01. 서론

- 운영체제는 컴퓨터 하드웨어를 관리하는 프로그램이다.
- 운영체제는 응용 프로그램을 위한 기반을 제공하며 컴퓨터 사용자와 컴퓨터 하드웨어 사이에서 중재자 역할을 실행한다.
- 운영체제는 편리성과 효율성을 제공한다.
  - 편리성 : 사용자에게 편리함을 제공 → 소형, 개인용 컴퓨터의 경우 더욱 중요하다
  - 효율성 : 컴퓨터 시스템의 효율적인 운영 → 다수의 사용자가 공유하는 대형 시스템에서 특히 중요하다
- 효율성과 편리성은 때때로 서로 상반된다.
  → 어떤 운영체제는 일반인들이 사용하기 편리하도록 설계되고, 일부는 효율성에 주안을 두고, 일부는 이들의 조합으로 설계된다.

▼ 목차

1. 운영체제가 할 일

> What Operating Systems Do

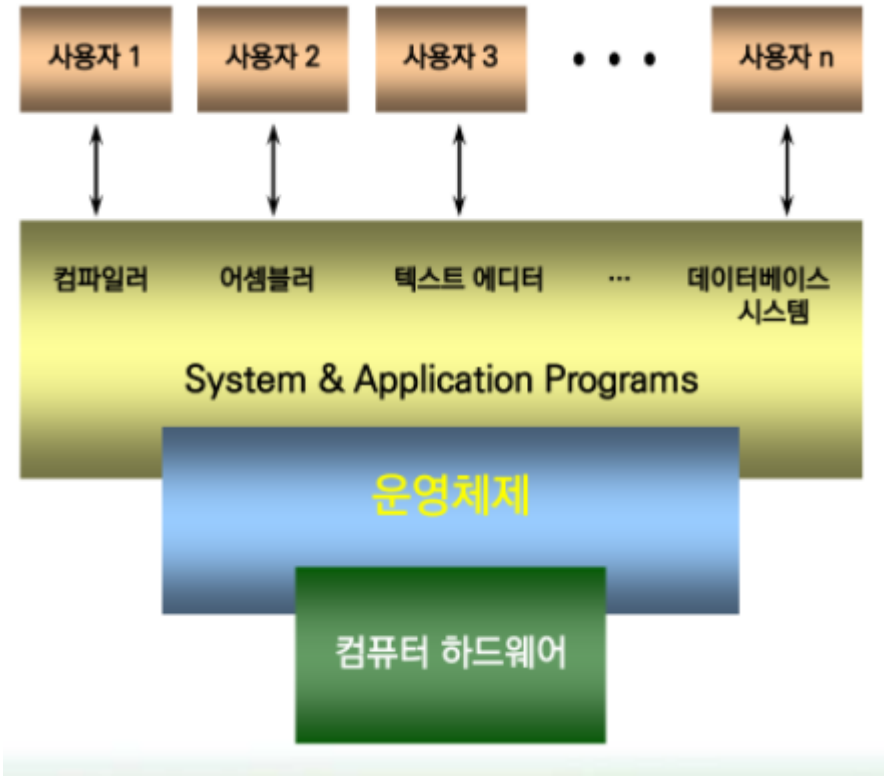
전체 컴퓨터 시스템에서 운영체제가 수행하는 역할에 대해 살펴본다.

컴퓨터 시스템 구성 요소

> 💡 컴퓨터 시스템은 하드웨어, 운영체제, 응용 프로그램, 사용자의 네 가지 구성요소로 구분할 수 있다.

- 하드웨어(Hardware) : 기본 계산용 자원을 제공한다.
  Ex. CPU, 메모리, 입출력 장치
- 응용 프로그램(Application programs) : 사용자의 계산 문제를 해결하기 위해 이들 자원이 어떻게 사용될지 정의한다.
  Ex. 워드 프로세서, 스프레드시트, 웹 브라우저
- 사용자(User)
  Ex. 사람, 기계, 다른 컴퓨터들

▼ <Figure> 컴퓨터 시스템 구성



- 운영체제는

  → 다양한 사용자들을 위해 다양한 응용 프로그램 간의 하드웨어 사용을 조정한다.

→ 컴퓨터 시스템이 동작할 때 자원을 적절하게 사용할 수 있는 방법을 제공한다.

→ 다른 프로그램이 유용한 작업을 할 수 있는 환경을 제공한다.

- 운영체제의 역할을 이해하기 위한 2가지 관점
  - 사용자 관점(User view) : 운영체제를 사용하는 사용자 입장에서 바라보는 운영체제의 역할
  - 시스템 관점(System view) : 운영체제가 운영되는 컴퓨터 시스템 입장에서 바라보는 운영체제의 역할

사용자 관점

컴퓨터에 대한 사용자의 관점은 사용하는 인터페이스에 따라 달라진다.

- PC
  : 한 사용자가 자원을 독점하도록 설계되었으며, 목표는 사용자가 수행하는 작업을 최대화하는 것
  → 사용의 편의성, 성능 또한 한 사용자가 사용하기에 적합하도록 최적화
- mainframe or minicomputer
  : 사용자들은 자원(가용한 CPU 시간, 메모리, 입출력 장치 등)을 공유하며 정보를 교환할 수 있다.
  → 자원 이용의 극대화, 모든 가용 자원의 효율적인 사용, 각 개인은 정당한 자신의 몫만 사용
- workstations
  : 사용자들은 자신의 전용 자원을 갖지만 네트워킹, 서버-파일, 계산, 프린트 서버를 공유한다.
  → 개인의 사용 편이성과 자원 이용 간에 적절한 조화를 이루도록 설계
- Handheld computers
  : 개인 사용자들을 위한 독립형 장치들
  : 이메일이나 웹 브라우징을 위해 컴퓨터를 사용하는 사람들에게 데스트콥 및 랩톱 컴퓨터를 대체한다.
- embedded computers
  : 가전 제품이나 자동차 내의 내장형 컴퓨터
  : 컴퓨터와 운영체제는 사용자의 개입 없이 동작하도록 설계되어야 한다.

→ 사용자 관점에서는 대부분 사용의 용이성을 위해 설계되고 성능에 약간 신경쓰고, 자원의 이용에는 전혀 신경을 쓰지 않는다.

시스템 관점

컴퓨터의 관점에서 운영체제는 하드웨어와 가장 밀접하게 연관된 프로그램이다.

1. 자원 할당자(resource allocator)로서의 운영체제
    - 컴퓨터 시스템의 자원(resource)들의 관리자로서 동작한다.
        - 자원 : CPU 시간, 메모리 공간, 파일 저장 공간, 입출력 장치 등
    - 특정 프로그램과 사용자에게 필요한 자원을 할당한다.
      → 컴퓨터 시스템을 효율적이고 공정하게 운영할 수 있도록 어느 요청에 자원을 할당할지 결정해야 한다.

