가 컨트롤러가 인터럽트 요청 라인에 신호를 보낸 것을 감지하면, 인터럽트 번호를 읽고 이 번호를 인터럽트 벡터의 인덱스로 사용하여 **인터럽트 핸들러 루틴**(interrupt-handler routine)으로 점프한다. 그런 다음 해당 인덱스와 관련된 주소에서 실행을 시작한다. 인터럽트 처리기는 작업 중에 변경될 상태를 저장하고, 인터럽트 원인을 확인하고, 필요한 처리를 수행하고, 상태 복원을 수행하고, `return_from_interrupt` 명령어를 실행하여 CPU를 인터럽트 전 실행 상태로 되돌린다. 장치 컨트롤러가 인터럽트 요청 라인에 신호를 선언하여 인터럽트를 발생(raise)시키고 CPU는 인터럽트를 포착(catch)하여 인터럽트 핸들러로 디스패치(dispatch)하고 핸들러는 장치를 서비스하여 인터럽트를 지운다(clear). 그림 1.4는 인터럽트-구동

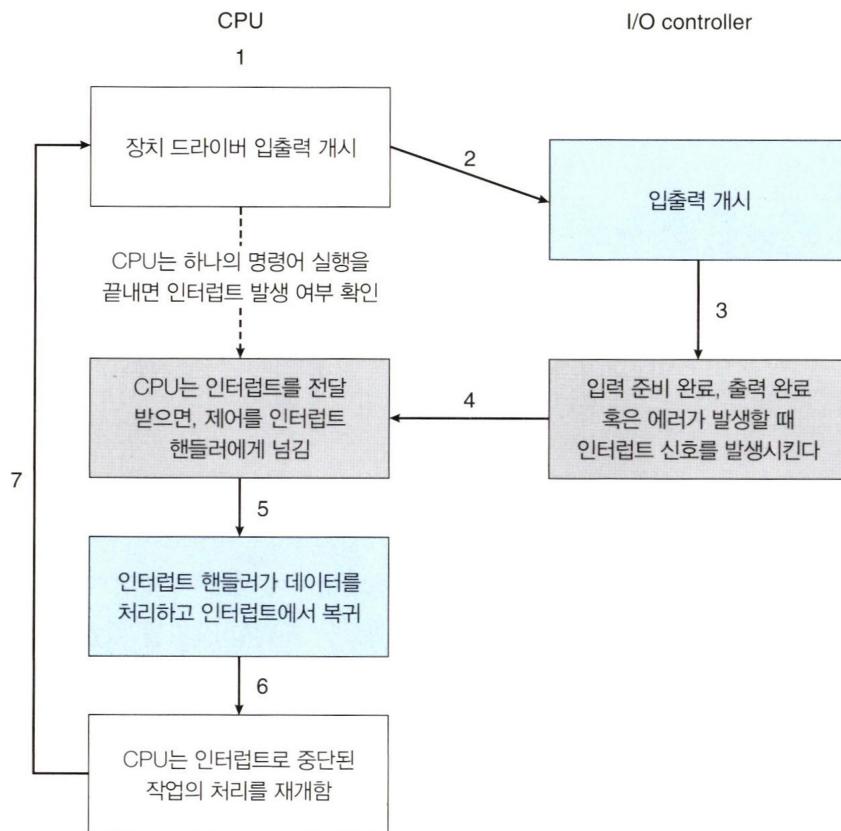


그림 1.4 인터럽트-구동 방식의 입출력 사이클

입출력 사이클을 요약한 것이다.

방금 설명한 기본 인터럽트 기법은 장치 컨트롤러가 서비스할 준비가 될 때와 같은 비동기 이벤트에 CPU가 대응할 수 있게 한다. 그러나 최신 운영체제에서는 더욱 정교한 인터럽트 처리 기능이 필요하다.

1. 중요한 처리 중에 인터럽트 처리를 연기할 수 있어야 한다.
2. 장치의 적절한 인터럽트 핸들러로 효율적으로 디스패치 할 방법이 필요하다.
3. 운영체제가 우선순위가 높은 인터럽트와 우선순위가 낮은 인터럽트를 구분하고 적절한 긴급도로 대응할 수 있도록 다단계 인터럽트가 필요하다.

최신 컴퓨터 하드웨어에서 이 세 가지 기능은 CPU 및 **인터럽트 컨트롤러 하드웨어**에 의해 제공된다.

대부분의 CPU에는 2개의 인터럽트 요청 라인이 있다. 하나는 복구할 수 없는 메모리 오류와 같은 이벤트를 위해 예약된 **마스크 불가능 인터럽트**(nonmaskable interrupt)이다. 두 번째 인터럽트 라인은 **마스킹 가능**(maskable)이다. 인터럽트 되어서는 안되는 중요한 명령 시퀀스를 실행하기 전에 CPU에 의해 꺼질 수 있다. 마스킹 가능한

인터럽트는 장치 컨트롤러가 서비스를 요청하기 위해 사용된다.

벡터 방식 인터럽트 기법의 목적은 서비스가 필요한 장치를 결정하기 위해 하나의 인터럽트 핸들러가 가능한 모든 인터럽트 소스를 검색할 필요를 줄이기 위한 것이라는 것을 상기하자. 그러나 실제로 컴퓨터에는 인터럽트 벡터의 주소 개수보다 많은 장치(따라서 인터럽트 처리기)가 있다. 이 문제를 해결하는 일반적인 방법은 **인터럽트 체인**을 사용하는 것이다. 인터럽트 벡터의 각 원소는 인터럽트 핸들러 리스트의 헤드를 가리킨다. 인터럽트가 발생하면 요청을 처리할 수 있는 핸들러가 발견될 때까지 상응하는 리스트의 핸들러가 하나씩 호출된다. 이러한 구조는 큰 크기의 인터럽트 테이블의 오버헤드와 하나의 인터럽트 핸들러로 디스패치 하는 비효율성의 절충안이다.

그림 1.5는 Intel 프로세서의 인터럽트 벡터 설계를 보여준다. 마스크 불가능한 0에서 31까지의 이벤트는 다양한 오류 조건을 알리는 데 사용된다. 마스크 가능한 32에서 255까지의 이벤트는 장치가 생성한 인터럽트 같은 그 외 인터럽트를 처리하기 위해 사용된다.

인터럽트 기법은 또한 인터럽트 **우선순위 레벨**(interrupt priority level)을 구현한다. 이러한 레벨을 통해 CPU는 모든 인터럽트를 마스킹하지 않고도 우선순위가 낮은

벡터 번호	설명
0	나눗셈 에러
1	디버그 예외
2	Null 인터럽트
3	중단점(breakpoint)
4	INTO-검출 오버플로
5	제한 범위 예외
6	유효하지 않은 opcode
7	장치 사용하지 않음
8	더블 폴트
9	부처리기 세그먼트 침범(예약됨)
10	유효하지 않은 태스크 상태 세그먼트
11	존재하지 않는 세그먼트
12	스택 폴트
13	일반 보호
14	페이지 폴트
15	(Intel에서 예약함, 사용하지 말 것)
16	부동소수점 에러
17	정렬 검사
18	기기 검사
19~31	(Intel에서 예약함, 사용하지 말 것)
32~255	Maskable 인터럽트

그림 1.5 Intel 처리기 이벤트-벡터 테이블

인터럽트 처리를 연기할 수 있고, 우선순위가 높은 인터럽트가 우선순위가 낮은 인터럽트의 실행을 선점할 수 있다.

