

Chapter 03

운영체제



목차

01 운영체제 개요

02 프로세스 관리

03 메인메모리 관리

04 저장 장치 관리

학습목표

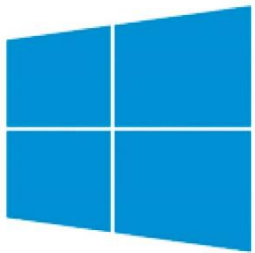
- 운영체제가 무엇이며, 왜 필요한지 알아본다.
- 커널 같은 운영체제의 구성 요소와 특징을 알아본다.
- 프로세스가 무엇인지 살펴본다.
- 메모리 관리를 학습한다.
- 가상 메모리와 스왑의 특징을 살펴본다.
- 저장 장치 관리를 학습한다.

■ 운영체제(OS, Operating System)

- 컴퓨터나 노트북 전원을 켜면 가장 먼저 만나는 소프트웨어

■ 운영체제 종류

- 대형 컴퓨터 : 유닉스
- 개인용 컴퓨터 : 윈도우, 맥 OS, 리눅스 등
- 스마트폰(모바일 운영체제) : 애플의 iOS와 구글의 안드로이드



윈도우



맥OS



리눅스



iOS



안드로이드

그림 6-1 다양한 운영체제 제품

■ 임베디드 운영체제

- CPU 성능도 낮고 메모리 크기도 작아 특정 시스템에 내장할 수 있도록 만든 운영체제
- MP3 플레이어, PMP(Personal Media Player), 내비게이션, 스마트 시계, 스마트 TV 등에 사용
- 일반 운영체제에 비하여 몇 가지 기능이 빠짐



(a) 스마트 시계



(b) 스마트 TV

그림 6-2 임베디드 운영체제의 예

■ 운영체제 역할

- 성능 향상 : 새로운 기능 추가, 성능을 변경이 가능하므로 성능 및 효율성 향상을 꾀할 수 있음
- 자원 관리 : 제한된 장치를 서로 독차지하려고 하는 응용 프로그램 사이에서 자원을 관리하는 중재자 역할을 함
- 자원 보호 : 자원을 악의적인 혹은 미숙한 사용자에게서 자원을 보호
- 사용자 인터페이스 제공 : 컴퓨터 하드웨어와 소프트웨어를 편리하게 사용할 수 있도록 환경 제공

■ 운영체제 정의

- 사용자에게 편리한 인터페이스를 제공하고 자원을 효율적으로 관리하는 소프트웨어

■ 운영체제 핵심기능

- 운영체제 핵심기능은 프로세스, 메인메모리, 저장장치 관리임.
- 요리사모형에서 주방장(CPU)는 보관창고(저장장치)의 재료를 도마(메인 메모리)로 가져와서 실행함.
- CPU가 작업을 실행하는 것은 프로세스 관리
- 도마위의 재료를 정리하는 것은 메인메모리 관리
- 저장장치에 데이터를 효율적으로 정리하는 것은 저장장치 관리.

표 6-1 운영체제의 핵심 기능

핵심 기능	설명
프로세스 관리	프로세스에 CPU를 배분하고 작업에 필요한 제반 환경을 제공한다.
메인메모리 관리	프로세스에 작업 공간을 배치하고 실제 메인메모리보다 큰 가상 공간을 제공한다.
저장 장치 관리	데이터를 저장하고 접근할 수 있는 인터페이스를 제공한다.

■ 운영체제 구조

- 운영체제는 크게 커널과 사용자 인터페이스로 구성
- 커널 내에는 드라이버와 시스템 호출 기능이 있음



그림 6-4 운영체제 구조

■ 커널

- 프로세스 관리, 메모리 관리, 저장 장치 관리 같은 운영체제의 핵심 기능을 모아 놓은 것

■ 사용자 인터페이스

- 운영체제가 사용자와 응용 프로그램에 인접하여 커널에 명령을 전달하고, 실행 결과를 사용자와 응용 프로그램에 돌려주는 기능

■ 운영체제 특징

- 모든 응용 프로그램은 운영체제 위에서 작동하기 때문에 운영체제가 불안정하면 다른 응용 프로그램도 함께 불안정
- 운영체제는 바이러스나 악의적인 소프트웨어에서 하드웨어뿐 아니라 자기 자신도 보호
- 운영체제는 사용자가 직접 자원에 접근하는 것을 막음으로써 자원을 보호

■ 드라이버

- 운영체제가 하드웨어 장치와 상호 작용하려고 만든 컴퓨터 프로그램
- 장치 드라이버, 디바이스 드라이버, 장치 제어기, 소프트웨어 드라이버라고도 함
- 그래픽 카드나 프린터 같이 복잡한 드라이버는 CD 형태로 제공



그림 6-6 NVIDIA 그래픽 카드 드라이버 다운로드 웹 사이트

■ 유닉스

- 여러 명이 동시에 사용할 수 있는 운영체제로 1969년 AT&T의 연구원인 켄 톰프슨(Ken Thompson)이 개발

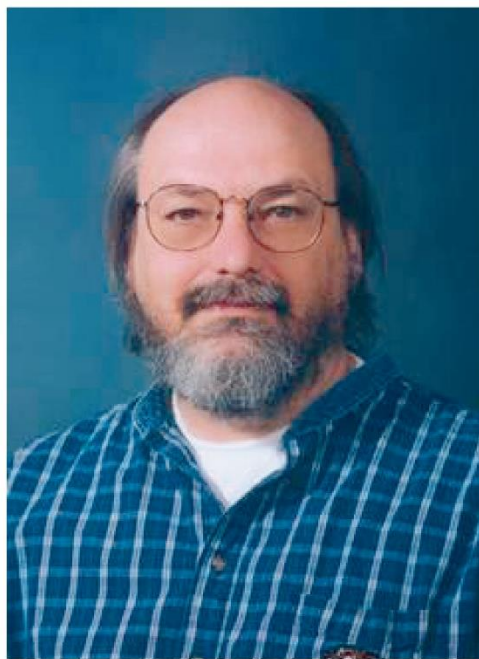


그림 6-7 유닉스 개발자 켄 톰프슨(왼쪽)과 리눅스 개발자 리누스 토르발스(오른쪽)