2. 제어 프로그램(control program)으로서의 운영체제
    - 오류와 컴퓨터의 부적절한 사용을 방지하기 위해 사용자 프로그램의 실행을 제어한다.
    - 사용자 프로그램이 실행하는 입출력 장치의 제어와 작동에 깊이 관여한다.

운영체제의 정의

일반적으로 운영체제에 대한 완벽한 정의는 없다. ← 운영체제가 굉장히 포괄적인 범위를 띄고 있기 때문

- 운영체제는 다양한 프로그램들이 필요로 하는 입출력 장치의 제어와 같은 공통적인 연산과 자원을 제어하고 할당하는 공통 기능을 하나의 소프트웨어로 통합한 것이다. → 운영체제의 핵심 기능에 대한 정의

- 운영체제의 구성요소는 무엇인가?
    - 어느 부분이 운영체제에 속하고 어느 부분이 속하지 않는가?
    - 사용자가 '운영체제'를 주문했을 때 판매업자가 배송하는 모든 것?
    - 커널(Kernel) = 운영체제?

    - 커널 : 컴퓨터 상에서 항상 실행되는 프로그램 ← OS의 핵심 기능만으로 구성된 소프트웨어
- 휴대용 장치의 운영체제를 보면 운영체제를 구성하는 기능의 수가 증가하고 있다.
    - 커널 외의 미들웨어(응용 개발자에게 추가 서비스를 제공하는 소프트웨어 프레임워크 집합)을 포함한다.

운영체제는 왜 공부하는가?

> 💡 거의 모든 코드가 운영체제 위에서 실행되므로 운영체제 작동방식에 대한 지식은 적절하고 효율적, 효과적이며 안전한 프로그래밍에 중요하기 때문이다. 그래서 운영체제의 기본 지식, 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.

2. 컴퓨터 시스템 구성

> Computer System Organization

컴퓨터 시스템에 대한 일반적인 지식을 학습한다.

- 운영체제는 컴퓨터 시스템의 정확한 동작을 보장해야 한다.
- 사용자 프로그램이 시스템의 적절한 동작을 방해하지 않도록 하기 위해 하드웨어는 정확한 실행을 보장할 수 있는 적합한 기법을 제공해야 한다.

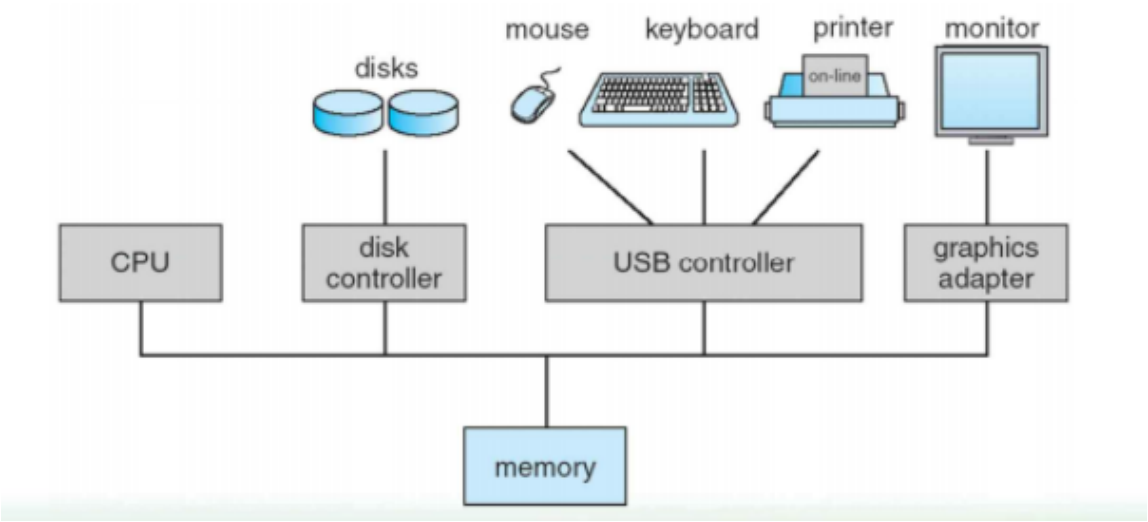
컴퓨터 시스템의 연산

> 💡 현대의 범용 컴퓨터 시스템은 하나 이상의 CPU와 다수의 장치 제어들로 구성되며 이들은 공용 버스로 연결된다.

- CPU와 장치제어기들은 이 버스를 통하여 공유 메모리에 접근할 수 있다.

- 각 장치 제어기는 특정 장치(디스크 드라이브, 오디오 장치, 비디오 디스플레이)를 관리한다.
- CPU와 장치 제어기는 메모리 사이클을 얻기 위해 경쟁하면서 병렬 실행될 수 있다.

▼ <Figure> 컴퓨터 시스템



컴퓨터의 구동

초기 프로그램 실행 : 부트스트랩 프로그램 (bootstrap program)

- 펌웨어(firmware)
  - 컴퓨터 내의 ROM이나 EEPROM에 저장된다.
- 시스템의 모든 상태를 초기화
  - CPU 레지스터의 내용, 장치 제어기, 메모리 내용 등
- 운영체제를 적재하는 방법 및 실행을 시작하는 방법을 알아야 한다.
  → 운영체제의 커널을 찾아 메모리에 적재한다.

운영체제 시작

- 커널이 적재되고 실행되면 시스템과 사용자에게 서비스를 제공할 수 있다.
- 일부 서비스는 시스템 프로그램에 의해 제공된다. → 시스템 프로세스 또는 시스템 데몬
  - 부트 시에 메모리에 적재되어 커널이 실행되는 동안 계속 실행된다.

- UNIX의 경우, 첫 번째 프로세스 init이 실행되고 이 프로세스가 다른 많은 데몬을 시작한다.

→ 시스템이 완전히 부트된 상태가 되면, 다음 사건이(event)이 발생하기를 기다린다.

인터럽트

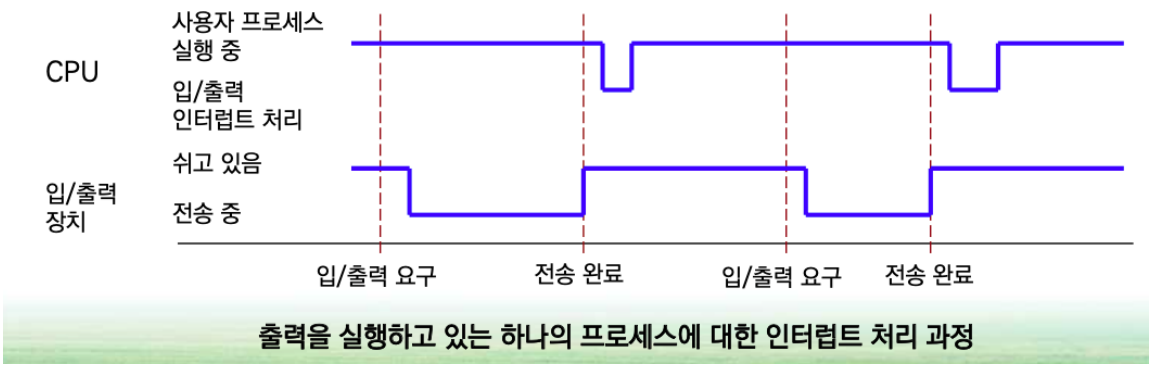
> 💡 운영체제는 어떠한 사건(event)이 발생하기를 기다린다.