요약하면, 인터럽트는 최신 운영체에서 비동기 이벤트를 처리하기 위해 사용된다(다른 목적으로 사용되는 것에 관해서는 교재 전체에서 논의할 것임). 장치 컨트롤러 및 하드웨어 오류로 인해 인터럽트가 발생한다. 가장 긴급한 작업을 먼저 수행하기 위해 최신 컴퓨터는 인터럽트 우선순위 시스템을 사용한다. 인터럽트는 시간에 민감한 처리에 빈번하게 사용되므로 시스템 성능을 좋게 하려면 효율적인 인터럽트 처리가 필요하다.

1.2.2 저장장치 구조 *Storage Structure*

CPU는 메모리에서만 명령을 적재할 수 있으므로 실행하려면 프로그램을 먼저 메모리에 적재해야 한다. 범용 컴퓨터는 프로그램 대부분을 메인 메모리(**random-access memory**, 또는 **RAM**이라 불린다)라 불리는 재기록 가능한 메모리에서 가져온다. 메인 메모리는 **dynamic random-access memory (DRAM)**라 불리는 반도체 기술로 구현된다.

컴퓨터는 다른 형태의 메모리도 사용한다. 예를 들어, 컴퓨터 전원을 켰 때 가장 먼저 실행되는 프로그램은 **부트스트랩 프로그램**이며 운영체제를 적재한다. RAM은 **휘발성**(전원이 깨지거나 손실될 때 내용이 손실됨)이므로 부트스트랩 프로그램을 유지하는 용도로 사용할 수 없다. 대신 이 목적과 다른 목적으로, 컴퓨터는 전기적으로 소거 가능한 프로그램 가능 읽기 전용 메모리(**EEPROM**) 및 기타 형태의 **펌웨어**(쓰기 작업이 자주 발생하지 않고 비휘발성이 저장장치)를 사용한다. EEPROM은 변경할 수는 있지만 자주 변경할 수는 없다. 또한 속도가 느리므로 주로 사용되지 않는 정적 프로그램과 데이터가 포함되어 있다. 예를 들어, iPhone은 EEPROM을 사용하여 장치의 일련 번호 및 하드웨어 정보를 저장한다.

모든 형태의 메모리는 바이트의 배열을 제공한다. 각 바이트는 자신의 주소를 가지고 있다. 상호 작용은 특정 메모리 주소들에 대한 일련의 적재(load), 또는 저장(store) 명령을 통하여 이루어진다. 적재 명령은 메인 메모리로부터 CPU 내부의 레지스터로 한 바이트 또는 한 워드를 옮기는 것이다. 반대로 저장 명령은 레지스터의 내용을 메인 메모리로 옮긴다. 명시적인 적재, 저장 명령 외에, CPU는 프로그램 카운터에 저장된 위치로부터 실행하기 위해 메인 메모리에서 명령을 자동으로 적재한다.

폰 노이만 구조 시스템에서 실행되는 전형적인 명령 – 실행 사이클은 먼저 메모리로부터 명령을 인출해, 그 명령을 **명령 레지스터**(instruction register)에 저장한다. 이어서 명령을 해독하고, 이는 메모리로부터 피연산자를 인출하여 내부 레지스터에 저장하도록 유발할 수 있다. 피연산자에 대해 명령을 실행한 후에 결과가 메모리에 다시 저장될 수 있다. 메모리 장치는 단지 일련의 메모리 주소만을 인식한다는 사실에 유의하라. 메모리는 이를 주소[**명령 카운터**(instruction counter), **색인**(indexing), **간접 주소**(indirection), **리터럴 주소**(literal addresses) 등]가 어떻게 생성되었는지 알지 못하며, 그

저장장치 정의와 표기

컴퓨터 저장장치의 기본단위는 **비트**이다. 한 비트는 0과 1 두 값 중 하나를 가진다. 컴퓨터의 모든 다른 저장장치는 비트의 집합을 기반으로 한다. 비트의 길이가 충분히 긴 경우 컴퓨터는 놀랍게도 다양한 것들을 표현할 수 있다. 몇 개 예를 든다면 숫자, 글자, 영상, 영화, 소리, 문서 그리고 프로그램 등이다. 한 **바이트**는 8비트이고 대부분의 컴퓨터에서는 가장 작은 편리한 저장 단위이다. 예를 들면 대부분의 컴퓨터들은 하나의 비트를 이동하는 명령어는 제공하지 않지만 한 바이트를 이동하는 명령어는 제공한다. 약간 덜 알려진 용어가 **워드**이다. 워드는 그 컴퓨터 구조의 본연의 데이터 단위이다. 한 워드는 하나 이상의 바이트로 구성된다. 예를 들면 64비트 레지스터들과 64비트 메모리 주소지정을 가지는 컴퓨터는 전형적으로 64비트(8바이트) 워드를 가진다. 컴퓨터는 많은 연산을 한 번에 한 바이트 단위가 아니라 본연의 워드 단위로 실행한다.

컴퓨터 저장장치는 대부분의 컴퓨터 처리량과 마찬가지로 바이트 단위 및 바이트의 집합 단위로 측정되며 조작된다. **1킬로바이트** 또는 **KB**는 1024바이트이고, **1메가바이트** 또는 **MB**는 $1,024^2$ 바이트, **1기가바이트** 또는 **GB**는 $1,024^3$ 바이트, **1테라바이트** 또는 **TB**는 $1,024^4$ 바이트, **1페타바이트** 또는 **PB**는 $1,024^5$ 바이트이다. 컴퓨터 제조업체는 이들 숫자를 반올림하여 1메가바이트를 백만 바이트, 1기가바이트를 십억 바이트라 말한다. 네트워킹 수치는 이러한 일반 규칙에 대한 예외로 비트 단위로 표현된다(왜냐하면 네트워크는 데이터를 한 번에 1비트씩 이동하기 때문이다).

것이 무엇인지(명령인지 데이터인지) 알지 못한다. 그러므로 우리는 메모리 주소가 프로그램에 의해 어떻게 생성되었는지 무시할 수 있다. 우리는 단지 실행 중인 프로그램에 의해 생성된 일련의 메모리 주소에만 흥미가 있다.

이상적으로는, 프로그램과 데이터가 메인 메모리에 영구히 존재하기를 원한다. 그러나 이는 대부분의 시스템에서 두 가지 이유로 불가능하다.

1. 메인 메모리는 모든 필요한 프로그램과 데이터를 영구히 저장하기에는 너무 작다.
2. 메인 메모리는 이미 언급한 것처럼 전원이 공급되지 않으면 그 내용을 잊어버리는 휘발성 저장장치이다.

그러므로 대부분의 컴퓨터 시스템은 메인 메모리의 확장으로 **보조저장장치**를 제공한다. 보조저장장치의 주요 요건은 대량의 데이터를 영구히 보존할 수 있어야 한다는 점이다.

가장 일반적인 보조저장장치는 **하드 디스크 드라이브(HDD)**와 **비휘발성 메모리**

(**NVM**) 장치로, 프로그램과 데이터 모두를 위한 저장소를 제공한다. 대부분의 프로그램(시스템 및 응용 프로그램)은 메모리에 적재될 때까지 보조저장장치에 저장된다. 그런 후 많은 프로그램이 보조저장장치를 처리 소스 및 대상으로 모두 사용한다. 보조저장장치도 메인 메모리보다 훨씬 느리다. 따라서 11장에서 논의할 것처럼, 2차 저장장치의 올바른 관리는 컴퓨터 시스템에서 가장 중요하다.

