# 운영체제: 강의노트 03

A. Silberschatz, P.B. Galvin, G. Gagne
*Operating System Concepts*,
Sixth Edition, John Wiley & Sons, 2003.

## Part I. Overview

# 3 운영체제 구조

## 3.1 시스템 구성요소

### 3.1.1 프로세스 관리

- **프로세스**: 실행 중인 프로그램
- 프로세스는 그것의 임무를 수행하기 위해 CPU 시간, 메모리, 파일, 입출력 장치와 같은 자원이 필요하다.
- 자원의 할당은 프로세스의 시작과 동시에 이루어질 수 있으며, 필요할 때마다 이루어질 수 있다.
- 프로세스는 능동적 개체인 반면 프로그램은 수동적 개체이다.
- 한 프로그램은 그것이 실행되는 동안 여러 개의 프로세스를 생성할 수 있다.
- 프로세스는 시스템의 작업 단위가 된다.
- 프로세스는 크게 운영체제 프로세스와 사용자 프로세스로 구분된다.
- 프로세스 관리와 관련된 운영체제의 임무
  - 사용자와 시스템 프로세스의 생성과 제거
  - 프로세스의 중지와 재개
  - 프로세스 동기화
  - 프로세스 간 통신
  - 교착상태(deadlock) 처리

### 3.1.2 주기억장치 관리

- 주기억장치 관리와 관련된 운영체제의 임무
  - 기억 공간의 사용 현황 관리
  - 적재할 프로세스 선택
  - 기억 공간의 할당과 회수

### 3.1.3 파일 관리

- 파일: 생성자에 의해 정의된 관련 정보의 모음
- 운영체제는 다양한 저장 매체에 대한 균일한 논리적 뷰를 제공
  - 저장 장치의 물리적 특성을 논리 저장 단위인 파일로 추상화해준다.
- 파일 관리와 관련된 운영체제의 임무
  - 파일의 생성과 삭제
  - 디렉토리(폴더)의 생성과 삭제
  - 파일과 디렉토리를 조작하기 위한 기초 연산 제공
  - 보조기억장치와 파일 간에 매핑
  - 백업

### 3.1.4 입출력 관리

- 운영체제의 주 임무 중 하나는 다양한 하드웨어 장치의 특성을 사용자에게 숨기는 것이다.
- 보통 입출력 하위시스템(I/O subsystem)을 사용하여 이 기능을 제공한다.
- 입출력 하위시스템의 구성요소
  - 버퍼링, 캐싱, 스풀링(spooling)[1]과 같은 기억장치 관리 구성요소
  - 일반적인 장치 구동기(device driver)[2] 인턴페이스
  - 특정 하드웨어를 위한 장치 구동기

### 3.1.5 보조기억장치 관리

- 주기억장치는 휘발성이며 그 용량이 제한적이므로 필요하다.
- 보조기억장치 관리와 관련된 운영체제의 임무
  - 빈 공간 관리
  - 저장 공간 할당
  - 디스크 스케줄링: 디스크와 관련된 요청의 실행 순서를 결정하는 문제

### 3.1.6 네트워킹

- 운영체제는 네트워크 접근을 파일 접근과 같은 형태로 일반화해준다.
- 운영체제는 각종 통신 프로토콜을 지원해야 하며, 각종 네트워크 장치 구동기를 위한 인터페이스를 제공해 주어야 한다.

[1] 직접 입출력을 하지 않고 스풀링 디렉토리에 입출력할 데이터를 저장하면 그 입출력을 담당하는 daemon이 스풀링 디렉토리에 있는 데이터를 입출력한다. 스풀링은 CPU 작업과 입출력을 병행할 수 있게 해주며, 같은 입출력 요청의 큐를 만들 수 있도록 해준다.

[2] 장치를 제어하는 프로그램으로서 각 장치는 자신만의 고유한 명령어 집합을 가진다. 프로그램은 일반 입출력 명령을 이용하여 운영체제를 통해 장치를 사용하며, 이 때 장치 구동기는 일반 명령을 자신의 고유의 명령으로 바꾸어 실행해준다.