- 사건의 발생 여부는 하드웨어 또는 소프트웨어로부터 발생한 인터럽트에 의해 전달 받는다.
  - 하드웨어 : 시스템 버스를 통해 CPU에 신호를 보내 인터럽트 발생
  - 소프트웨어 : 시스템 호출(system call)이라 불리는 특별한 연산을 실행하여 인터럽트 발생

인터럽트 처리

1. 인터럽트가 발생하면 CPU는 하던 일을 중단한다.
2. 인터럽트 서비스 루틴이 실행된다.
3. 인터럽트 서비스 루틴의 실행이 완료되면 CPU는 인터럽트 된 연산을 재개한다.

▼ <Figure> 인터럽트 처리 과정



출력을 실행하고 있는 하나의 프로세스에 대한 인터럽트 처리 과정

인터럽트는 적절한 서비스 루틴으로 제어를 전달한다.

- 간단한 방법 : polling ← 조건문으로 검사하는 방법과 유사 매우 비효율적이다.
  - 총괄 루틴을 호출하여 인터럽트에 관한 정보를 조사하고 이어 인터럽트 고유의 핸들러를 호출한다.
- 일반적인 방법 : vectored interrupt system ← 인덱스를 이용해 인터럽트 발생 효율적이다.
  - 인터럽트 루틴에 대한 포인터들의 테이블(인터럽트 벡터)를 이용한다.
  - 인터럽트 벡터 : 인터럽트 서비스 루틴의 주소 저장
    → 인터럽트가 요청되면, 인터럽트를 유발한 장치의 번호로 색인되어 인터럽트 서비스 루틴의 주소를 얻을 수 있다.

중단된 명령으로 복귀

- 인터럽트에 의해 중단된 명령의 주소를 저장해야 한다.
  - 고정된 위치 또는 장치 번호에 의해 색인되는 위치에 저장한다.
    → 최근의 구조들은 시스템 스택에서 복귀 주소를 저장한다.
- 인터럽트 루틴이 프로세서의 상태를 변경시킨 경우, 복귀하기 전에 상태를 복원해야 한다.
  - 저장되어 있는 복귀 주소를 프로그램 카운터에 적재하고 인터럽트에 의해 중단되었던 연산이 인터럽트가 발생되지 않았던 것처럼 다시 시작한다.

→ 이러한 인터럽트 처리 기법은 장치 컨트롤러가 서비스할 준비가 될 때와 같은 비동기 이벤트 CPU가 대응할 수 있게 한다.

▼ 하지만 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 긴급토로 대응할 수 있도록 다단계 인터럽트가 필요하다.

→ 이러한 기능은 CPU 및 인터럽트 컨트롤러 하드웨어에 의해 현재 제공된다.

저장장치 구조

> 💡 CPU는 명령어를 오로지 메모리에서만 가져올 수 있으므로 프로그램을 실행하려면 프로그램이 반드시 메모리에 있어야 한다.

- 범용 컴퓨터는 대부분의 프로그램을 주 메모리(RAM, Random-Access Memory)라 불리는 재기록 가능한 메모리에서 가져온다.
  - ROM(Read-Only Memory) : 갱신될 수 없으므로 정적인 프로그램만 저장, 부트스트랩 프로그램을 유지하는데 사용
    - EEPROM(Electrically Erasable Programmable Read-Only Memory) : 전기적으로 소거 가능한 프로그램 가능 읽기 전용 메모리. 변경할 수 있지만 자주 변경할 수는 없다. 또한 속도가 느리다.
- 모든 형태의 메모리는 바이트의 배열을 제공하고 각 바이트는 자신의 주소를 가지고 있다. 이를 통해 적재(load), 저장(store) 명령을 통해 상호작용을 할 수 있다.
  - 적재 : 메인 메모리로부터 CPU 내부의 레지스터로 한 바이트 또는 한 워드를 옮기는 것이다.
  - 저장 : 레지스터의 내용을 메인 메모리로 옮긴다.

이상적으로 프로그램과 데이터가 주 메모리에 영구히 존재하기를 원한다. → 적은 용량 & 휘발성

- 주 메모리는 모든 필요한 프로그램과 데이터를 영구히 저장하기에 용량이 너무 작다.
- 주 메모리는 전원이 공급되지 않으면 그 내용이 사라지게 되는 휘발성 저장장치이다.

주 메모리 확장용으로 보조 저장장치(Secondary storage)를 제공한다.

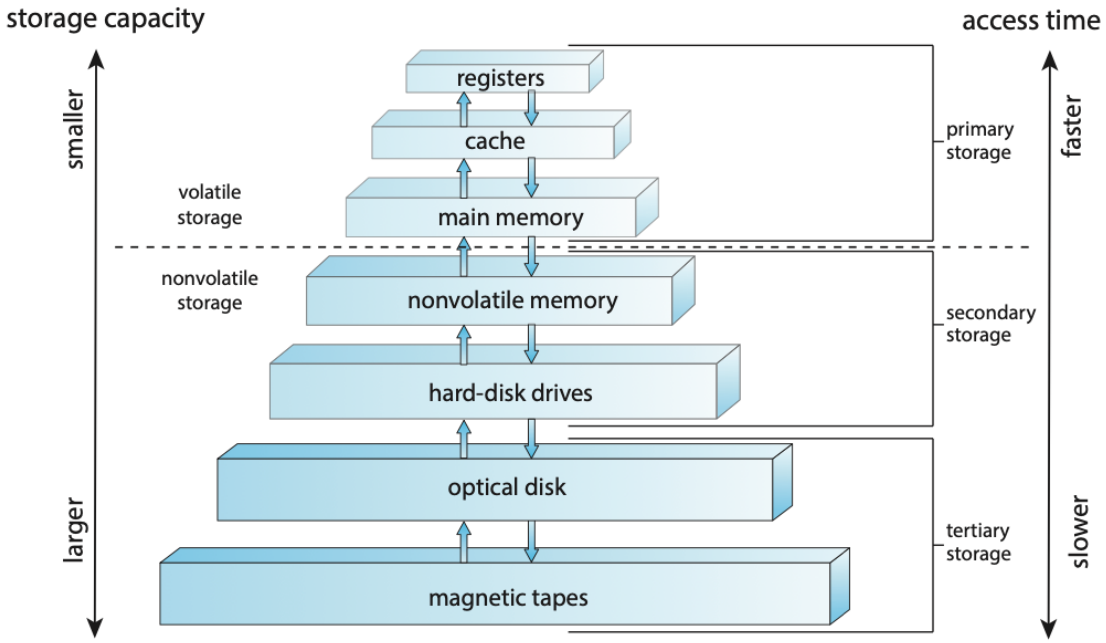
- 대량의 데이터를 영구히 보존할 수 있어야 한다. → 비 휘발성 메모리(NVM)
- 하드 디스크 드라이브(hard disk drive, HDD)
  : 부분의 프로그램은 메모리에 적재될 때까지 디스크에 저장된다.
- 반도체 디스크, Solid-state disk(SSD)
  : 하드 디스크보다 빠르며 비휘발성이다.