■ 유닉스 계열

▪ System V

- 소스 코드를 공개, 많은 사람이 개선 발전시켜 현재까지 내려온 유닉스

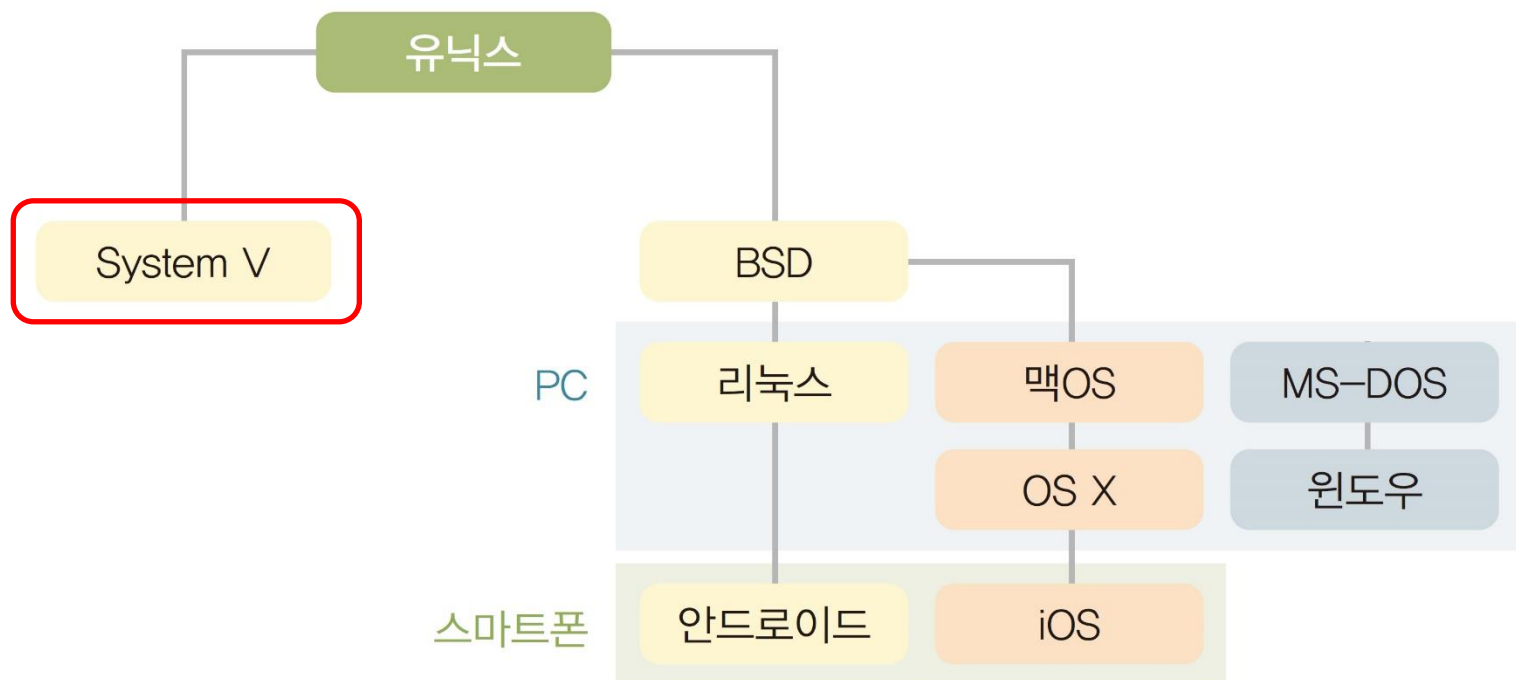


그림 6-8 운영체제의 역사

■ 유닉스 계열

- BSD(Berkeley Software Distribution)
 - 빌 조이(Bill Joy)와 척 헤일리(Chuck Haley) 학생이 소스 코드를 수정하여 만든 유닉스(1978년 출시)
 - 계속 발전하여 누구나 공짜로 사용할 수 있는 FreeBSD로 발전

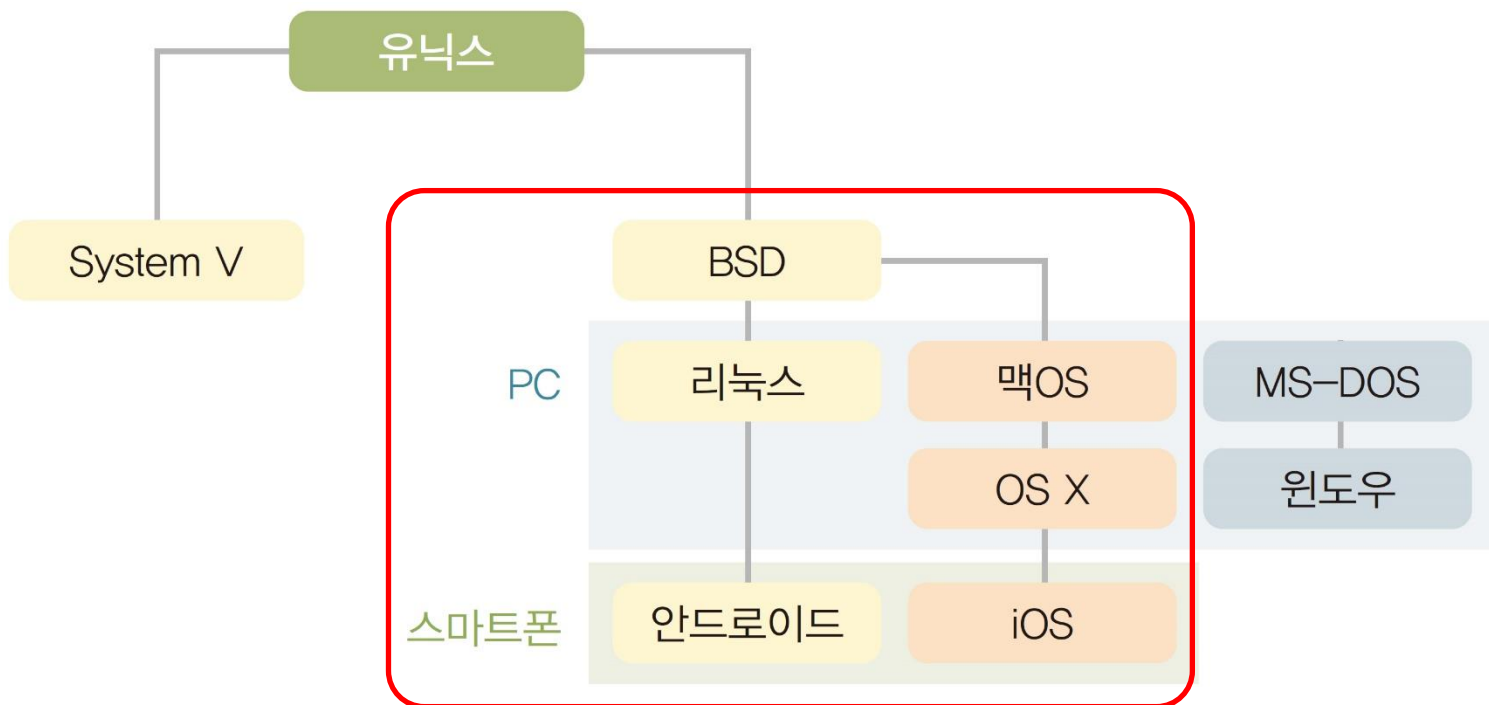


그림 6-8 운영체제의 역사

■ 리눅스

- 1991년 리누스 토르발스(Linus Torvalds)가 개인용 컴퓨터에서 동작하는 유닉스 호환 커널을 만들어 공개
- FreeBSD를 기반으로 만들었기 때문에 누구나 공짜 사용, 소스 코드 공개
- 구글이 리눅스 커널을 가져다가 스마트폰에서 사용할 수 있는 안드로이드 운영체제를 개발

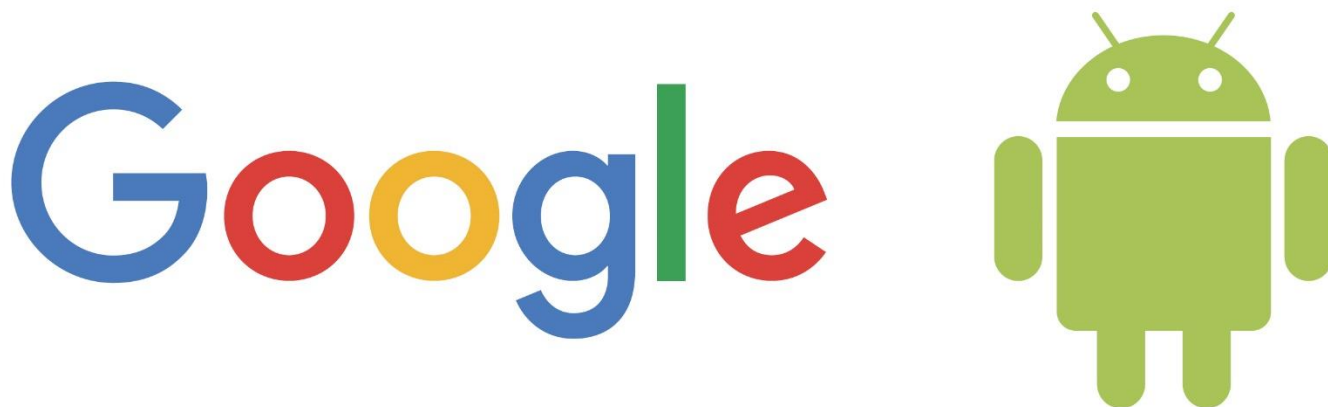


그림 6-9 구글과 안드로이드

■ 맥 OS

- 애플은 FreeBSD를 변형하여 매킨토시 컴퓨터에서 동작하는 운영체제 개발
- 현재 'OS X'으로 발전하여 애플용 컴퓨터와 노트북에서 사용
- iOS : OS X을 스마트폰용으로 바꾼 것

■ 리눅스(Linux) 배포판

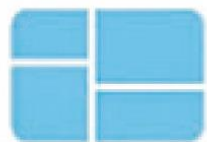
1. 리눅스 커널, GNU소프트웨어 및 여러 가지 자유 소프트웨어로 구성된 운영체제이다.
2. 현재 전 세계적으로 300여 가지의 배포판이 존재한다.