그러나 더 큰 의미에서 레지스터, 메인 메모리 및 보조저장장치로 구성된 저장장치 구조는 가능한 많은 저장장치 시스템 설계 중 하나일 뿐이다. 다른 가능한 구성요소로는 캐시 메모리, CD-ROM 또는 Blu-ray, 자기 테이프 등이 있다. 다른 장치에 저장된 자료의 백업 사본을 저장하기 위해 특수 목적으로만 사용하기에 매우 느리고 충분히 큰 장치를 **3차 저장장치**라고 한다. 각 저장장치 시스템은 데이터를 저장하고 나중에 검색될 때까지 해당 데이터를 유지하는 기본 기능을 제공한다. 다양한 저장장치 시스템 간의 주요 차이점은 속도, 크기 및 휘발성에 있다.

다양한 저장장치 시스템은 저장 용량 및 액세스 시간에 따라 계층 구조로 구성될 수 있다(그림 1.6). 일반적으로 크기와 속도 사이에는 상충하는 측면이 있어서 메모리가 작고 빠를수록 CPU에 더 가깝다. 그림에서 볼 수 있듯이 속도와 용량의 차이 외에도 다양한 저장장치 시스템은 휘발성 또는 비휘발성이다. 앞에서 언급했듯이 휘발성 저장장치는 장치의 전원이 제거될 때 내용을 잃어버리므로 안전하게 보관하기 위해 데이터를 비휘발성 저장장치에 기록해야 한다.

그림에서 최상위 4단계 메모리는 반도체 기반 전자회로로 구성된 **반도체 메모리**를 사용하여 구성된다. 네 번째 수준의 NVM 장치에는 여러 가지 변형이 있지만 일반적으로 하드 디스크보다 빠르다. NVM 장치의 가장 일반적인 형태는 스마트폰 및 태블릿과

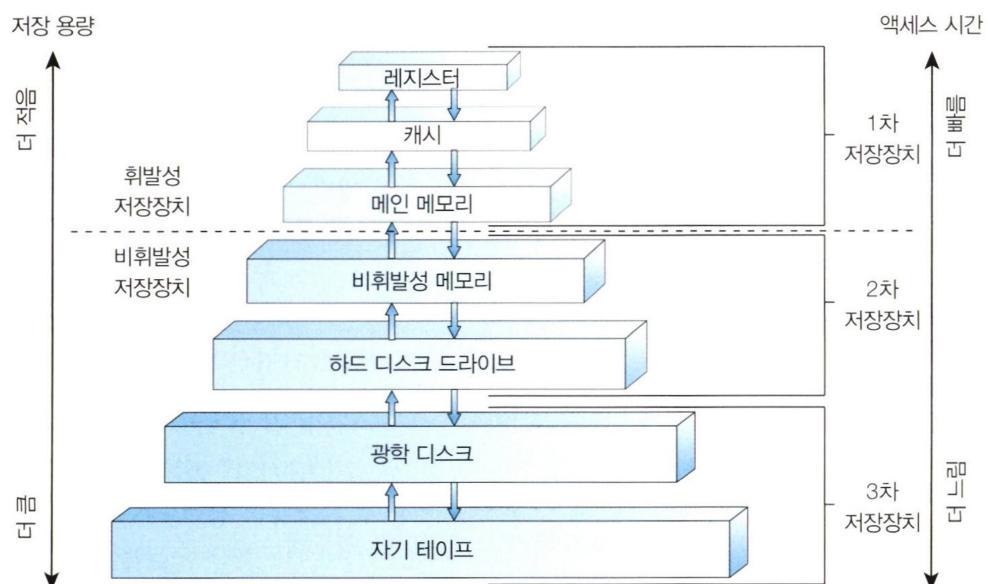


그림 1.6 저장장치 계층 구조

같은 모바일 장치에서 널리 사용되는 플래시 메모리이다. 점차 랩톱, 데스크톱과 서버에서의 장기 저장에도 플래시 메모리가 사용되고 있다.

저장장치는 운영체제 구조에서 중요한 역할을 하기 때문에 교재에서 자주 참조할 것이다. 일반적으로 다음 용어를 사용할 것이다.

- 휘발성 저장장치는 단순히 **메모리**라고 할 것이다. 특정 유형의 저장장치(예: 레지스터)를 강조해야 하는 경우 명시적으로 표현할 것이다.
- 비휘발성 저장장치는 전원이 꺼졌을 때 내용을 유지한다. 이를 **NVS**라고 한다. 대부분의 경우 NVS는 보조저장장치를 가리킨다. 이 유형의 저장장치는 다음 두 가지 유형으로 분류될 수 있다.
 - **기계적**. 이러한 저장장치 시스템의 몇 가지 예는 HDD, 광 디스크, 홀로그램 저장장치 및 자기 테이프이다. 특정 유형의 기계식 저장장치(예: 자기 테이프)를 강조해야 하는 경우 명시적으로 그 용어를 사용할 것이다.
 - **전기적**. 이러한 저장장치 시스템의 몇 가지 예는 플래시 메모리, FRAM, NRAM 및 SSD이다. 전기적 저장장치를 **NVM**으로 언급될 것이다. 특정 유형의 전기적 저장장치(예: SSD)를 강조해야 하는 경우 명시적으로 그 용어를 사용할 것이다.

기계적 저장장치는 일반적으로 전기적 저장장치보다 용량이 크고 바이트당 비용이 저렴하다. 반대로, 전기적 저장장치는 일반적으로 기계적 저장장치보다 비싸고 용량이 적으며 빠르다.

완전한 저장장치 시스템의 설계는 방금 논의한 모든 요소의 균형을 맞추어야 한다. 가능한 한 많은 저렴한 비휘발성 저장장치를 제공하는 동시에 필요한 만큼만 비싼 메모리를 사용해야 한다. 캐시는 두 구성요소 간에 액세스 시간이나 전송 속도의 차이가 큰 경우 성능을 향상하기 위해 설치할 수 있다.

1.2.3 입출력 구조 *I/O Structure*

운영체제 코드의 상당 부분은 시스템의 안정성과 성능에 대한 중요성과 장치의 다양한 특성으로 인해 I/O 관리에 할애된다.

이 절의 시작 부분에서 시스템은 범용 버스를 통해 데이터를 교환하는 여러 장치로 구성된 범용 컴퓨터라는 점을 상기하라. 1.2.1절에 설명된 인터럽트 구동 I/O의 형태는 소량의 데이터를 이동하는 데는 좋지만 NVS I/O와 같은 대량 데이터 이동에 사용될 때 높은 오버헤드를 유발할 수 있다. 이 문제를 해결하기 위해 **직접 메모리 액세스 (DMA)**가 사용된다. 장치에 대한 버퍼 및 포인터, 입출력 카운트를 세팅한 후 장치 제어기는 CPU의 개입 없이 메모리로부터 자신의 버퍼 장치로 또는 버퍼로부터 메모리로 데이터 블록 전체를 전송한다. 속도가 느린 장치처럼 한 바이트마다 인터럽트가 발생하

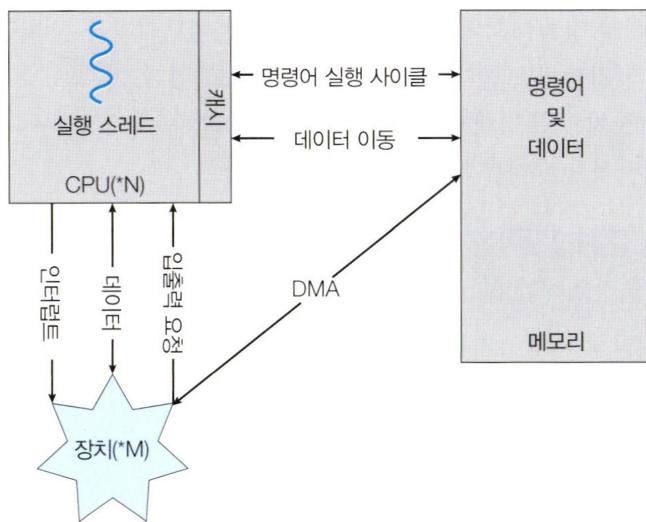


그림 1.7 현대 컴퓨터의 작동방식

는 것이 아니라 블록 전송이 완료될 때마다 인터럽트가 발생한다. 장치 컨트롤러가 전송 작업을 수행하고 있는 동안 CPU는 다른 작업을 수행할 수 있다.