저장장치 계층

💡 저장장치 시스템의 다향성은 속도와 가격에 따라 하나의 계층으로 구성될 수 있다.

- 저장장치 시스템은 휘발성(volatility)일 수도 있고 비휘발성일 수 도 있다.

▼ <Figure> 저장장치 계층



- 메모리 시스템의 설계
  - 필요한 만큼만 값비싼 메모리를 사용하고, 가능한 한 많은 저렴한 비휘발성 메모리를 제공해야 한다.

입출력 구조

💡 범용 컴퓨터 시스템은 여러 개의 장치 제어기와 CPU들로 구성되며 이들은 공통 버스에 의해 연결되어 있다.

- 각 장치 제어기가 특정 유형의 장치를 담당한다.
  - 약간의 로컬 버퍼와 특수 목적용 레지스터 집합을 관리한다.
  - 자신이 제어하는 주변 장치와 자신의 로컬 버퍼 간의 데이터 전송을 담당한다.

- 운영체제는 각 장치 제어기 마다 디바이스 드라이버를 가지고 있다.

저장장치는 컴퓨터 내의 여러 형태의 입/출력 장치 중에 하나이다.

인터럽트 구동 방식의 입출력

- 디바이스 드라이버는 장치 제어기의 적절한 레지스터에 필요한 값을 적재한다.

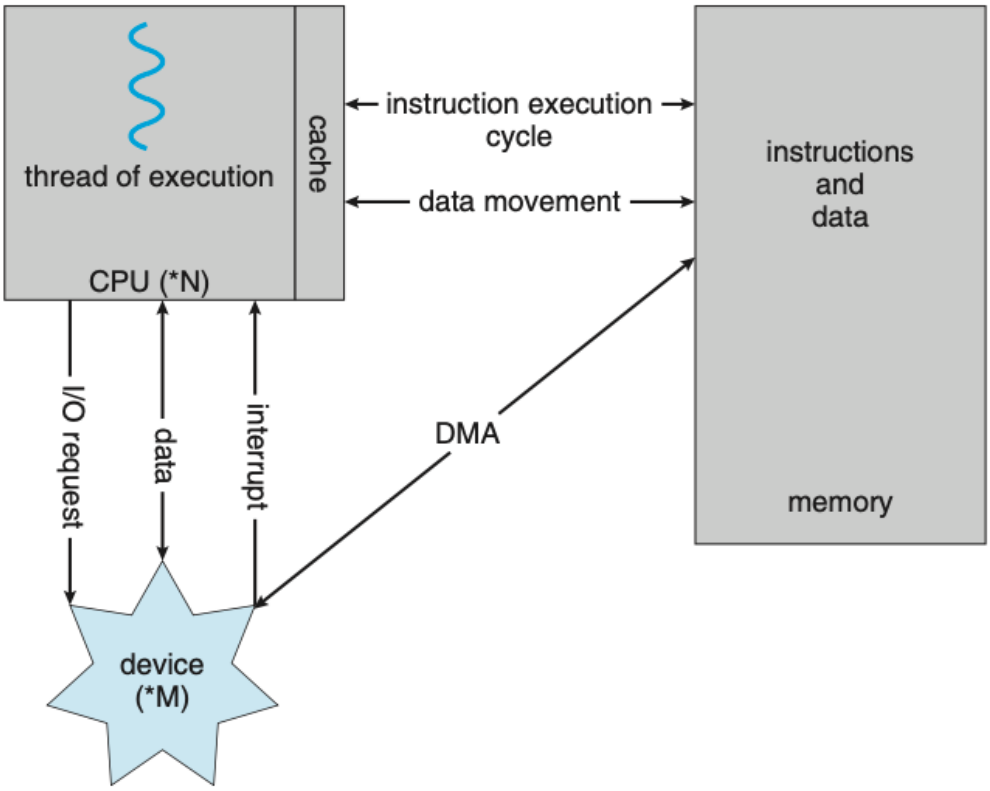
  → 장치 제어기는 동작을 결정하기 위해 레지스터의 내용을 조사하고, 장치로부터 자신의 로컬 버퍼로 데이터 전송을 시작하고 데이터 전송이 완료되면 연산을 완료했음을 인터럽트를 이용하여 디바이스 드라이버에게 통보한다

  → 디바이스 드라이버는 제어를 운영체제에게 반환하고
  입력이 완료된 경우에는 데이터 또는 데이터에 대한 포인터를 반환한다.

인터럽트 구동 방식의 입출력은 적은 양의 데이터를 전송하는 데에는 문제가 없으나 디스크 입출력과 같은 대량의 데이터를 전송하는 데에는 높은 오버헤드를 초래한다. → 직접 메모리 접근(DMA, Direct memory access)

직접 메모리 접근 [Direct Memory Access, DMA]

▼ <Figure> DMA 작동 방식, 현대의 컴퓨터 작동 방식



- 장치 제어기는 입출력 장치를 위해 버퍼, 포인터, 계수기 등을 설정한 후 CPU의 개입 없이 기억 장치와 저장 장치 간에 블록 단위로 전송한다.
- 블록 전송이 완료될 때 마다 인터럽트가 발생한다.
- 장치 제어기가 전송 작업을 실행하고 있는 동안 CPU는 다른 작업을 실행할 수 있다.

3. 컴퓨터 시스템 구조(Computer System Architecture)

> 💡 컴퓨터 시스템은 다양한 방식으로 분류 가능하고, 사용된 범용 처리기의 개수에 따라 분류 될 수 있다.

단일 처리기 시스템

> 범용 CPU가 하나 있는 시스템

사용자 프로세스의 명령어를 포함하여 범용 명령어 집합을 실행할 수 있는 하나의 주 CPU를 가진다.

다중 처리기 시스템

> 병렬 시스템, 멀티 코어 시스템

- 매우 밀접한 통신을 하는 둘 이상의 처리들을 가지며, 컴퓨터 버스, 클록, 메모리와 주변 장치들을 공유한다.
  - 초기에는 서버에 주로 사용되었으나 이후 데스크톱 및 랩톱, 최근에는 스마트폰, 태블릿 컴퓨터와 같은 휴대용 장치에도 등장하고 있다.

- 다중처리기 시스템의 주요 장점
  - 증가된 처리량(throughput) : 처리기의 개수를 늘리면 짧을 시간 동안 더욱 많은 일을 수행할 수 있다.

    ▼ 하지만 N 프로세서의 속도 향상 비율은 N이 아니다.