● 리눅스(Linux) 배포판 종류

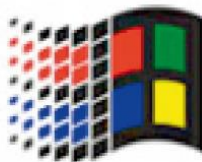
- Debian Linux(데비안 리눅스)
- Ubuntu Linux(우분투 리눅스)
- RedHat Linux(레드햇 리눅스)
- Centos(센트OS)
- Kali Linux(칼리 리눅스)

■ 윈도우

- MS-DOS에 그래픽 사용자 인터페이스(GUI)를 붙인 것
- 윈도우는 3.1로 시작하여 현재 10 버전까지 출시



Windows1
1985



Windows 3.1
1992



Windows 95
1995



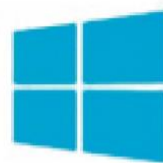
Windows XP
2001



Windows
Vista 2006



Windows 7
2009



Windows 8
2012



Windows 10
2015

그림 6-10 윈도우의 버전별 로고

■ 윈도우

- 마이크로소프트의 몇몇 연구원이 개발해서 무겁고 불안한 운영체제
- 윈도우 운영체제가 중단되면 파란 화면이 나타나고, 작업하던 모든 데이터가 사라짐
- 무료 사용이 가능한 리눅스는 안정적이고 강력한 운영체제지만 일반인이 사용하기 어려워서 윈도우를 유료로 사용

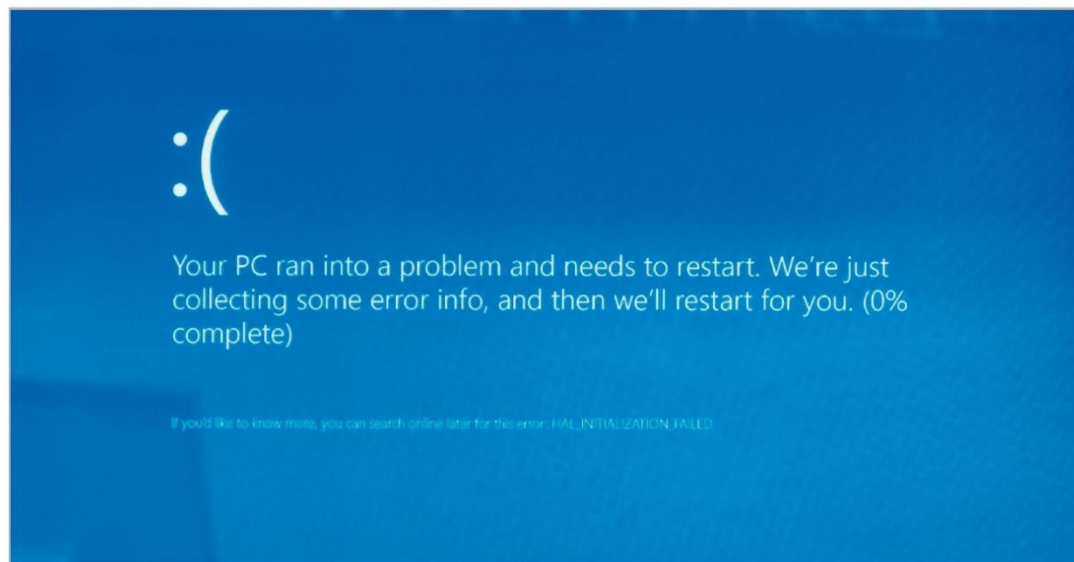


그림 6-11 윈도우 블루 스크린

■ 프로세스

- 프로세스(process)는 하나의 작업 단위
- 사용자가 마우스를 더블클릭하여 프로그램(program)을 실행하면 그 프로그램은 프로세스가 됨



(a) 프로그램 실행



(b) 프로세스 종료

그림 6-12 프로그램 실행과 프로세스 종료

■ 프로그램과 프로세스의 비유

- 레시피 → 조리 → 요리
- 프로그램 → 생성 → 프로세스



그림 6-13 레시피와 요리에 비유한 프로그램과 프로세스

■ 프로그램

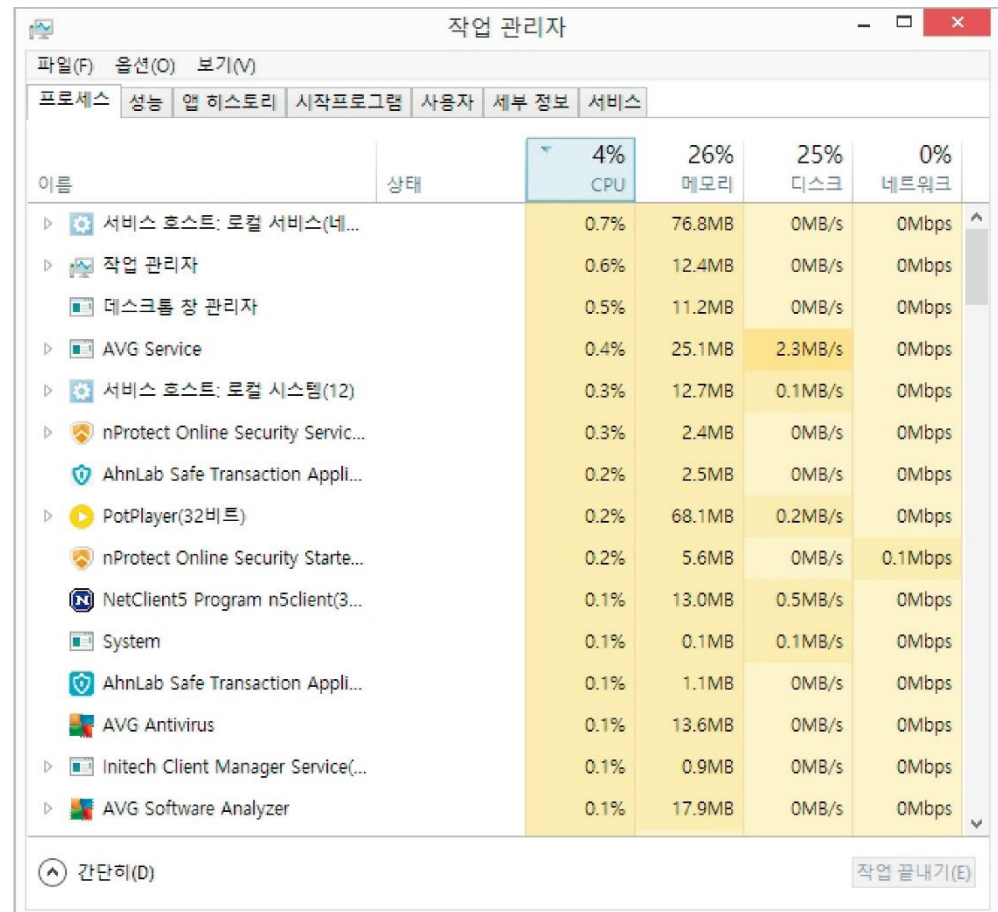
- 어떤 데이터를 사용하여 어떤 작업을 할지 그 절차를 적어 놓은 것
- 하드디스크 같은 저장 장치에 보관하고 있다가 마우스로 더블클릭하면 실행

■ 프로세스

- 프로그램으로 작성된 작업 절차를 실제로 실행에 옮긴다는 의미
(‘실행한다’고 표현)
- 해당 코드가 메모리에 올라와서 작업이 진행된다는 의미

■ 프로그램과 프로세스 차이

- 엑셀은 하드디스크에 저장된 프로그램 중 하나
 - 저장 장치에 보관된 정적인 상태
- 엑셀을 실행시키면
프로세스가 됨
 - 메모리에 올라와서 작업을 수행하는 동적인 상태



The screenshot shows the Windows Task Manager window titled '작업 관리자' (Task Manager). The '프로세스' (Processes) tab is selected. The table below represents the data shown in the task manager, sorted by CPU usage.

이름	상태	CPU	메모리	디스크	네트워크
서비스 호스트: 로컬 서비스(네...		4%	26%	25%	0%
작업 관리자		0.7%	76.8MB	0MB/s	0Mbps
데스크톱 창 관리자		0.6%	12.4MB	0MB/s	0Mbps
AVG Service		0.5%	11.2MB	0MB/s	0Mbps
서비스 호스트: 로컬 시스템(12)		0.4%	25.1MB	2.3MB/s	0Mbps
nProtect Online Security Servic...		0.3%	12.7MB	0.1MB/s	0Mbps
AhnLab Safe Transaction Appli...		0.3%	2.4MB	0MB/s	0Mbps
PotPlayer(32비트)		0.2%	2.5MB	0MB/s	0Mbps
nProtect Online Security Starte...		0.2%	68.1MB	0.2MB/s	0Mbps
NetClient5 Program n5client(3...		0.2%	5.6MB	0MB/s	0.1Mbps
System		0.1%	13.0MB	0.5MB/s	0Mbps
AhnLab Safe Transaction Appli...		0.1%	0.1MB	0.1MB/s	0Mbps
AVG Antivirus		0.1%	1.1MB	0MB/s	0Mbps
Initech Client Manager Service(...		0.1%	13.6MB	0MB/s	0Mbps
AVG Software Analyzer		0.1%	0.9MB	0MB/s	0Mbps
AVG Software Analyzer		0.1%	17.9MB	0MB/s	0Mbps

그림 6-14 윈도우 작업 관리자 화면

■ 일괄 처리 작업(batch job)

- 한 번에 작업을 1개만 처리하는 시스템
- 컴퓨터 초창기의 운영체제나 MS-DOS가 사용하던 방식

■ 시분할 작업(timesharing job)