몇몇 고가의 시스템은 버스 대신에 스위치 구조를 사용한다. 이러한 시스템에서는 공유 버스를 사용하기 위한 사이클을 경쟁하지 않고 다수의 구성요소가 다른 구성요소들과 동시에 통신하는 것이 가능하다. 이 경우 DMA의 사용은 더욱 효과적이다. 그림 1.7은 컴퓨터 시스템의 구성요소 간의 상호 작용을 보여준다.

1.3 컴퓨터 시스템 구조 *Computer-System Architecture*

1.2절에서 전형적인 컴퓨터 시스템의 일반적인 구조를 소개하였다. 컴퓨터 시스템은 사용된 범용 처리기의 수에 따라 분류 가능한 다양한 방식으로 구성될 수 있다.

1.3.1 단일 처리기 시스템 *Single-Processor Systems*

몇 년 전, 대부분의 컴퓨터 시스템은 단일 처리 코어를 가진 하나의 CPU를 포함하는 단일 프로세서를 사용했다. **코어**는 명령을 실행하고 로컬로 데이터를 저장하기 위한 레지스터를 포함하는 구성요소이다. 코어를 가진 하나의 메인 CPU는 프로세스의 명령어를 포함하여 범용 명령어 세트를 실행할 수 있다. 이 시스템에는 다른 특수 목적 프로세서도 있다. 디스크, 키보드 및 그래픽 컨트롤러와 같은 장치별 프로세서 형태로 제공될 수도 있다.

이 모든 전용 처리기들은 제한된 명령어 집합을 실행하고 사용자 프로세스를 실행하지는 않는다. 때로 이 처리기들은 운영체제에 의해 관리되기도 하는데, 운영체제는 이 처리기들이 수행할 다음 태스크에 대한 정보를 보내고 처리기들의 상태를 감시한다. 예

를 들면, 디스크 컨트롤러 마이크로프로세서는 주 CPU로부터 연속된 요청을 받아들여 자기 고유의 디스크 큐와 스케줄링 알고리즘을 구현한다. 이 배합은 CPU가 직접 디스크 스케줄링을 해야 하는 오버헤드를 감소시킨다. PC의 키보드는 키스트로크를 CPU에 전송할 코드로 변환하는 마이크로프로세서를 가지고 있다. 다른 시스템 또는 환경에서는 전용 처리기가 하드웨어로 구현되는 저수준의 구성요소이다. 운영체제는 이 처리기들과 통신할 수 있으며 이 처리기들은 독립적으로 자신의 작업을 처리한다. 전용 마이크로프로세서의 사용은 일반적인 형태이며 그렇다고 단일 처리기 시스템을 다중 처리기 시스템으로 변환하지는 않는다. 단일 처리 코어를 가진 범용 CPU가 하나만 있는 경우 시스템은 단일 프로세서 시스템이다. 그러나 이 정의에 따르면, 현대 컴퓨터 시스템은 단일 프로세서 시스템이 거의 없다.

1.3.2 다중 처리기 시스템 *Multiprocessor Systems*

모바일 장치에서 서버에 이르기까지 최신 컴퓨터에서는 **다중 처리기 시스템**이 컴퓨팅 환경을 지배하고 있다. 일반적으로 이러한 시스템에는 각각 단일 코어 CPU가 있는 두 개 이상의 프로세서가 있다. 프로세서는 컴퓨터 버스 및 때때로 클록, 메모리 및 주변 장치를 공유한다. 다중 처리기 시스템의 주요 장점은 처리량 증가이다. 즉, 프로세서 수를 늘리면 더 적은 시간에 더 많은 작업을 수행할 수 있다. 그러나 N 프로세서의 속도 향상 비율은 N 이 아니다. 여러 프로세서가 하나의 작업에 협력할 때 모든 프로세서가 올바르게 작동하게 유지하는 데 일정한 양의 오버헤드가 발생한다. 이 오버헤드와 공유 자원에 대한 경합은 추가 프로세서의 예상 이득을 낮춘다.

가장 일반적인 다중 처리기 시스템은 각 피어 CPU 프로세서가 운영체제 기능 및 사용자 프로세스를 포함한 모든 작업을 수행하는 **SMP (symmetric multiprocessing)**를 사용한다. 그림 1.8은 각각 자체 CPU를 가지는 두 개의 프로세서가 있는 일반적인

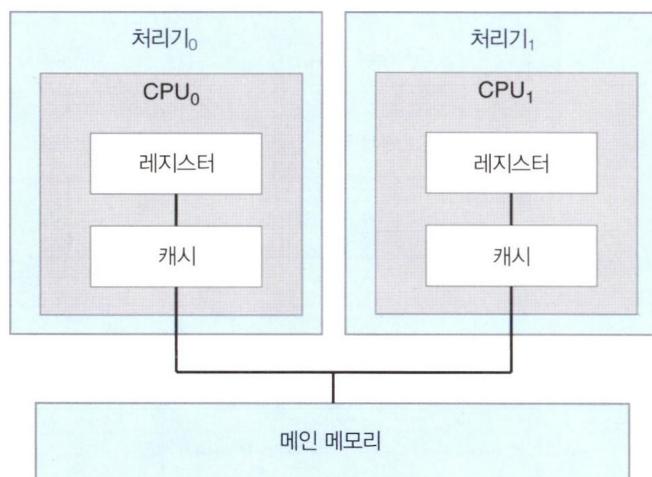


그림 1.8 대칭형 다중 처리 구조

SMP 구조를 보여준다. 각 CPU 처리기에는 개별 또는 로컬 캐시뿐만 아니라 자체 레지스터 세트가 있다. 그러나 모든 프로세서는 시스템 버스를 통해 물리 메모리를 공유한다.

이 모델의 장점은 많은 프로세스를 동시에 실행할 수 있다는 것이다. N 개의 CPU가 있으면 성능을 크게 저하하지 않으면서 N 개의 프로세스를 실행할 수 있다. 그러나 CPU가 독립적이기 때문에 하나는 유휴 상태이고 다른 하나는 과부하가 걸려 비효율적일 수 있다. 프로세서가 특정 자료구조를 공유하는 경우 이러한 비효율성을 피할 수 있다. 이 형식의 다중 처리기 시스템을 사용하면 프로세스 및 메모리와 같은 자원을 다양한 프로세서 간에 동적으로 공유할 수 있으며 프로세서 간의 작업 부하 분산을 낮출 수 있다. 이러한 시스템은 5장과 6장에서 볼 수 있듯이 신중하게 작성해야 한다.