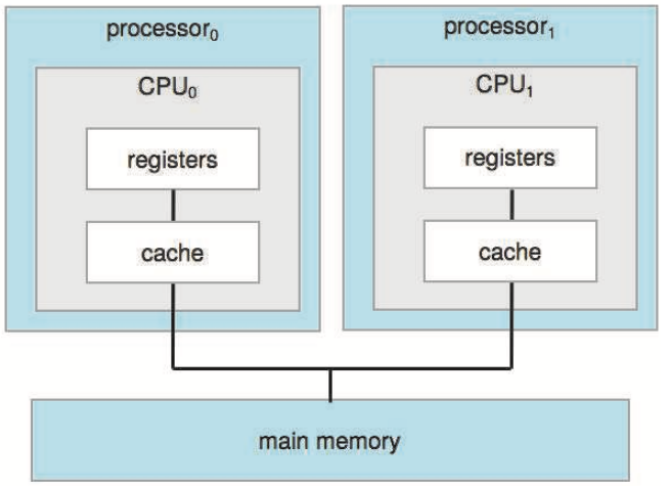
    여러 프로세서가 하나의 작업에 협력할 때 모든 프로세서가 올바르게 작동을 유지하는 데 일정한 양의 오버헤드가 발생한다. 이 오버헤드와 공유 자원에 대한 경합은 추가 프로세서의 예상 이득을 낮춘다.
  - 규모의 경제 : 여러 개의 단일 시스템에 비해 비용을 절약
  - 증가된 신뢰성 : 기능들이 여러 개의 처리기에 적절히 분산된다면, 한 처리기가 고장이 나도 시스템이 정지되지 않고 속도만 느려지게 된다.

- 컴퓨터 시스템의 증가된 신뢰성은 많은 응용 프로그램에게 필수적이다.
  - 우아한 퇴보(graceful degradation) : 살아남은 하드웨어의 비율만큼의 속도로 서비스를 계속 제공하는 능력
  - 결함 허용(fault tolerant) : 오류를 탐지하고, 진단하고, 가능하다면 교정할 수 있는 기법이 필요하다.

다중 처리기 시스템의 형태

- 비대칭적 다중 처리(asymmetric multiprocessing) : 주 처리기가 시스템을 제어하고 다른 처리기들은 주 처리기의 명령을 실행하거나 미리 정의된 태스크를 실행한다.
- 대칭적 다중 처리(SMP ; symmetric multiprocessing) : 각 처리기가 운영체제 내의 모든 작업을 실행한다. 현대의 거의 모든 운영체제가 SMP를 지원한다.

▼ <Figure> 대칭적 다중 처리 구조

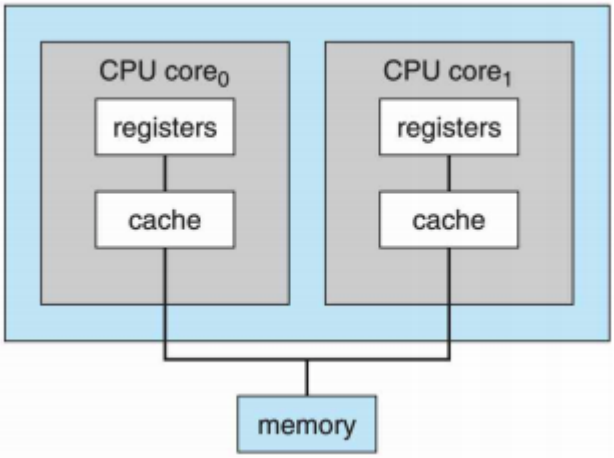


→ 대칭적 다중 처리와 비대칭적 다중 처리의 차이는 하드웨어나 소프트웨어의 결과일 수 있다.

멀티코어 시스템

- 하나의 칩에 여러 개의 코어(core)를 포함시키는 것
- 칩 내에서의 통신이 칩 사이의 통신보다 빠르기 때문에 하나의 코어를 가진 복수 칩 구조보다 효율적이다.

▼ <Figure> 멀티코어 시스템 구조

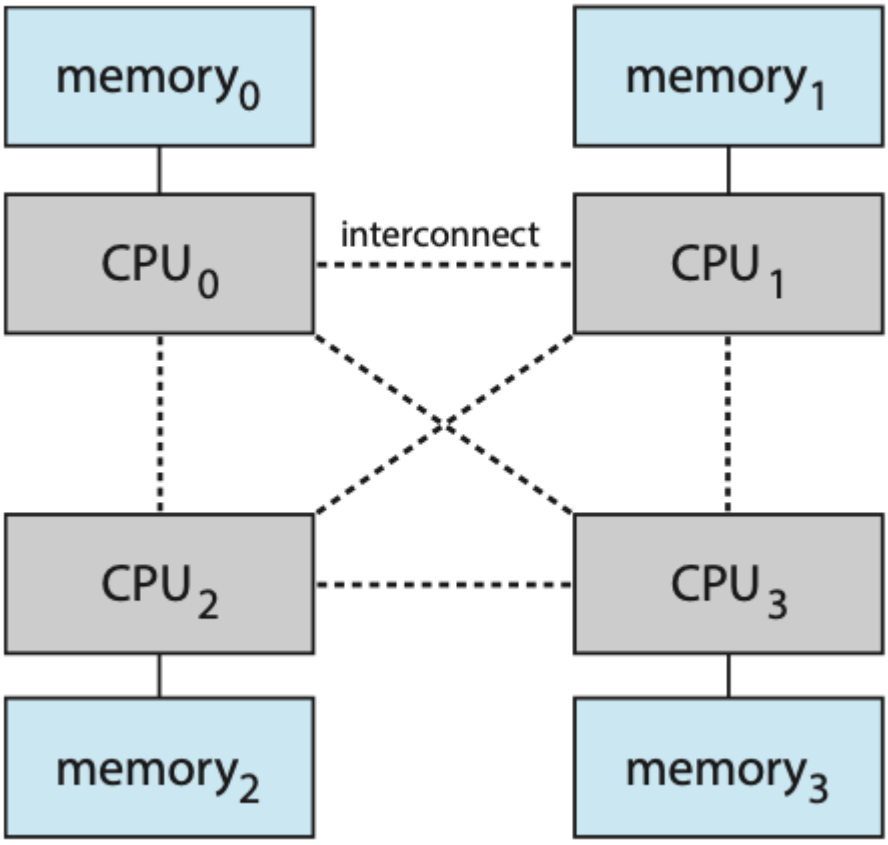


NAMA

> Non-uniform memory access

- 다중 처리기 시스템에 CPU를 추가하는 것에 대해, 예상보다 성능 저하가 일어나는 것에 대한 해결방법으로 제시
- 모든 CPU가 공유 시스템 연결로 연결되어 모든 CPU가 하나의 물리 공간을 공유하는 것을 말한다.

▼ <Figure> NAMA 구조



- CPU가 로컬 메모리에 엑세스 할 때 빠를 뿐만 아니라 시스템 상호 연결에 대한 경합이 없다.
- 잠재적 단점은 CPU가 시스템 상호 연결을 통해 원격 메모리에 액세스 해야할 때 지연 시간이 증가하여 성능 저하가 발생할 수 있다는 점이다.