- 프로세스 여러 개가 아주 짧은 시간 동안 CPU를 사용하는 방식
- 프로세스 여러 개가 동시에 실행되는 것처럼 보이는 작업

■ 시분할 작업



그림 6-15 시분할 작업의 주문서 처리

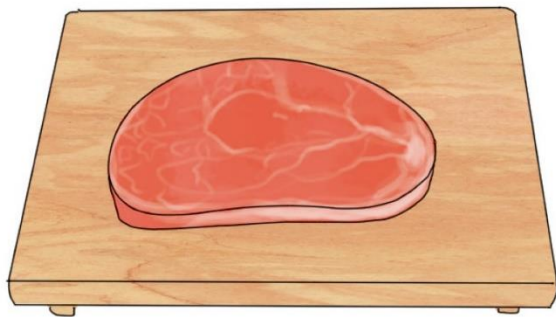
- 프로세스 제어 블록(Process Control Block, PCB)
 - 운영체제에서 주문서에 해당하는 것

■ 프로세스 상태

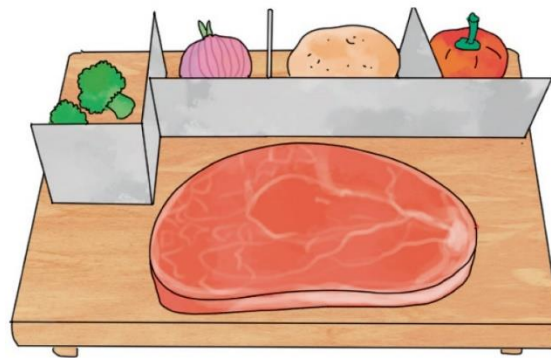
- 실행 상태(running status)
 - 준비 상태에 있는 프로세스 중 하나가 CPU를 얻어 실제 작업을 수행하는 상태
 - 자신의 작업이 끝날 때까지 준비 상태와 실행 상태를 반복
- 완료 상태(terminate status)
 - 실행 상태의 프로세스가 주어진 시간 동안 작업을 마치면 완료 상태로 진입, PCB 반납

■ 메모리 관리

- 폰노이만 구조의 컴퓨터에서 메모리는 유일한 작업 공간, 모든 프로그램은 메모리에 올라와야 실행 가능
- 과거 일괄 처리 시스템에서는 한 번에 작업 하나만 처리했기 때문에 메모리 관리가 어렵지 않음
- 오늘날 시분할 시스템에서는 운영체제를 포함한 여러 프로세스가 한꺼번에 메모리에 올라오기 때문에 메모리 관리가 복잡



(a) 일괄 처리 시스템



(b) 시분할 시스템

그림 6-17 일괄 처리 시스템과 시분할 시스템의 메모리 관리 예

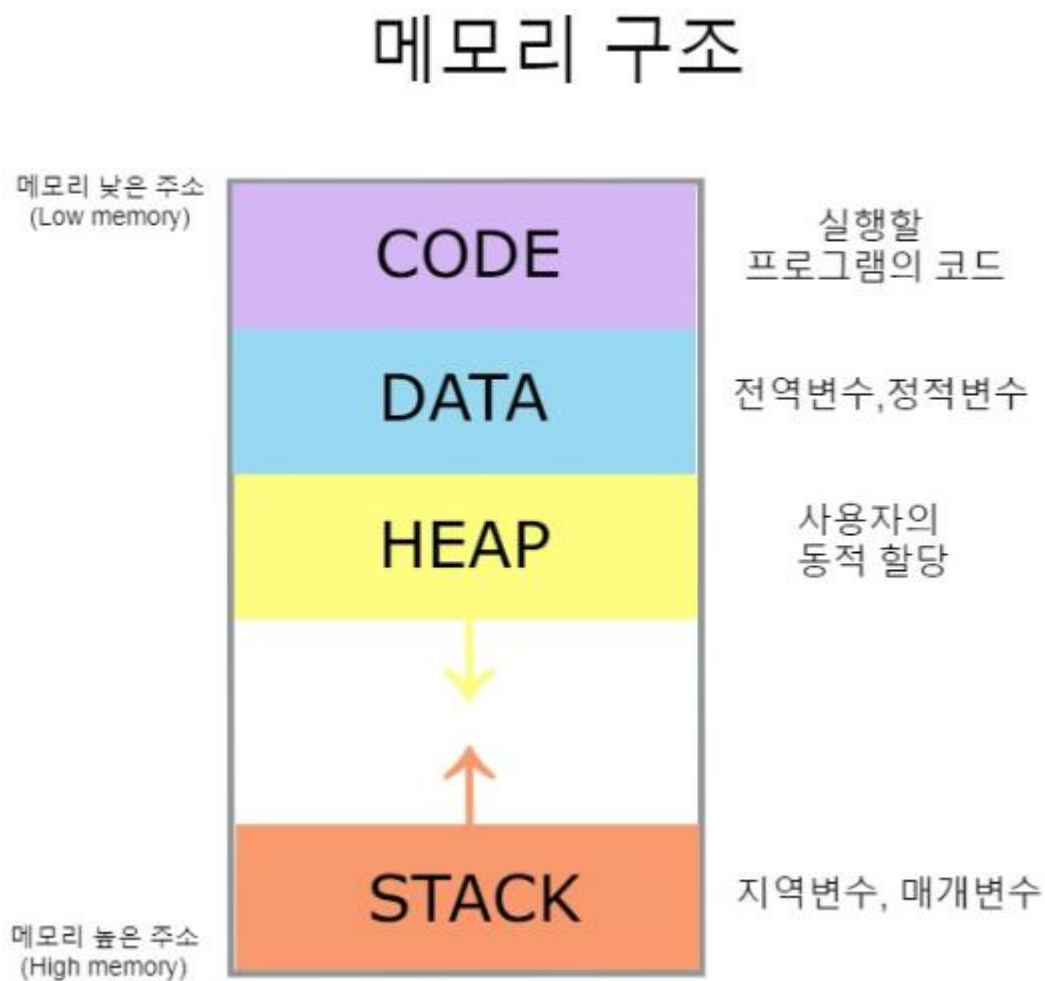
■ 메모리 구조

- 운영체제도 프로세스이기 때문에 메모리로 올라와야 실행 가능
- 메모리 관리 시스템(Memory Management System, MMS)
 - 운영체제 영역과 다른 작업 영역 침범 막기
 - 자리 부족 시 빈 공간 확보



그림 6-18 일괄 처리 시스템과 시분할 시스템의 메모리 구조

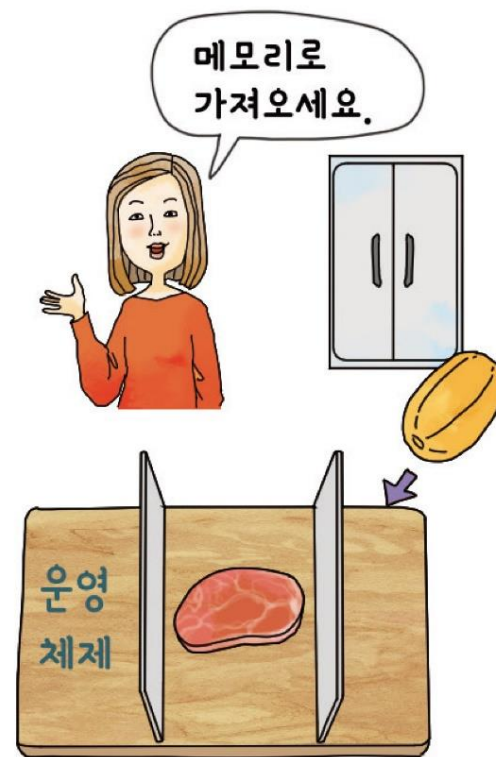
■ 자바의 메모리 구조



■ 메모리 관리자가 하는 일

■ 가져오기 작업

- 프로세스와 데이터를 메모리로 가져오는 것



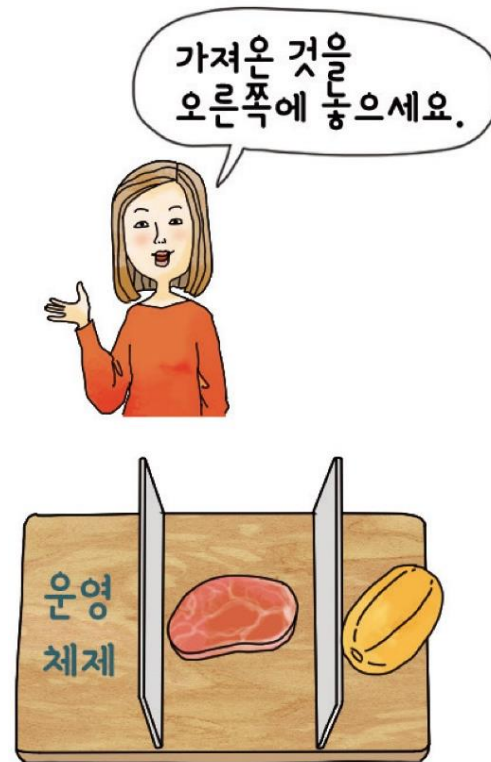
(a) 가져오기

그림 6-19 메모리 관리자가 하는 작업

■ 메모리 관리자가 하는 일

■ 배치 작업

- 가져온 프로세스와 데이터를
메모리의 어떤 부분에 올려놓을지 결정하는 것
- 배치 작업 전에 같은 크기로 자르느냐,
프로세스 크기에 맞게 자르느냐에 따라
복잡성이 달라짐