### 3.1.7 보호 시스템

- 다중 사용자, 다중 프로세스 환경에서 운영체제는 각 프로세스를 다른 프로세스로부터 보호해 주어야 한다.
- 보호: 컴퓨터 시스템에서 정의한 자원에 대한 프로세스와 사용자의 접근을 제어하기 위한 메커니즘

### 3.1.8 명령 해석기 시스템

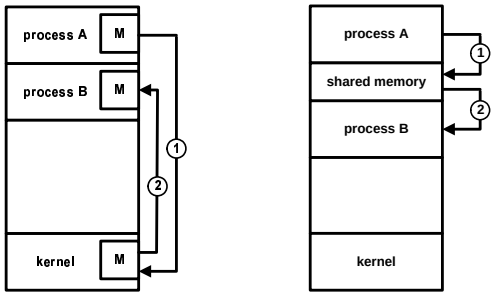
- 명령 해석기: 사용자와 운영체제 간에 인터페이스로 사용자로부터 명령을 받아 실행해준다. 명령 해석기는 커널에 포함되지 않는다.
  - 유닉스에서는 보통 쉘(shell)이라 한다.
- 초창기에는 키보드를 이용한 명령어 기반 방식이었지만 현재는 대부분 마우스 기반 윈도우 방식이다.

## 3.2 운영체제 서비스

- 프로그래머의 편리성을 위해 제공되는 서비스
  - 프로그램 실행
  - 입출력 수행
  - 파일시스템 조작
  - 통신: 공유 메모리, 메시지 전달
  - 오류 검출
- 시스템의 효율적 운영을 위해 제공되는 서비스
  - 자원 할당
  - 회계(accounting)
  - 보호

## 3.3 시스템 호출

- 시스템 호출은 프로세스와 운영체제 간에 인터페이스를 제공해준다.
- 많은 고급 프로그래밍 언어(예, C, C++, Java)는 사용자가 직접 시스템 호출을 할 수 있도록 지원한다. 그러나 내부 복잡한 과정은 사용자는 알 필요가 없도록 숨겨준다.
- 시스템 호출할 때 운영체제에 파일 이름과 같은 추가 정보를 전달해야 한다.
- 운영체제에 파라미터를 전달하는 방법
  - 레지스터를 통해 직접 전달
  - 메모리에 저장한 후에 그것의 시작주소와 크기를 레지스터를 통해 전달
  - 스택 구조를 이용
- 시스템 호출의 종류
  - 프로세스 제어



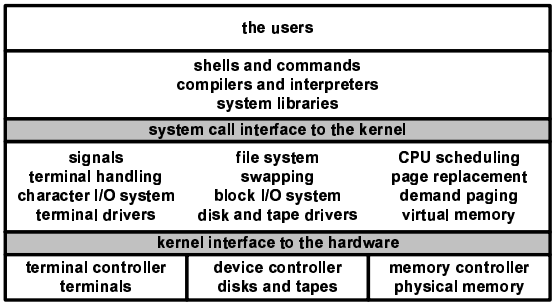

(a) 메시지 전달 모델 (b) 공유 메모리 모델

<그림 3.1> 통신 모델

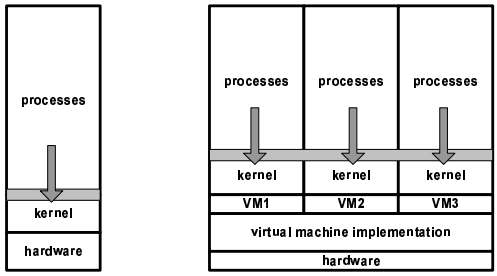
    - 프로세스의 정상 종료(end)와 비정상 종료(abort)
    - 프로세스 적재와 수행
    - 프로세스의 생성과 강제 종료(terminate)
    - 프로세스 속성 획득 및 설정
    - 시간 대기
    - 사건 대기, 사건 발생 알림
    - 메모리 할당과 해제
  - 파일 관리
    - 파일 생성과 삭제
    - 파일 열기와 닫기
    - 읽기, 쓰기, 재배치(reposition)
    - 파일 속성 획득 및 설정
  - 장치 관리
    - 장치 사용 요청, 장치 해제
    - 읽기, 쓰기, 재배치: 파일과 장치는 매우 유사하므로 보통 장치을 가상 파일로 추상화하여 사용한다.
    - 장치 속성 획득 및 설정
    - 장치의 논리적 부착과 제거
  - 정보 유지보수
    - 시간과 날짜 요청
    - 시스템 정보 획득 및 설정
    - 프로세스, 파일, 장치 속성 획득 및 설정
  - 통신(메지지 전달 모델과 공유 메모리 모델): 그림 3.1 참조
    - 통신 연결 생성 및 제거
    - 데이터 전송 및 수신
    - 상태 정보 전달
    - 원격 장치 부착 및 제거