클러스터형 시스템

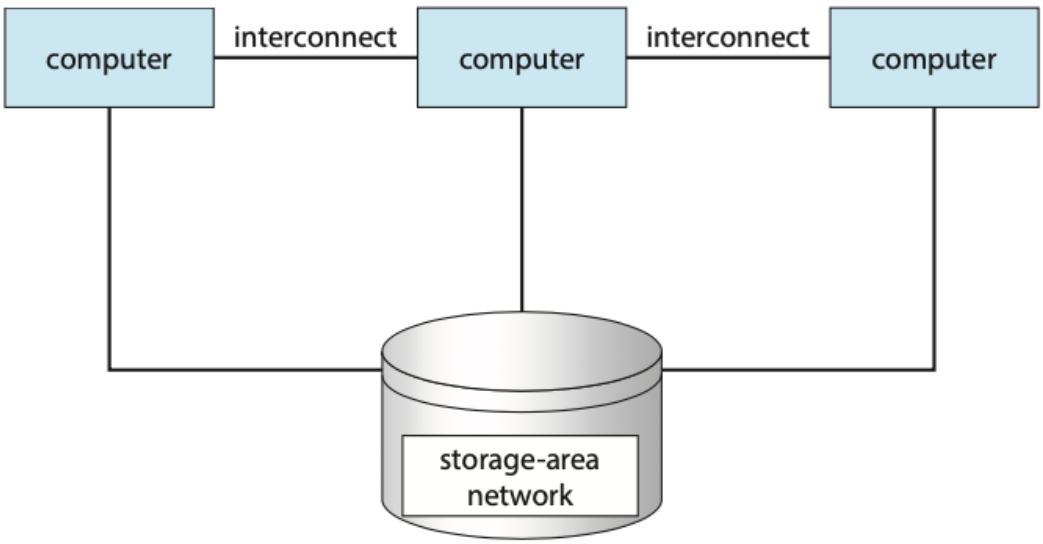
- 다중처리기 시스템과 같이 여러 개의 CPU들이 함께 계산 작업을 실행하지만 둘 이상의 독자적 시스템들을 연결하여 구성한다.

  → 저장장치를 공유하고 근거리 통신망이나 고속의 상호 연결망으로 연결된다.

- 통상 높은 가용도(availability)를 제공하기 위해 사용된다.
  - 클러스터 내 하나 이상의 컴퓨터 시스템이 고장이 나더라도 서비스는 계속 제공된다.

    - 고 가용성은 시스템에 중복 기능을 추가함으로써 얻어진다.
- 비대칭적, 대칭적으로 구성할 수 있다.
    - 비 대칭적 클러스터링 : 다른 컴퓨터들이 응용 프로그램을 실행하는 동안 한 컴퓨터는 긴급 대기모드 상태를 유지하고 서버가 고장이 나면 이 컴퓨터가 활성 서버가 된다.
    - 대칭형 클러스터링 : 둘 이상의 호스트들이 응용 프로그램을 실행하고 서로를 감시한다.
- 클러스터는 다수의 컴퓨터 시스템을 네트워크로 연결하여 구성하기 때문에 고성능 계산 환경을 제공하는데 이용될 수 있다.
    - 응용 프로그램을 클러스터의 모든 컴퓨터에서 동시에 실행시킬 수 있기 때문에, 단일처리기 또는 SMP 시스템에 비해 현저히 나은 계산 능력을 제공할 수 있다.
    - 응용 프로그램은 병렬화 기법을 사용하여 특별하게 작성되어야 한다.

▼ <Figure> 클러스터형 시스템 구조



4. 운영체제 구조

- 운영체제는 프로그램이 실행될 환경을 제공한다.

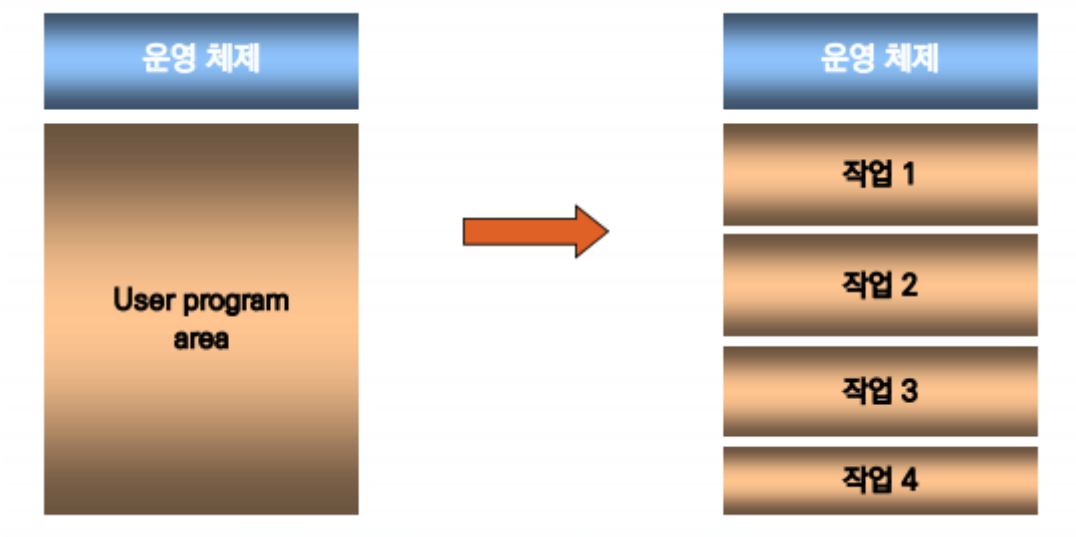
- 운영체제는 다양한 발전 방향을 따라 구성되어 내부 구조는 매우 다양하나 많은 공통점들이 존재한다.
  - 다중 프로그래밍
  - 시분할 시스템

다중 프로그래밍

> 💡 운영체제의 가장 중요한 측면 중 하나는 다중 프로그래밍을 할 수 있는 능력이다.

CPU가 항상 작업을 실행할 수 있도록 작업들을 구성함으로써 CPU 이용률을 증가시킨다.

▼ <Figure> 다중 프로그래밍 형태



- 한 번에 여러 작업을 메모리에 적재한다.
  1. 작업들은 처음에 디스크 상에 존재하는 작업 풀 내에 유지된다.
  2. 운영체제는 메모리 내에 있는 작업 중에서 하나를 선택해서 실행한다.

     → 작업이 입출력 등을 기다리는 상태가 되면 제어를 해당 작업으로 전환하고 입출력 작업을 실행한다.

     → 입출력을 완료하면 이전 작업으로 돌아와 작업을 이어서 실행한다.
  3. 작업이 종료되면 다시 2번으로 돌아가 작업을 다시 선택해서 위를 반복한다.
- 다중 프로그래밍은 여러 가지 시스템 자원을 효율적으로 이용할 수 있는 환경을 제공하지만, 사용자를 위해 컴퓨터 시스템과의 상호 작용은 제공하지 않는다.