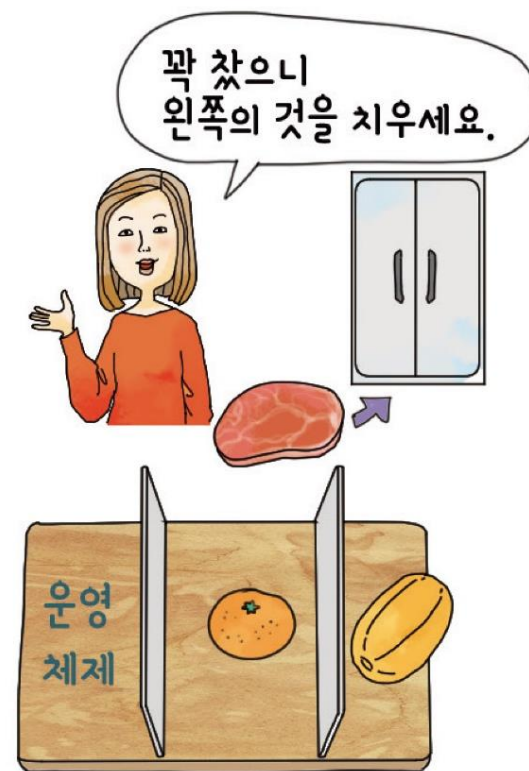


(b) 배치

■ 메모리 관리자가 하는 일

■ 재배치 작업

- 꽉 차 있는 메모리에 새로운 프로세스를 가져오려고 오래된 프로세스를 내보내는 작업



(c) 재배치

■ 메모리 크기를 고려한 개발

- 사용자 메모리는 1GB부터 16GB까지 다양
- 동작 프로그램에 따라 작은 메모리에서는 동작하지 않을 수 있음
- 메모리 크기를 고려하여 프로그래밍하기는 매우 어려움



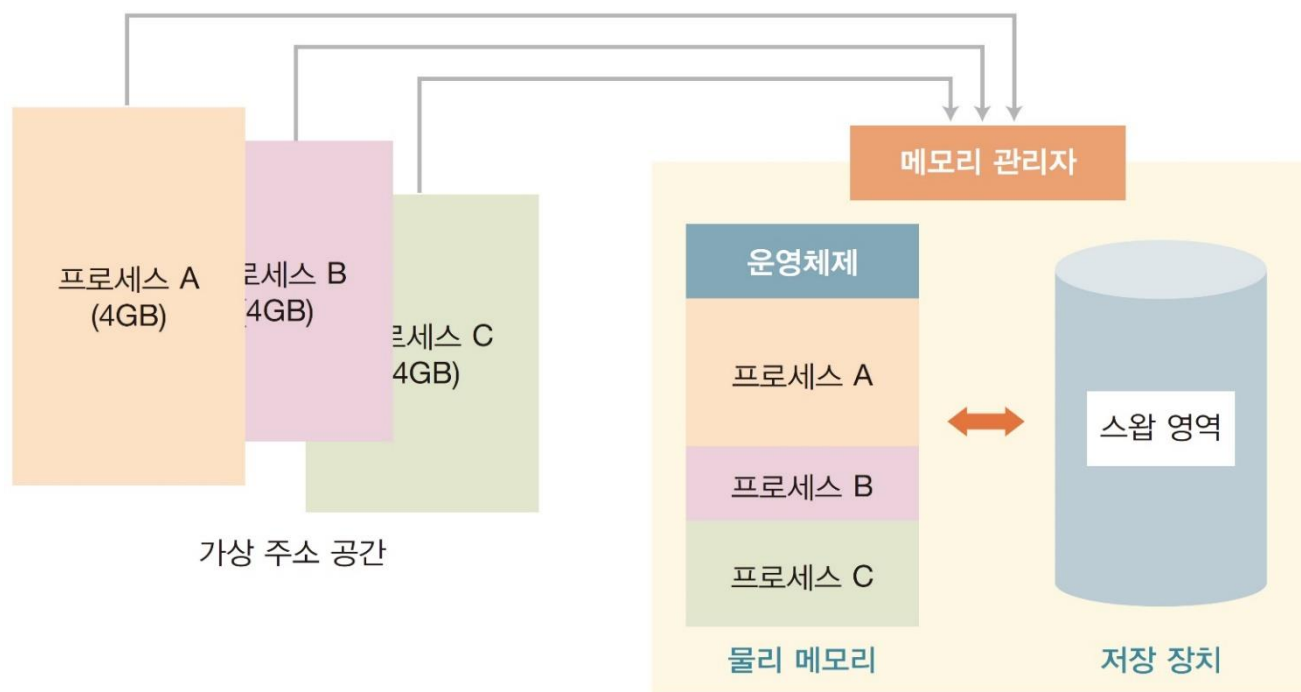
그림 6-20 주방 크기를 고려한 레시피 개발의 어려움

■ 가상 메모리(virtual memory)

- 사용자가 가지고 있는 실제 메모리 크기와 프로세스가 올라갈 메모리 위치를 신경 쓰지 않고 프로그래밍을 하도록 지원하는 메모리 시스템
- 실제 메모리 크기와 상관없이 프로세스에 커다란 메모리 공간을 제공하는 기술

■ 가상 메모리 구성

- 가상 메모리는 <프로세스가 바라보는 메모리 영역>과 <메모리 관리자가 바라보는 메모리 영역>으로 나뉜다
- 이론적으로 가상 메모리 크기는 무한대



(a) 프로세스가 바라보는 메모리 영역

(b) 메모리 관리자가 바라보는 메모리 영역

그림 6-21 가상 메모리 구성

■ 스왑 영역(swap area)

- 메모리가 모자라서 쫓겨난 프로세스를 저장 장치의 특별한 공간에 모아 두는 영역
- 하드디스크 같은 저장 장치는 장소만 빌려주고 메모리 관리자가 담당