## 3.4 시스템 프로그램

- 시스템 프로그램은 프로그램 개발과 실행을 위한 편리한 환경을 제공해준다.
- 시스템 프로그램의 종류




<그림 3.2> 유닉스 시스템 구조




(a) 비 가상 기계 모델 (b) 가상 기계 모델

<그림 3.3> 시스템 모델

- 파일 관리: 예) 탐색기
- 상태 정보: 예) 등록정보 기능
- 파일 편집: 예) 문서 편집기
- 프로그래밍 언어 지원: 예) 컴파일러, 어셈블러, 인터프리터
- 프로그램 적재와 실행: 예) 디버거
- 통신

## 3.5 시스템 구조

### 3.5.1 단순 구조

- 대부분의 상용 운영체제는 잘 정의된 구조를 가지고 있지 않다. 이들은 처음에는 작고 단순한 시스템에서 출발하여 그것의 인기를 바탕으로 점점 고급 기능이 추가되었다.
- 이들 시스템이 단순 구조를 가지게 주된 이유 중 하나는 개발 당시 제공된 하드웨어의 기능적 제한 때문이다.

### 3.5.2 계층적 접근

- 계층적 접근 방식: 운영체제를 여러 계층으로 나누는 방식으로, 이 방식에서 상위 계층은 하위 계층의 인터페이스를 이용한다. 가장 하위 계층은 하드웨어이고, 가장 상위 계층은 사용자 인터페이스가 된다.
- 계층적 접근 방식의 장단점
  - 장점: 모듈화을 통한 하위 계층의 추상화
    - 인턴페이스만 그대로 유지하면 하위 계층은 상위 계층과 무관하게 수정할 수 있다. 즉, 하위 계층은 하위 계층의 복잡성을 상위 계층에게 숨겨준다.
  - 단점
    - 계층 구분의 어려움
    - 효율성 문제: 각 계층 간에 인터페이스를 통한 작업 수행으로 많은 오버헤드가 존재할 수 있음

### 3.5.3 마이크로 커널

- 마이크로 커널: 커널에 꼭 필요한 기능만 가지도록 구성된 커널
- 대표적인 마이크로 커널은 Carnegie Mellon 대학에서 개발한 Mach이다.
- 마이크로 커널의 장점: 확장의 용이성, 높은 안전성(많은 것이 사용자 모드에서 동작), 높은 신뢰성, 높은 이식성(portability)

## 3.6 가상 기계

- 프로세스가 마치 자신만의 프로세서와 메모리를 가지고 있는 것처럼 생각할 수 있도록 운영체제를 설계할 수 있다. 이런 접근 방식을 가상 기계(VM, Virtual Machine)라 한다.
- 가상 기계 방식에서 모든 프로세스는 컴퓨터의 복사본이 제공된다.
- 가상 기계를 사용하면 입출력의 실행 속도는 실제 기계에서 이루어지는 것보다 빠를 수도 있고(스풀링) 느릴 수도 있다(해석 방법).
- CPU는 여러 가상 기계 사이에 다중 프로그래밍되므로 가상 기계의 속도는 더 느려질 수 있다.

### 3.6.1 구현

- 보통 기계는 사용자 모드와 모니터 모드 두 가지 모드를 제공한다.
- 가상 기계를 구현하는 소프트웨어는 운영체제이므로 모니터 모드에서 동작 가능하지만 실제 가상 기계는 사용자 모드에서 동작해야 한다.
- 가상 기계를 사용하는 프로세스 측면에서 보면 가상 기계 역시 가상 기계 사용자 모드와 가상 기계 모니터 모드 두 가지 모드를 제공해야 한다. 그런데 이 두 모드는 모두 실제로는 사용자 모드에서 동작해야 한다.