시분할 시스템

> Timesharing System 또는 다중 태스킹(Multitasking)이라고도 한다.

💡 시분할 시스템(Timesharing system)은 다중 프로그래밍의 논리적인 확장이다.

시분할 시스템은 CPU가 다수의 작업들을 서로 교대로 실행하지만 매우 빈번하게 교대가 일어나기 때문에 프로그램이 실행되는 동안 사용자들은 각자 자기의 프로그램과 상호작용할 수 있다.

- 사용자와 시스템 간에 직접적인 통신을 제공하는 대화식(interactive) 컴퓨터 시스템을 필요로 한다.

  → 키보드나 마우스 같은 입력 장치를 사용하여 운영체제나 프로그램에 직접 명령하고 출력 장치의 즉각적인 응답을 기다린다.

- 일괄 처리 시스템 (Batched system) : 처리 속도 향상을 위해, 유사한 요구를 갖는 작업들을 함께 모아서 하나의 그룹으로 실행한다
  - 시스템 자원을 효율적으로 이용할 수 있지만, 작업이 실행되고 있는 동안 사용자와 작업 간의 대화가
    부족하므로 대화가 거의 필요 없는 큰 작업을 실행하는데 적합하다.
- 대화식 컴퓨터 시스템 (Interactive computer system) : 사용자와 시스템 간에 온라인 통신을 제공하여 사용자가 운영체제나 프로그램에 직접 명령을 주고 즉시 응답을 받을 수 있도록 한다.
  - 응답 시간이 아주 짧은 시간을 요구할 때 사용한다.

▼ 시분할 운영체제는 각 사용자에게 시분할되는 컴퓨터의 작은 부분을 제공하기 위해 CPU 스케줄링과 다중 프로그래밍을 사용한다.

- 사용자는 메모리에 최소한 하나의 독립된 프로그램을 가진다. → 프로세스 (process)
  - 짧은 CPU 시간만이 각 사용자에게 제공된다.
- 시스템에 들어온 모든 작업은 작업 풀에 보관된다.

        - 잡 스케줄링 : 작업 풀에서 메모리에 보관할 작업 선택
    - CPU 스케줄링(CPU scheduling)
        - 여러 개의 작업이 동시에 실행 준비가 되어 있으면 이들 중 하나를 선택한다.

▼ 시분할 시스템에서 운영체제는 적절한 응답 시간을 보장해야 한다.

- 적절한 응담 시간을 얻기 위해, 작업을 주 메모리에서 디스크로 적절하게 스왑 인/아웃한다.
- 가상 메모리(virtual memory) : 합리적인 응답 시간을 보장하는 더 일반적인 방법으로 작업의 일부만 메모리에 적재되어도 수행을 허용한다.

5. 운영체제 연산

> 💡 운영체제와 사용자는 컴퓨터 시스템의 하드웨어와 소프트웨어 자원을 공유하므로 사용자 프로그램의 오류가 실행 중인 프로그램에만 문제를 일으키도록 보장해야 한다.

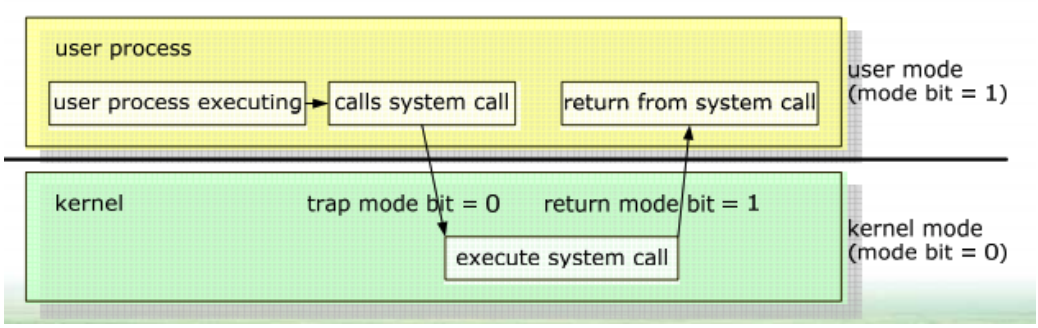
> 💡 운영체제는 잘못된(악의적인) 프로그램이 다른 프로그램(운영체제 포함)의 올바른 실행을 방해하지 못하도록 해야 한다.

이중 연산 모드

- 운영체제의 적절한 동작을 보장하기 위해 운영체제 코드의 실행과 사용자 정의 코드의 실행을 구분할 수 있어야 한다.
- 여러 실행 모드를 구분할 수 있도록 지원하는 하드웨어 기능을 이용한다.
  → 최소한 두 개의 분리된 연산 모드 지원 : 커널 모드(0) / 사용자 모드(1)
    - 사용자 모드(user mode)
        - 사용자에 의한 실행

    - 커널 모드(supervisor mode, system mode, kernel mode)
        - 운영체제에 의한 실행
- 작동의 이중 모드는 잘못된 사용자로부터 운영체제를 그리고 잘못된 사용자 서로를 보호할 수 있는 방법을 제공한다.
    - 악영향을 끼칠 수 있는 일부 명령을 특권 명령(privileged instruction)으로 지정하고, 하드웨어는 특권 명령이 커널 모드에서만 수행되도록 허용한다.
- 컴퓨터 시스템은 커널 모드에서 시작하여 제어가 사용자 응용으로 넘어가면 모드가 사용자 모드로 지정되고, 제어는 인터럽트, 트랩 또는 시스템 호출을 통하여 운영체제에게 다시 넘어오게 된다.
    - 시스템 호출은 운영체제가 사용자를 대신하여 하도록 예약되어 있는 작업을 운영체제에 요청하는 방법이다.

        ▼ <Figure> system call 작동 방식



        - 사용자 프로그램이 자신을 대신하여 운영체제가 수행하도록 지정되어 있는 작업을 운영체제에 요청할 수 있는 방법을 제공한다.

타이머

- 운영체제가 CPU에 대한 제어를 유지하도록 보장해야 한다.
    - 사용자 프로그램이 무한 루프에 빠지거나 시스템 서비스 호출에 실패하여 제어가 운영체제로 복귀하지 않는 경우를 허용해서는 안된다.
- **타이머(timer)의 사용**
    - 타이머는 지정된 시간 후 컴퓨터를 인터럽트 하도록 설정할 수 있다.

        → 사용자에게 제어를 넘기기 전에 타이머가 인터럽트 할 수 있도록 허용한다.

        1. 운영체제가 카운터 값 설정

2. clock tick마다 카운터 감소
3. 카운터가 0일때 인터럽트 발생
4. 타이머가 인터럽트 하면 제어가 자동으로 운영체제로 넘어간다.