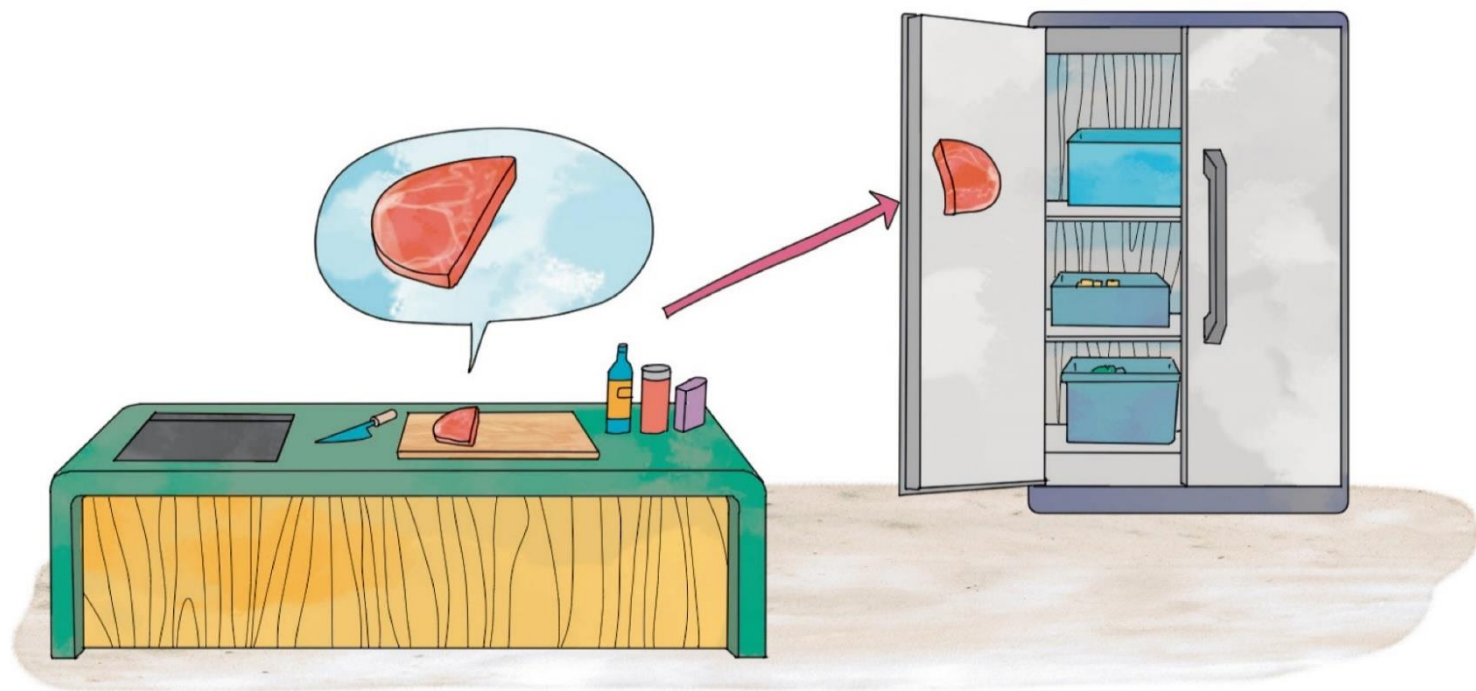


그림 6-22 도마에서 손질 중인 고깃덩어리 일부를 다시 보관 창고에 가져다 놓는 모습

■ 가상 메모리 크기

- 가상 메모리 시스템은 실제 메모리와 스왑 영역을 활용하여 메모리 크기에 상관없이 모든 프로그램을 실행할 수 있는 시스템

가상 메모리의 크기

가상 메모리 크기 = 실제 메모리 크기 + 스왑 영역 크기

■ 파일 확장자

- 파일은 논리적인 데이터 집합으로 하드디스크나 CD 같은 제2 저장 장치에 저장
- 파일 구분은 확장자를 사용하고 확장자에 따라 파일 성격 구분
 - ** 표기 : 파일이름.확장자
- 초창기 운영체제에서는 파일 이름은 여덟 글자, 확장자는 세 글자로 제한
- 파일 이름
 - 마지막 마침표 다음 글자를 확장자로 인식
 - 현재 경로 이름을 포함하여 최대 255자
 - ₩, /, :, *, ?, “, <, >, | 등은 사용 불가

파일 이름에는 다음 문자를 사용할 수 없습니다.

₩ / : * ? “ < > |

그림 6-25 파일 이름 오류 메시지

■ 파일 포맷의 종류

- 이미지 파일 포맷 : BMP, JPG, GIF, PNG 등
- 음악 파일 포맷 : MP3, WAV, OGG, AAC, FLAC

■ 파일 헤더

- 파일 이름, 버전, 크기, 만든 날짜 등 정보가 저장



그림 6-26 파일 구조

■ 파일 확장자

- mp3, bmp, jpg, zip 등과 같이 전 세계적으로 표준을 정하여 사용
- 응용 프로그램 제작자가 필요에 따라 새로 만들어 사용 가능

표 6-2 파일 종류와 확장자

파일	확장자	설명
실행 파일	exe, com	CPU가 작업하는 프로세스 확장자
소스코드 파일	c, cpp, java, py	다양한 소스코드의 확장자
문서 파일	txt, doc, hwp, pdf, ps 등	문서 데이터 파일 확장자
동영상 파일	avi, mp4, mkv, mov	동영상 데이터 파일 확장자
음악 파일	wav, mp3, ogg, flac, aac	음악 데이터 파일 확장자
이미지 파일	bmp, gif, jpg, png, tiff	이미지 데이터 파일 확장자
압축 파일	rar, zip, arc, al	원래 크기보다 줄여서 저장한 파일 확장자

■ 실행 파일과 데이터 파일

- 실행 파일 : 운영체제가 메모리로 가져와 CPU를 사용하여 작업하는 파일
(사용자 요청으로 프로세스가 되는 파일)
- 데이터 파일 : 프로세스나 응용 프로그램이 사용하는 데이터를 모아 놓은 파일

■ 연결 프로그램

- 데이터 파일을 더블클릭하면 실행되는 응용 프로그램

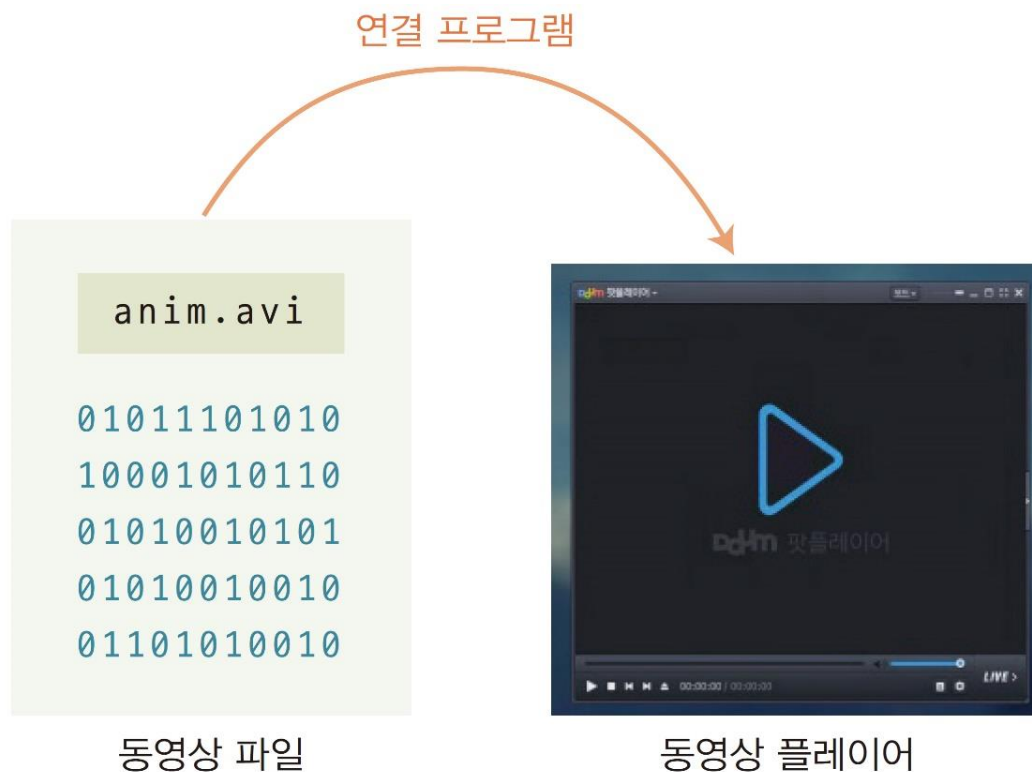


그림 6-27 동영상 파일 실행

■ 연결 프로그램 변경

- 변경하는 파일에서 마우스 오른쪽 버튼 → [속성] 메뉴 선택 → ‘변경’ 버튼
→ 프로그램 선택



그림 6-28 연결 프로그램 변경

■ 파일과 디렉터리

- 관련 있는 파일을 하나로 모아 놓은 곳
- 여러 층으로 구성 가능(디렉터리 밑에는 또 다른 디렉터리를 만들 수 있음)
- 최상위 디렉터를 루트 디렉터리(root directory)라고 함



그림 6-29 파일과 디렉터리

■ 디렉터리 계층 구조

- 역슬래시(\)는 루트 디렉터리를 의미
- 한글 자판에서 역슬래시는 ₩(원화 표시)로 대체
- 디렉터리 헤더에는 디렉터리 이름, 만든 시간, 접근 권한 등 정보가 기록되어 있음