다중 처리기의 정의는 시간이 지남에 따라 발전해 왔으며 이제는 여러 개의 컴퓨팅 코어가 단일 칩에 상주하는 **다중 코어** 시스템을 포함한다. 칩 내 통신이 칩 간 통신보다 빠르므로 다중 코어 시스템은 단일 코어를 가지는 여러 칩보다 효율적일 수 있다. 또한 여러 개의 코어를 가지는 하나의 칩은 여러 개의 단일 코어 칩보다 훨씬 적은 전력을 사용하는데, 이는 노트북뿐만 아니라 모바일 장치의 중요한 문제이다.

그림 1.9에서는 같은 프로세서 칩에 두 개의 코어를 가지는 이중 코어 설계를 보여준다. 이 설계에서 각 코어에는 자체 레지스터 세트와 레벨 1 (L1) 캐시라고도 하는 자체 로컬 캐시가 있다. 또한 레벨 2 (L2) 캐시는 칩에 국한되지만 두 처리 코어에서 공유한다. 아키텍처 대부분은 로컬 및 공유 캐시를 결합한 이 접근 방식을 채택한다. 로컬 하위 레벨 캐시는 일반적으로 상위 레벨 공유 캐시보다 작고 빠르다. 캐시, 메모리 및 버스 경험과 같은 아키텍처 고려 사항 외에도 N 코어를 가지는 다중 코어 프로세서는 운영체제에 N 개의 CPU처럼 보인다. 이러한 특성은 운영체제 설계자 및 응용 프로그램 프로그

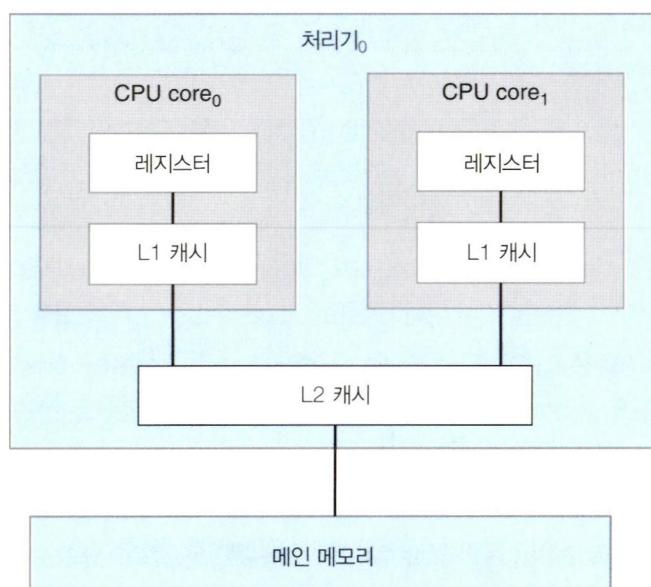


그림 1.9 하나의 칩에 두 개의 코어를 가지는 이중-코어 설계

컴퓨터 시스템 구성요소의 정의

- **CPU** — 명령을 실행하는 하드웨어.
- **프로세서(processor)** — 하나 이상의 CPU를 포함하는 물리적 칩.
- **코어(core)** — CPU의 기본 계산 단위.
- **다중 코어(multicore)** — 동일한 CPU에 여러 컴퓨팅 코어를 포함함.
- **다중 처리기(multiprocessor)** — 여러 프로세서를 포함함.

사실상 거의 모든 시스템은 이제 다중 코어이지만 컴퓨터 시스템의 단일 계산 단위를 가리킬 때는 일반적인 용어인 **CPU**를 사용하고 하나의 CPU에 존재하는 하나 이상의 코어를 구체적으로 언급할 때 **코어**와 **다중 코어** 용어를 사용한다.

래머에게 이러한 처리 코어를 효율적으로 사용하는 코드를 개발하도록 압박한다. 이러한 쟁점은 4장에서 다룰 것이다. Windows, macOS 및 Linux를 포함한 거의 모든 최신 운영체제는 물론 Android 및 iOS 모바일 시스템도 다중 코어 SMP 시스템을 지원한다.

다중 처리기 시스템에 CPU를 추가하면 컴퓨팅 성능이 향상된다. 그러나 앞에서 제안한 것처럼 이러한 개념은 그다지 확장성이 좋지 않고, CPU를 너무 많이 추가하면 시스템 버스에 대한 경합이 병목 현상이 되어 성능이 저하되기 시작한다. 다른 방법은 각 CPU(또는 CPU 그룹)에 작고 빠른 로컬 버스를 통해 액세스 되는 자체 로컬 메모리를 제공하는 것이다. 모든 CPU가 **공유 시스템 연결**로 연결되어 모든 CPU가 하나의 물리 주소 공간을 공유한다. **NUMA (non-uniform memory access)**라고 하는 이 방법은

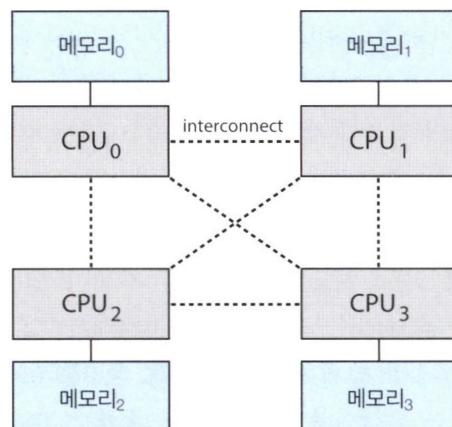


그림 1.10 NUMA 다중 처리 구조

그림 1.10에 도시되어 있다. 장점은 CPU가 로컬 메모리에 액세스 할 때 빠를 뿐만 아니라 시스템 상호 연결에 대한 경합도 없다는 것이다. 따라서 NUMA 시스템은 더 많은 프로세서가 추가될수록 더 효과적으로 확장할 수 있다.

NUMA 시스템의 잠재적 단점은 CPU가 시스템 상호 연결을 통해 원격 메모리에 액세스해야 할 때 지연 시간이 증가하여 성능 저하가 발생할 수 있다는 것이다. 즉, 예를 들어, CPU₀은 자체 로컬 메모리에 액세스 할 수 있는 만큼 빠르게 CPU₃의 로컬 메모리에 액세스 할 수 없어 성능이 저하된다. 5.5.2절과 10.5.4절에서 논의된 것처럼 운영체제는 신중한 CPU 스케줄링 및 메모리 관리를 통해 이 NUMA의 단점을 최소화할 수 있다. NUMA 시스템은 많은 수의 프로세서를 수용할 수 있도록 확장할 수 있으므로 고성능 컴퓨팅 시스템뿐만 아니라 서버에서도 점점 인기를 얻고 있다.

마지막으로 **블레이드 서버**는 다수의 처리기 보드 및 입출력 보드, 네트워킹 보드들이 하나의 챠시(chassis) 안에 장착되는 형태를 가진다. 블레이드 서버와 전통적인 다중 처리기 시스템과의 차이점은 각 블레이드-처리기 보드는 독립적으로 부팅될 수 있고 자기 자신의 운영체제를 수행한다는 것이다. 어떤 블레이드-서버 보드는 자체가 다중 처리기이기도 하며 이 사실은 컴퓨터 유형 간의 경계를 모호하게 만든다. 근본적으로 이 블레이드 서버는 여러 독립적인 다중 처리기 시스템으로 구성된다.