### 3.6.2 장점

- 각 가상 기계는 서로 독립적이므로 시스템 자원 보호 측면에서 유리하다. 그러나 자원 공유가 어렵다.

- 시스템 개발의 용이성: 시스템 프로그래머는 자신만의 가상 기계에서 개발하므로 실제 시스템에 영향을 주지 않고 개발이 가능하다.
- 호환성 문제 해결하는 수단으로 사용: 가상 기계를 만들어야 할 실제 기계가 복잡하면 할 수록 정학한 가상 기계를 만드는 것이 어려워지며, 이것은 가상 기계의 수행 속도에 나쁜 영향을 준다.

#### 3.6.3 자바

- 자바: Sun사에서 개발한 객체지향 언어
- 자바 프로그램은 자바 가상 기계(JVM, Java Virtual Machine)를 사용하며, 이 때문에 자바 프로그램은 자바 가상 기계가 있는 어떤 컴퓨터에서도 실행된다.
- 자바 컴파일러는 각 자바 클래스마다 컴퓨터 구조와 독립적인 바이트코드(bytecode)을 생성하며, 이 코드는 자바 가상 기계에서 실행된다.
- 자바 가상 기계는 클래스 로더(class loader), 클래스 확인자(class verifier) 자바 인터프리터(java interpreter)로 구성되어 있다.
- 자바 가상 기계는 폐영역 회수(garbage collection) 방법으로 메모리를 관리한다.
- 자바 인터프리터는 바이트코드를 그때 그때 해석하여 실행하는 소프트웨어 모듈이거나 바이트코드를 모두 목적 컴퓨터의 기계코드로 바꾸어 실행하는 JIT(Just-In-Time) 컴파일러일 수 있다.

### 3.7 시스템 설계 및 구현

#### 3.7.1 설계 목표

- 최상위 목표: 사용하는 하드웨어, 시스템의 종류에 의해 결정
- 하위 레벨의 목표
  - 사용자 목표: 사용의 편리성, 이해의 용이성, 신뢰성, 처리 속도의 빠름
  - 시스템 목표: 유연성(메커니즘과 정책의 분리), 신뢰성, 효율성
    - 범용적 메커니즘을 사용하여 정책이 바뀌어도 메커니즘까지 바꿀 필요가 없도록 하는 것이 바람직하다.

#### 3.7.2 구현

- 초창기에는 어셈블리 언어로 운영체제를 구현하였지만 현재는 대부분 C나 C++와 같은 고급 프로그래밍 언어로 구현한다.
- 고급 프로그래밍 언어로 구현할 때의 장단점
  - 장점: 구현의 용이성, 디버깅의 용이성, 코드 이해의 용이성, 높은 이식성
  - 단점: 속도 저하, 저장 공간 증가
- 단점에 대한 반론
  - 올바른 데이터 구조와 알리고즘을 사용하는 것이 시스템 성능에 더 큰 도움이 된다.
  - 시스템 성능에 큰 영향을 주는 부분은 극소수이다. 이런 것을 찾아 이 부분만 어셈블리 언어로 구현할 수 있다.

### 3.8 시스템 설치

- 시스템 설치: 각 특정 사이트에 동작하도록 구성하는 과정
- 시스템 설치할 때 결정해야 하는 정보
  - CPU의 종류, CPU의 수
  - 주기억장치의 크기
  - 부착된 장치
  - 운영체제 옵션 결정: CPU 스케줄링 방법, 버퍼의 크기 등
- 설치 방법
  - 방법 1. 설치자가 위 정보를 알아낸 후에 필요한 소스 코드를 수정하고 컴파일한 후에 설치
  - 방법 2. 필요한 모든 코드를 가지고 있는 시스템을 이용하여 설치한다. 위 정보는 설치하면서 동적으로 결정되거나 선택된다. 방법 1보다 이 방법의 운영체제의 크기가 크다.
- 운영체제가 설치된 후에 컴퓨터를 다시 시작하면 컴퓨터는 커널을 주기억장치에 적재하고 실행한다. 이 과정을 부팅(booting)이란 한다. 이것을 해주는 프로그램을 부트스트랩 프로그램 또는 부트스트랩 로더(bootstrap loader)라 한다.