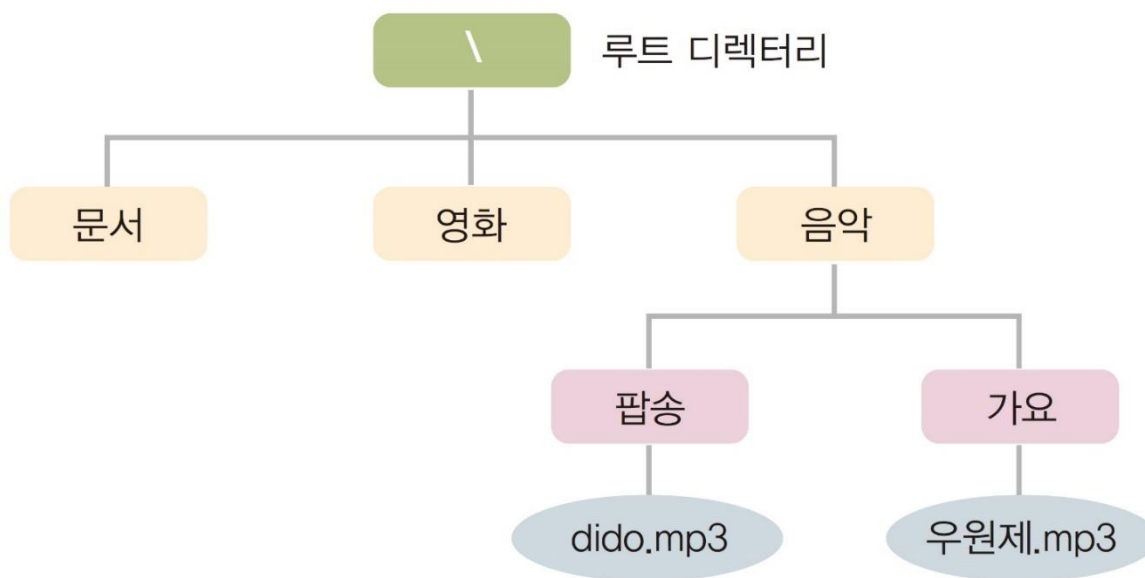
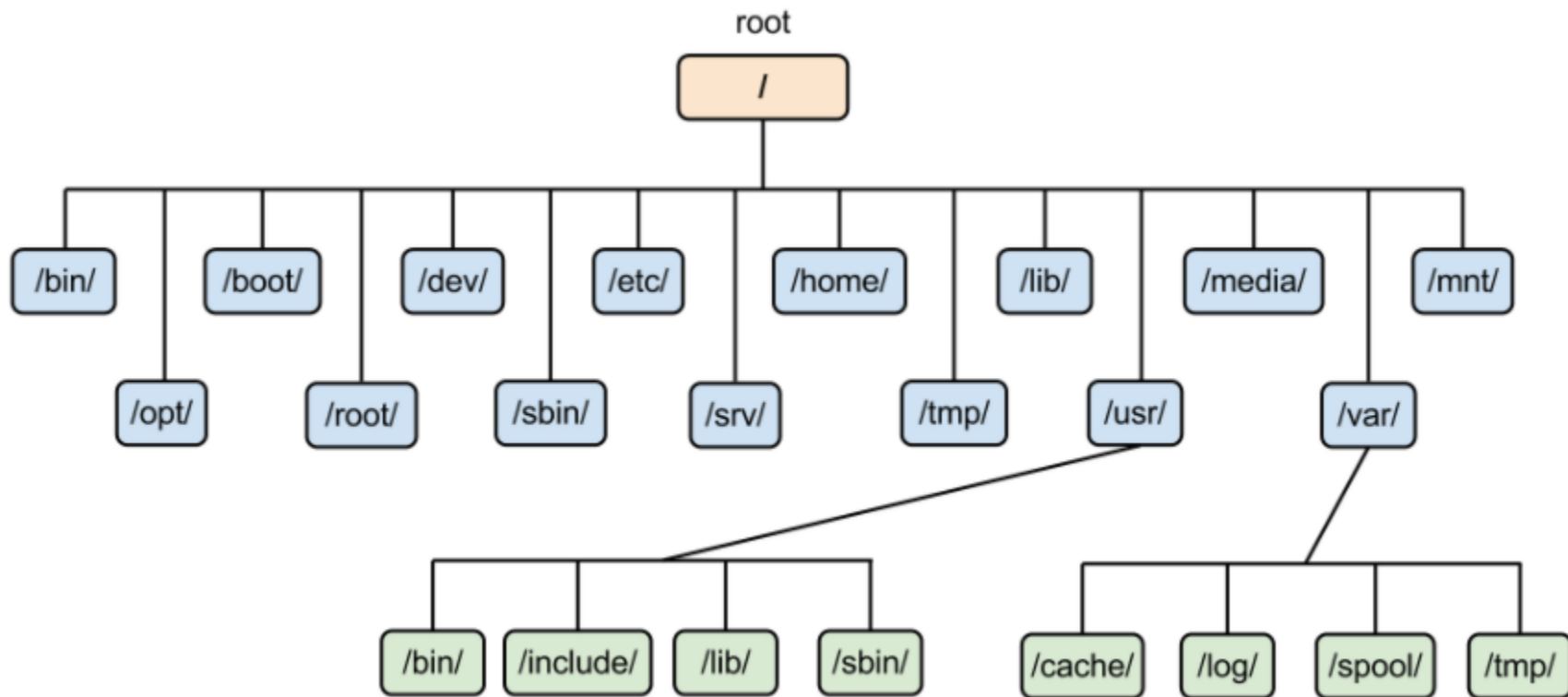


그림 6-30 디렉터리 계층 구조

■ 리눅스의 디렉터리 구조와 역할



■ 리눅스의 디렉터리 구조와 역할

1. **/bin (Binary)** 일반 사용자 및 관리자가 사용하는 명령어의 실행 파일이 배치되어 있는 디렉터리입니다.

(/bin은 특히 시스템과 관련된 중요도가 높은 명령어를 포함하고 있습니다)

2. **/dev (devices)** 디바이스 파일이 배치되어 있는 디렉토리입니다.

디바이스 파일이란 디스크나 키보드 등 하드웨어를 다루기 위한 특수 파일입니다.

3. **/etc** 리눅스에서 돌아가는 다양한 애플리케이션의 설정 파일이 **/etc** 아래에 배치됩니다.

애플리케이션 뿐만 아니라 리눅스 자체의 설정 파일도 이곳에 있습니다.

따라서 리눅스를 운영하고 관리할 때 무척 중요한 디렉토리입니다.

4. **/home** 사용자별로 할당되는 홈 디렉토리가 배치되는 디렉토리입니다.

홈 디렉토리란 사용자별로 할당되는 개인용 디렉토리를 말합니다. 사용자 이름이 디렉토리 이름으로 사용됩니다.

예를 들어 사용자 이름이 **idk**라면 사용자의 홈 디렉토리는 **/home/idk**가 됩니다.

5. **/sbin (System Binary)** **/bin**과 비슷하게 실행 파일을 포함하는 디렉토리입니다.

그런데 이 디렉토리에는 관리자용 명령어가 포함되어 있습니다.

* **/bin**과의 차이점은 **/bin**은 모든 사용자가 사용 가능한 기본 명령어를 담고 있고,

/sbin은 시스템관리자와 같은 특정 권한을 가진 명령어를 가집니다.

6. **/tmp** 임시 파일이 들어있는 디렉토리입니다. 애플리케이션 실행 중 임시로 작업 결과를 파일로 보존할 때 보통 이

디렉토리에 저장합니다.

7. **/usr (Unix System Resources)** 설치한 애플리케이션의 실행 파일, 문서, 라이브러리 등이 이 디렉토리에 포함됩니다.

/usr 아래에는 **bin**, **sbin**, **etc** 등이 있어 루트 디렉토리와 구조가 비슷합니다.

8. **/var (variable)** 변화하는 데이터를 저장하기 위한 디렉토리입니다.

애플리케이션 실행 중에 만들어진 데이터나 로그, 메일 등이 이곳에 저장됩니다.

9. **/opt (optional)** 시스템의 기본 설치 위치가 아닌 선택적으로 설치된 소프트웨어가 위치하는 공간

■ 파일 테이블

- 파일이 저장된 파일 이름, 위치 정보 등이 저장
- 모든 운영체제는 고유의 파일 테이블을 가짐
- 윈도우는 FAT(File Allocation Table)나 NTFS, 유닉스는 i-node 같은 파일 시스템 운영

파일 A	1, 3, 9
파일 B	4, 2
파일 C	13
파일 D	15, 12
파일 E	23, 7

파일 테이블

	0	1	2	3	4	5	6	7	8	9	블록 번호
0		A	B	A	B			E		A	
1			D	C		D					
2				E							
3											
4											
5											

저장 장치

그림 6-31 파일 테이블

■ 포매팅(formatting)

- 디스크에 파일 시스템을 탑재하고 디스크 표면을 초기화하여 사용할 수 있는 형태로 만드는 작업
 - 빠른 포매팅 : 데이터는 그대로 둔 채 파일 테이블을 초기화하는 방식
 - 느린 포매팅 : 파일 시스템을 초기화할 뿐 아니라 저장 장치의 모든 데이터를 0으로 만들어 버림