1.3.3 클러스터형 시스템 *Clustered Systems*

여러 CPU를 가진 시스템의 또 다른 유형은 **클러스터형 시스템**이다. 클러스터 시스템은 둘 이상의 독자적 시스템 또는 노드들을 연결하여 구성한다는 점에서 1.3.2절에서 설명한 다중 처리기 시스템과 차이가 난다. 각 노드는 통상 다중 코어 시스템이다. 그러한 시스템은 **약결합**(loosely coupled)이라고 간주된다. **클러스터형**(clustered)의 정의는 분명하지 않다. 많은 상업용 패키지와 공개 소스 패키지들은 클러스터 시스템을 정의하고 왜 한 형태가 다른 형태보다 좋은지에 대한 문제에 답을 제시하는 데 어려움을 겪고 있다. 일반적으로 받아들여지는 정의에 의하면 클러스터 컴퓨터는 저장장치를 공유하고 근거리 통신망(local area network, LAN)이나 InfiniBand와 같은 고속의 상호 연결망(interconnect)으로 연결된다.

클러스터링은 통상 **높은 가용성**(availability)을 제공하기 위해 사용된다. 즉, 클러스터 내 하나 이상의 컴퓨터 시스템이 고장 나더라도 서비스는 계속 제공된다. 일반적으로 높은 가용성은 시스템에 중복 기능을 추가함으로써 얻어진다. 클러스터 소프트웨어 중 한 층이 클러스터 노드에서 실행된다. 각 노드는 하나 이상의 다른 노드(네트워크로 연결되어 있는)들을 감시한다. 만일 감시받던 노드가 고장 나면 감시하던 노드가 고장 난 노드의 저장장치에 대한 소유권을 넘겨받고, 그 노드에서 실행 중이던 응용 프로그램을 다시 시작한다. 사용자와 응용 프로그램의 클라이언트는 잠깐의 서비스 중단만을 경험하게 된다.

높은 가용성은 안정성을 향상해 많은 응용 프로그램에서 중요하다. 남아 있는 하드웨어 수준에 비례하여 서비스를 계속 제공하는 기능을 우아한 **성능 저하**(graceful deg-

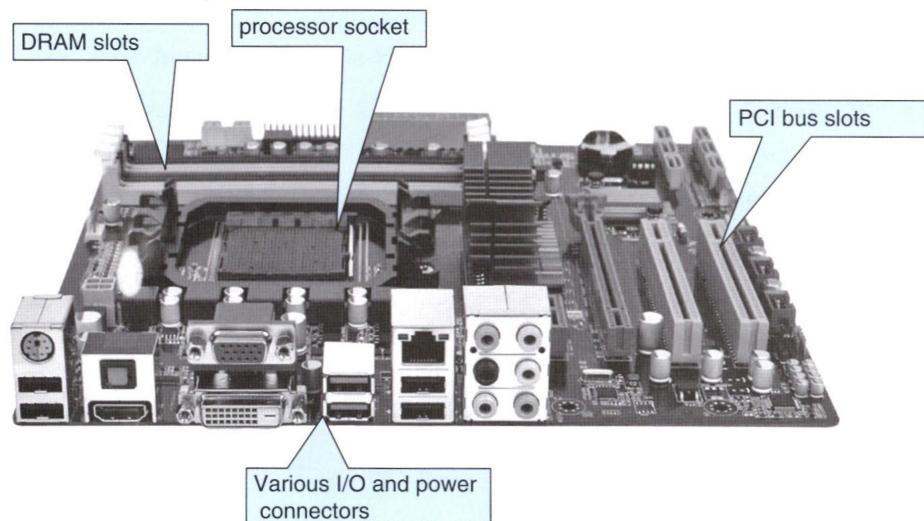
aradation)라고 한다. 일부 시스템은 정상적인 성능 저하를 넘어 단일 구성요소에 오류가 발생하여도 계속 작동할 수 있으므로 결합허용 시스템이라고 한다. 결합허용에는 장애를 감지, 진단 및 가능한 경우 수정할 수 있는 기법이 필요하다.

클러스터링은 비대칭적으로 또는 대칭적으로 구성될 수 있다. **비대칭형 클러스터링**에서는 다른 컴퓨터들이 응용 프로그램을 실행하는 동안 한 컴퓨터는 **긴급 대기** (hot-standby) 모드 상태를 유지한다. 이 긴급 대기 모드의 호스트는 활성 서버들을 감시하는 작업만을 수행한다. 서버가 고장 난다면 긴급 대기 모드의 호스트가 활성 서버가 된다. **대칭형 클러스터링**에서는 둘 이상의 호스트들이 응용 프로그램을 실행하고 서로를 감시한다. 가용한 하드웨어를 모두 사용하기 때문에 대칭형 구성이 더 효율적이다. 대칭형 구성이 효율적으로 동작하기 위해서는 하나 이상의 응용 프로그램들이 실행 가능해야 한다.

한 클러스터가 네트워크로 연결된 다수의 컴퓨터 시스템으로 구성되므로 클러스터는 **고성능 계산** 환경을 제공하도록 사용될 수 있다. 이러한 시스템은 클러스터 내의 모든

PC 마더보드

아래에 표시된 프로세서 소켓이 있는 데스크톱 PC 마더보드를 고려하자.



이 보드는 슬롯이 채워지면 온전히 작동하는 컴퓨터이다. CPU를 포함하는 프로세서 소켓, DRAM 소켓, PCIe 버스 슬롯 및 다양한 유형의 I/O 커넥터로 구성된다. 가장 저렴한 범용 CPU조차도 여러 개의 코어를 포함한다. 일부 마더보드에는 여러 개의 프로세서 소켓이 있다. 고급 컴퓨터일수록 둘 이상의 시스템 보드를 허용하여 NUMA 시스템을 형성한다.

컴퓨터에서 응용을 병렬 수행할 수 있으므로 단일 처리기나 SMP 시스템보다 훨씬 큰 계산 능력을 제공할 수 있다. 그렇지만 응용이 클러스터를 이용할 수 있도록 작성되어야 한다. 이는 **병렬화**라는 기법으로 프로그램을 컴퓨터의 개별 코어에서 혹은 클러스터의 각 컴퓨터에서 수행되는 분리된 요소로 나누는 작업을 포함한다. 전형적으로 이를 응용은 클러스터의 각 계산 노드가 문제 일부를 해결한 후 모든 노드의 결과를 결합하여 최종 해답을 얻게 된다.

다른 형태의 클러스터로 병렬(parallel) 클러스터와 WAN을 이용한 클러스터링이 있다. 병렬 클러스터는 여러 호스트가 공유 저장장치상의 동일한 데이터에 접근할 수 있게 한다. 운영체제의 대부분이 여러 호스트에 의한 이러한 동시 접근을 지원하지 않으므로 병렬 클러스터는 특수 소프트웨어 버전과 특별히 발매된 응용으로 달성된다. 예를 들면, Oracle Real Application Cluster는 병렬 클러스터에서 수행하도록 설계된 Oracle의 데이터베이스 버전이다. 각 기계는 Oracle을 수행하고 하나의 소프트웨어 층이 공유 디스크에 대한 접근을 관리한다. 각 기계는 데이터베이스 내의 모든 데이터에 대한 완전한 접근을 한다. 데이터에 대한 공유 접근을 제공하기 위하여, 시스템은 접근 간의 충돌이 발생하지 않는 것을 보장하기 위하여 접근 제어와 잠금 기법을 제공해야 한다. **분산 잠금 관리자**(distributed lock manager, **DLM**)라고 불리는 이 기능은 몇몇의 클러스터 기술에 포함되어 있다.