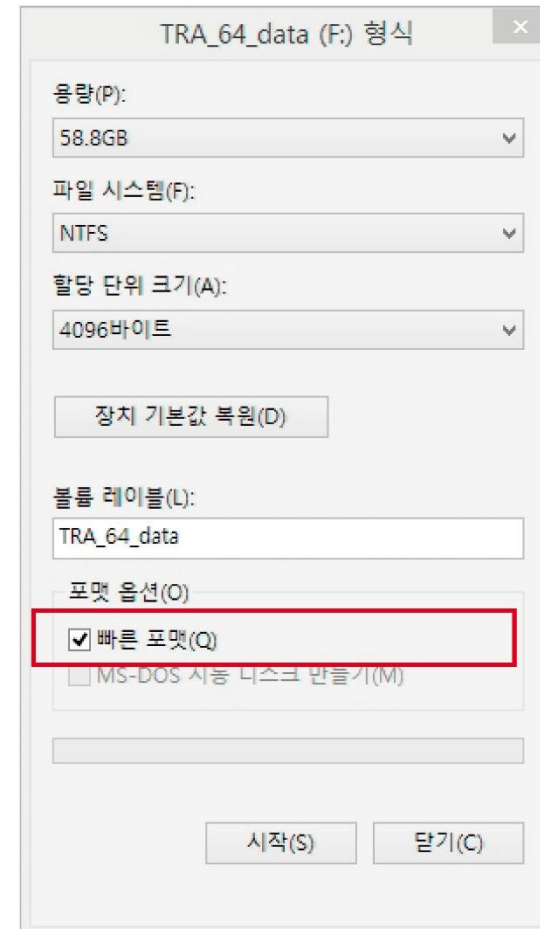


그림 6-32 저장 장치 포맷 화면

■ 섹터

- 하드디스크의 물리적인 구조상 가장 작은 저장 단위
- 섹터마다 주소를 부여하면 너무 많은 양의 주소가 필요하기 때문에 파일 관리자는 여러 섹터를 묶어 하나의 블록으로 만들고 블록 하나에 주소 하나를 배정

■ 블록

- 저장 장치에서 사용하는 가장 작은 단위
- 한 블록에 주소 하나를 할당
- 데이터는 운영체제와 저장 장치 간에 블록 단위로 전송

■ 블록 크기

- 블록 크기는 시스템마다 다름
- 포맷할 때 시스템이 정한 기본 블록 크기를 사용 또는 4,096B~64KB의 다양한 블록 크기를 직접 지정
- 블록 크기를 작게 설정하면 저장 장치를 효율적으로 쓸 수 있지만, 파일이 여러 블록으로 나뉘어 파일 입출력 속도가 느려짐

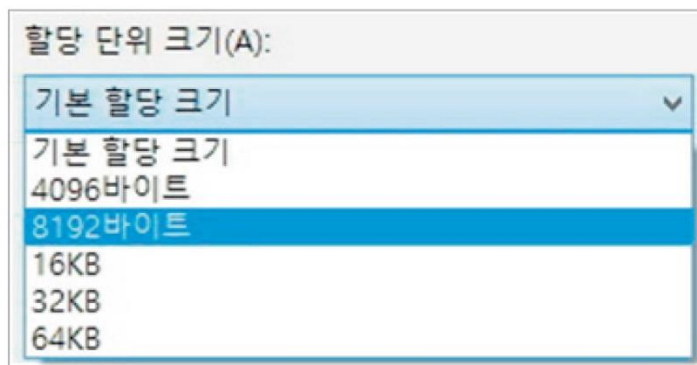
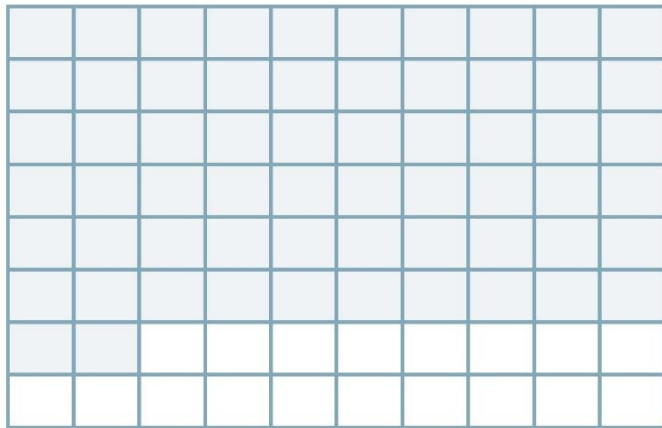


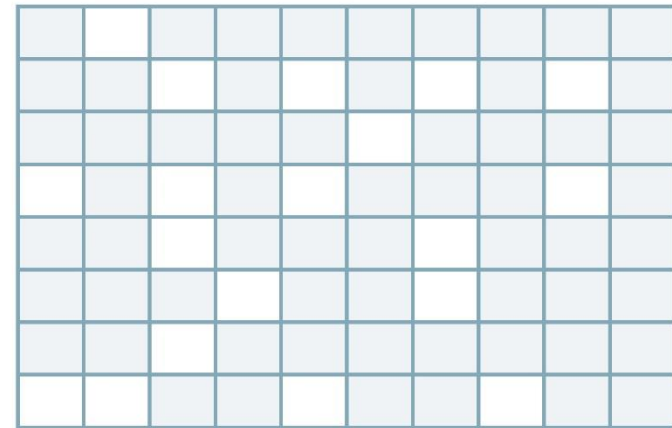
그림 6-33 윈도우의 블록 크기 설정

■ 조각화(단편화)

- 하드디스크를 처음 사용할 때는 데이터가 앞부터 차곡차곡 쌓이지만, 사용하다 보면 파일이 삭제되면서 중간중간 빈 공간이 생김
- 하드디스크에 조각이 많이 생기면 큰 파일을 여러 조각으로 나누어 저장
→ 성능 저하



초기 상태



조각 난 상태

그림 6-34 초기 상태와 조각 난 상태

■ 조각모음(defragmentation)

- 주기적으로 조각모음 실행
- 시간이 오래 걸리는 작업이므로 작업이 없는 특정 시간에 조각모음을 실행
- USB 메모리, SSD 등 반도체를 사용하는 저장 장치는 조각모음을 하지 않음
- 조각모음을 통해 특정 위치의 메모리만 계속 사용하면 수명 단축

■ FAT 파일 시스템 구조

- 왼쪽 테이블은 파일 정보와 함께 파일의 시작 블록 정보를 가짐
- 파일 B : 2 → 4 → 12 → 8(null)번 블록
- 파일 C : 0 → 3 → 10 → 6 → 9 → 7 → 11 → 14(null)번 블록

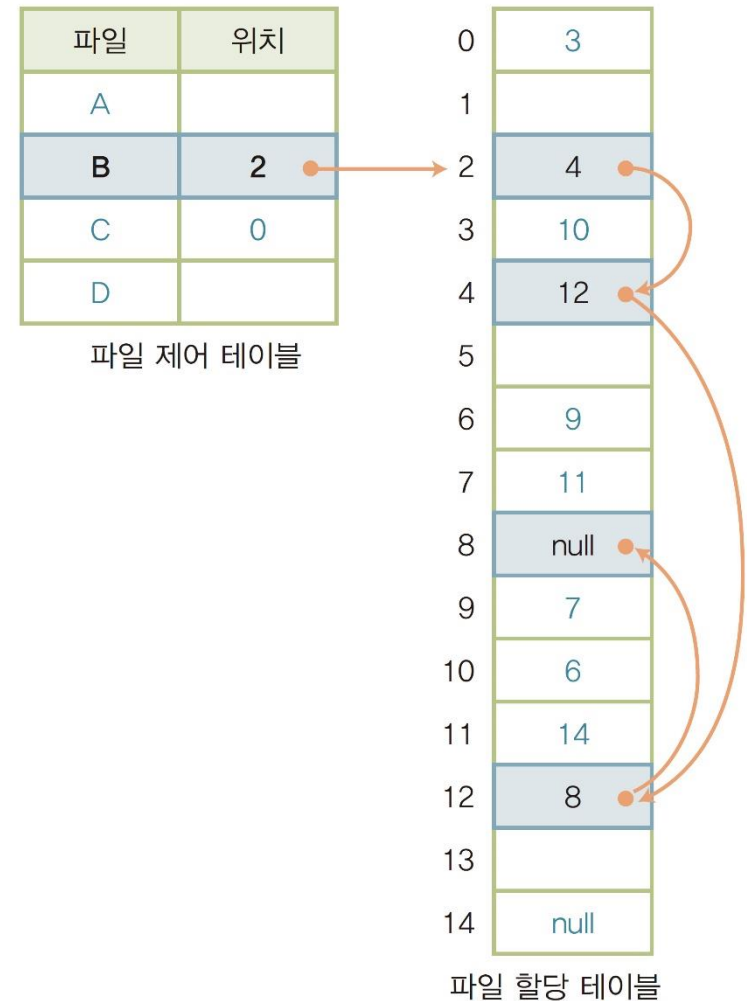


그림 6-35 FAT 파일 시스템 구조

Thank you!