클러스터 기술은 급변하고 있다. 어떤 클러스터 제품은 수 킬로미터 떨어진 클러스터 노드들뿐 아니라 한 클러스터 안에서 수천 개의 노드를 지원한다. 이러한 개선은 11.7.4에서 설명될 **스토리지 전용 네트워크**(storage-area network, **SAN**)에 의해 가능해졌다. SAN은 여러 호스트를 여러 저장장치에 부착할 수 있게 한다. 만일 응용과 데이터가 SAN에 저장된다면 클러스터 소프트웨어는 SAN에 연결된 임의의 호스트에서 수행되도록 응용을 배정할 수 있다. 호스트가 고장 나면 다른 호스트가 그 응용을 넘겨 받는다. 데이터베이스 클러스터에서는 수십 개의 호스트가 동일한 데이터베이스를 공유할 수 있기 때문에 성능과 신뢰도를 매우 증가시키게 된다. 그림 1.11은 클러스터 시스템의 일반 구조를 보인다.

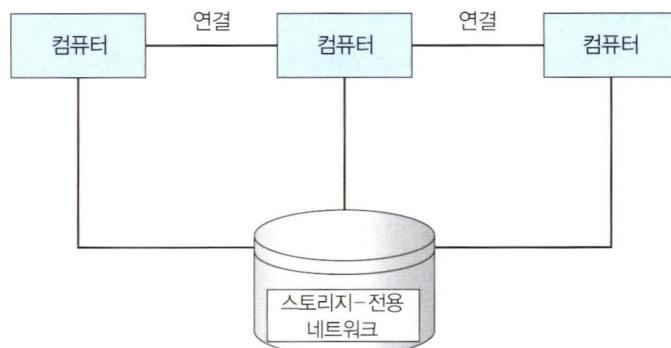


그림 1.11 클러스터 시스템의 일반적인 구조

1.4 운영체제의 작동 *Operating-System Operations*

컴퓨터 시스템 구성 및 아키텍처에 대한 기본 정보를 살펴보았으므로 이제 운영체제에 대해 이야기할 준비가 되었다. 운영체제는 프로그램이 실행되는 환경을 제공한다. 내부적으로 운영체제는 여러 경로를 거쳐 구성되기 때문에 운영체제마다 큰 차이를 보인다. 그러나 많은 공통점이 존재하고 이 절에서 그 공통점들에 대해 논의한다.

컴퓨터의 전원을 켜거나 재부팅 할 때와 같이 컴퓨터를 실행하려면 초기 프로그램을 실행해야 한다. 앞에서 언급했듯이 이 초기 프로그램 또는 부트스트랩 프로그램은 단순한 형태를 띠는 경향이 있다. 일반적으로 컴퓨터 하드웨어 내에 펌웨어로 저장된다. CPU 레지스터에서 장치 컨트롤러, 메모리 내용에 이르기까지 시스템의 모든 측면을 초

HADOOP

Hadoop은 단순하고 저렴한 하드웨어 구성요소를 포함하는 클러스터형 시스템에서 대용량 데이터 세트(빅데이터라고 함)의 분산 처리에 사용되는 공개 소스 소프트웨어 프레임워크이다. Hadoop은 단일 시스템에서 수천 개의 컴퓨팅 노드를 포함하는 클러스터로 확장되도록 설계되었다. 작업은 클러스터의 노드에 할당되며 Hadoop은 노드 간 통신을 정렬하여 처리할 병렬 계산을 관리하고 결과를 통합한다. Hadoop은 또한 노드의 장애를 감지하고 관리하여 효율적이고 매우 안정적인 분산 컴퓨팅 서비스를 제공한다.

Hadoop은 다음 세 가지 구성요소로 구성된다.

1. 분산 컴퓨팅 노드에서 데이터와 파일을 관리하는 분산 파일 시스템.
2. YARN (“Yet Another Resource Negotiator”) 프레임워크는 클러스터 내의 자원을 관리하고 클러스터의 노드에 작업을 스케줄 한다.
3. **MapReduce** 시스템은 클러스터의 노드에서 데이터를 병렬 처리할 수 있게 한다.

Hadoop은 Linux 시스템에서 실행되도록 설계되었으며 Hadoop 응용 프로그램은 PHP, Perl 및 Python과 같은 스크립팅 언어를 비롯한 여러 프로그래밍 언어를 사용하여 작성할 수 있다. MapReduce를 지원하는 여러 Java 라이브러리를 가지고 있으므로 Hadoop 응용 프로그램 개발에 Java가 널리 사용된다. MapReduce 및 Hadoop에 대한 자세한 정보는 https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html 및 <https://hadoop.apache.org>에서 확인 할 수 있다.

기화한다. 부트스트랩 프로그램은 운영체제를 적재하는 방법과 해당 시스템 실행을 시작하는 방법을 알아야 한다. 이 목표를 달성하려면 부트스트랩 프로그램이 운영체제 커널을 찾아 메모리에 적재해야 한다.

커널이 적재되어 실행되면 시스템과 사용자에게 서비스를 제공할 수 있다. 일부 서비스는 커널이 실행되는 전체 시간 동안 실행되는 **시스템 데몬**이 되기 위해 부팅할 때 메모리에 적재되는 시스템 프로그램에 의해 커널 외부에서 제공된다. Linux에서 첫 번째 시스템 프로그램은 “**systemd**”이며 다른 많은 데몬을 시작한다. 이 단계가 완료되면 시스템이 완전히 부팅되고 시스템은 어떤 이벤트가 발생할 때까지 기다린다.

실행할 프로세스, 서비스할 I/O 장치 및 응답할 사용자가 없는 경우 운영체제는 조용히 앉아 무언가가 발생할 때까지 기다린다. 이벤트는 거의 항상 인터럽트를 발생시켜 신호를 보낸다. 1.2.1절에서 하드웨어 인터럽트에 관해 설명하였다. 또 다른 형태의 인터럽트는 **트랩**(또는 **예외**)으로, 오류(예: 0으로 나누거나 유효하지 않은 메모리 액세스) 또는 사용자 프로그램의 특정 요청 때문에 발생하는 소프트웨어 생성 인터럽트이다. 이 특정 요청은 **시스템 콜**이라는 특수 연산을 실행하여 요청되고 운영체제가 제공하는 서비스가 수행될 것을 요구한다.

1.4.1 다중 프로그래밍과 다중 태스킹 *multiprogramming and multitasking*

운영체제의 가장 중요한 측면 중 하나는 하나의 프로그램은 일반적으로 항상 CPU나 I/O 장치를 항상 바쁘게 유지할 수 없으므로 여러 프로그램을 실행할 수 있다는 것이다. 또한 사용자는 일반적으로 한 번에 둘 이상의 프로그램을 실행하려고 한다. **다중 프로그래밍**은 CPU가 항상 한 개는 실행할 수 있도록 프로그램을 구성하여 CPU 이용률을 높이고 사용자 만족도를 높인다. 다중 프로그램 시스템에서 실행 중인 프로그램을 **프로세스**라고 한다.

운영체제는 여러 프로세스를 동시에 메모리에 유지한다(그림 1.12). 운영체제는 이러한 프로세스 중 하나를 선택하여 실행하기 시작한다. 결국 프로세스는 I/O 작업과 같은 일부 작업이 완료되기를 기다려야 할 수도 있다. 다중 프로그래밍 되지 않은 시스템에서는 CPU가 유휴 상태일 수 있다. 다중 프로그램 시스템에서 운영체제는 단순히 다른 프로세스로 전환하여 실행한다. 해당 프로세스가 대기해야 하는 경우 CPU는 다른 프로세스로 전환한다. 결국 첫 번째 프로세스는 대기를 마치고 CPU를 다시 돌려받는다. 하나 이상의 프로세스를 실행해야 하는 한 CPU는 유휴 상태가 아니다.

이 아이디어는 다른 일상생활에서도 흔히 볼 수 있다. 변호사는 한 번에 단지 한 사람의 의뢰인만 상대하는 것이 아니다. 한 사건이 재판을 기다리거나 문서 작성 등 동안, 변호사는 또 다른 사건을 처리할 수 있다. 만일 변호사가 충분히 많은 의뢰인을 확보하고 있다면 변호사가 일이 없어서 쉬는 일은 발생하지 않는다. (쉬는 변호사들은 정치가가 되는 경향이 있다. 따라서 변호사를 바쁘게 만드는 일은 약간의 사회적 가치가 있다).

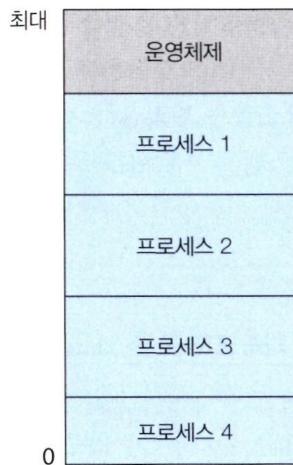


그림 1.12 다중 프로그래밍 시스템을 위한 메모리 배치

다중 태스킹(multitasking)은 다중 프로그래밍의 논리적 확장이다. 다중 태스킹 시스템에서 CPU는 여러 프로세스를 전환하며 프로세스를 실행하지만 전환이 자주 발생하여 사용자에게 빠른 응답 시간을 제공하게 된다. 프로세스가 실행될 때 일반적으로 프로세스가 완료되거나 I/O를 수행하기 전에 짧은 시간 동안만 실행된다는 것을 고려하자. 입출력은 대화식일 수 있다. 즉, 출력이 사용자를 위해 디스플레이되고, 입력은 사용자 키보드, 마우스 또는 터치스크린으로부터 들어온다. 이러한 대화식 입출력은 전형적으로 사람의 속도로 수행되므로, 완료 시까지 상당히 긴 시간이 걸릴 수 있다. 예를 들면 입력은 사용자의 타이핑 속도에 제한을 받으며, 사람에게는 1초에 7문자가 빠른 편이지만, 컴퓨터에게는 아주 느린 것이다. 이러한 대화식 입력이 진행되는 동안 CPU를 쉬게 하지 않고, 운영체제는 CPU를 다른 사용자의 프로그램으로 신속하게 전환한다.

동시에 여러 프로세스를 메모리에 유지하려면 9장과 10장에서 다루는 메모리 관리 방식이 필요하다. 또한 여러 프로세스가 동시에 실행할 준비가 되면 시스템은 다음에 실행할 프로세스를 선택해야 한다. 이 결정을 내리는 것은 5장에서 논의된 **CPU 스케줄링**이다. 마지막으로, 여러 프로세스를 병행하게 실행하려면 프로세스 스케줄링, 디스크 저장장치 및 메모리 관리를 포함하여 운영체제의 모든 단계에서 서로 영향을 미치는 기능이 제한되어야 한다. 우리는 이 책 전반에 걸쳐 이러한 고려 사항에 대해 논의한다.

다중 태스킹 시스템에서 운영체제는 적절한 응답 시간을 보장해야 한다. 적절한 응답 시간을 보장하는 더 일반적인 방법은 **가상 메모리**(virtual memory)인데, 이것은 일부만 메모리에 적재된 프로세스의 실행을 허용하는 기법이다(10장). 이 기법의 주요한 이점은 프로그램이 **물리 메모리**의 크기보다 더 커도 된다는 것이다. 더욱이 가상 메모리는 메인 메모리를 크고 균등한 저장장치의 배열로 추상화하여, 사용자에게 보이는 **논리 메모리**를 물리 메모리로부터 분리시킨다. 이러한 기법은 프로그래머를 메모리 저장장치의 한계로부터 자유롭게 해준다.

다중 프로그래밍 및 다중 태스킹 시스템도 파일 시스템을 제공해야 한다(13장, 14장 및 15장). 파일 시스템은 보조저장장치에 존재한다. 따라서 저장장치 관리가 제공되어야 한다(11장). 또한 시스템은 부적절한 사용으로부터 자원을 보호해야 한다(17장). 질서 있게 실행하려면 시스템은 프로세스 동기화 및 통신을 위한 기법을 제공해야 하며(6장 및 7장) 프로세스가 서로를 영원히 기다리는 교착 상태에 빠지지 않도록 보장해야 한다(8장).

1.4.2 이중-모드와 다중모드 운용 *Dual-Mode and Multimode Operation*

운영체제와 사용자는 컴퓨터 시스템의 하드웨어 및 소프트웨어 자원을 공유하기 때문에 올바르게 설계된 운영체제는 잘못된 (또는 악의적인) 프로그램으로 인해 다른 프로그램 또는 운영체제 자체가 잘못 실행될 수 없도록 보장해야 한다. 시스템을 올바르게 실행하려면 운영체제 코드 실행과 사용자-정의 코드 실행을 구분할 수 있어야 한다. 대부분의 컴퓨터 시스템이 취하는 접근 방식은 다양한 실행 모드를 차별화 할 수 있는 하드웨어 지원을 제공하는 것이다.

적어도 두 개의 독립된 연산 모드, 즉 **사용자 모드**와 **커널 모드**[수퍼바이저 모드, 시스템 모드, 혹은 특권 모드(privileged mode)]로도 부른다]를 필요로 한다. **모드 비트** (mode bit)라고 하는 하나의 비트가 현재의 모드를 나타내기 위해 컴퓨터의 하드웨어에 추가되었다. 이 비트는 커널 모드(0) 또는 사용자 모드(1)를 나타낸다. 모드 비트의 사용으로, 우리는 운영체제를 위하여 실행되는 작업과 사용자를 위해 실행되는 작업을 구분할 수 있다. 컴퓨터 시스템이 사용자 응용을 위하여 실행될 때 시스템은 사용자 모드에 있게 된다. 그러나 사용자 응용이 운영체제로부터 서비스를 요청하면(시스템 콜을 통함) 이 요청을 수행하기 위해서는 사용자 모드에서 커널 모드로 전환해야 한다. 이 절차가 그림 1.13에 나와 있다. 앞으로 살펴보겠지만 이러한 구조상의 개선은 시스템 동작의 여러 측면에서 유용하다.

시스템 부트 시, 하드웨어는 커널 모드에서 시작한다. 이어 운영체제가 적재되고, 사용자 모드에서 사용자 프로세스가 시작된다. 트랩이나 인터럽트가 발생할 때마다, 하드웨어는 사용자 모드에서 커널 모드로 전환한다(즉, 모드 비트를 0으로 변경). 그러므로

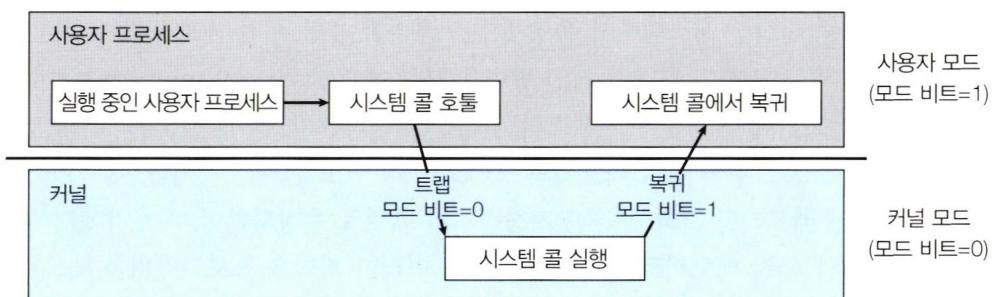


그림 1.13 사용자 모드에서 커널 모드로의 전환