- 타이머의 또 다른 용도
  - 시분할을 구현하기 위한 용도 / 현재 시간을 계산하기 위해서도 사용

6. 프로세스, 메모리, 저장장치 관리

- 프로그램은 그 명령이 CPU에 의해 실행되지 않으면 아무 일도 할 수 없다. → 실행중인 프로그램 → 프로세스
- 운영체제는 정보 저장장치에 대한 균등한 논리적 관점을 제공한다.
- 운영체제는 저장장치의 물리적 특성을 추상화하여 논리적인 저장 단위인 파일을 저장한다.
- **프로그램** : 디스크 상에 저장된 파일의 내용과 같이 수동적 개체
- **프로세스** : 다음 수행할 명령을 지정하는 프로그램 계수기를 가진 능동적 개체

프로세스 관리

- 프로세스는 자신의 일을 수행하기 위하여 자원(CPU 시간, 메모리, 파일, 입출력 장치)들을 필요로 한다.
  - 자원은 프로세스가 생성될 때 제공될 수도 있고 실행되는 동안 할당될 수도 있다.
  - 프로세스가 끝나면 운영체제는 재사용할 수 있는 자원을 회수할 것이다.

- 한 프로세스의 실행은 반드시 순차적이어야 한다.
  - CPU는 그 프로세스가 끝날 때까지 그 프로세스를 위해 하나의 명령만 실행된다.
  - 두 개의 프로세스가 동일한 프로그램과 연관되어 있더라도 이들은 두 개의 별도의 실행 순서로 간주된다.

    - 다중 스레드 프로세스는 복수개의 프로그램 카운터를 가지면 이 카운터들은 각 스레드가 실행할 다음 명령어를 가리키게 된다.

- 한 프로세스는 한 시스템 내의 작업의 단위이다.
    - 프로세스들 중 일부는 운영체제 프로세스들이고 나머지는 사용자 프로세스들이다.
    - 모든 프로세스는 그들 간에 하나의 CPU를 멀티플렉싱 함으로써 병행 수행될 수 있다.

- 운영체제는 프로세스 관리와 연관해 다음의 활동에 대한 책임을 진다.
    - CPU에 프로세스와 스레드를 스케줄하기
    - 사용자 프로세스와 시스템 프로세스들의 생성과 제거
    - 프로세스의 일시 중지와 재실행
    - 프로세스 동기화를 위한 기법 제공
    - 프로세스 통신을 위한 기법 제공

메모리 관리

- 메인 메모리는 CPU와 입출력 장치에 의하여 공유되는, 빠른 접근이 가능한 데이터의 저장소이다.
    - 메인 메모리는 일반적으로 CPU가 직접 주소를 지정할 수 있고, 직접 접근할 수 있는 유일한 대량 메모리이다.

- 프로그램이 실행되기 위해서는 반드시 절대 주소로 매핑되고 메모리에 적재되어야 한다.
    - 프로그램을 수행하면서 절대 주소를 생성하여 메모리의 프로그램 명령어와 데이터에 접근한다.
    - 프로그램이 종료되고 프로그램이 차지하던 메모리 공간은 가용공간으로 선언되고, 다음 프로그램이 적재되어 수행될 수 있다.

- CPU 이용률과 컴퓨터 응답 속도를 개선하기 위해 메모리에 여러 개의 프로그램을 저장한다. ← multiprogramming
    - 특정 시스템에 대한 메모리 관리 기법의 선택은 여러 가지 요인들에 의해 결정되지만, 특히 하드웨어 설계에 의하여 주로 결정된다.

- 운영체제는 메모리 관리와 관련하여 다음 일을 담당한다.
    - 메모리의 어느 부분이 현재 사용되고 있으며 누구에 의하여 사용되고 있는지를 추적한다.
    - 메모리 공간이 유용할 때 어떤 프로세스들을 메모리에 저장할 것인가를 결정한다.
    - 메모리 공간을 할당하고 회수하는 방법을 제공한다.

저장장치 관리

파일 시스템 관리

각기 다른 특성과 물리적 다른 구성을 가지는 저장 매체들

- 파일 : 파일 생성자에 의해 정의된 관련 정보의 집합체

- 운영체제는 디스크 같은 대량 저장 매체와 그것을 제어하는 장치를 관리함으로써 파일의 추상적인 개념을 구현한다.
    - 다수의 사용자가 파일을 접근하고자 할 때 누구에 의해서, 어떤 방법으로 파일이 접근되어야 하는가를 통제하는 것이 바람직하다.
- 운영체제는 파일 관리를 위한 일을 담당한다.
    1. 파일의 생성 및 제거
    2. 디렉토리 생성 및 제거
    3. 파일과 디렉토리 조작을 위한 프리미티브의 제공
    4. 파일을 보조 저장장치로 매핑
    5. 안정적인 저장 매체에서 파일을 백업

대용장 저장장치 관리

- 메인 메모리는 모든 데이터와 프로그램을 수용하기에 용량이 작고, 전원이 꺼질 경우 저장하고 있던 데이터가 사라진다.
    - 컴퓨터 시스템은 반드시 메인 메모리 내용을 저장하기 위해 보조 저장장치를 제공해야 한다.
    - 대부분의 컴퓨터 시스템은 디스크를 프로그램과 데이터를 위한 온라인 저장 매체로 사용하고 있다.
- 운영체제의 디스크 관리 기능
    - 자유 공간 관리
    - 저장 장소 할당
    - 디스크 스케줄링
- 보조 기억장치는 매우 빈번하게 사용되므로 효율적으로 사용해야 한다. → 알고리즘의 속도와 연관
- 보조 저장장치보다 느리고 비용을 적게 들고 더 큰 장치를 사용하는 경우도 많다.
- 디스크 데이터의 백업, 가끔 사용하는 데이터 및 장기간 보존이 필요한 데이터를 저장한다. → 3차 저장장치

캐싱

정보는 통상 어떤 저장장치에 보관되며 정보가 사용됨에 따라, 보다 빠른 장치(캐시)에 일시적으로 복사된다.

▼ <Figure> 저장장치 비교

층	1	2	3	4	5
명칭	레지스터	캐시	메인 메모리	반도체 디스크	하드 디스크
Typical size	< 1KB	< 16MB	< 64GB	< 1TB	< 10TB
접근 시간 (ns)	0.25 ~ 0.5	0.5 ~ 25	80 ~ 250	25,000 ~ 50,000	5,000,000
대역폭 (MB/sec)	20,000 ~ 100,000	5,000 ~ 10,000	1,000 ~ 5,000	500	20 ~ 500
Managed by	컴파일러	하드웨어	운영체제	운영체제	운영체제

- 각 층 간의 정보 이동은 하드웨어 설계나 운영체제에 따라 명시적으로 또는 묵시적으로 이루어진다
  - √ 캐시/메모리/레지스터 간의 전송은 하드웨어적으로 이루어진다
  - √ 디스크/메모리 간의 전송은 운영체제에 의해 이루어진다

- 캐시 일관성 : 한 캐시에 있는 A값이 갱신될 경우, A가 존재하는 모든 캐시에 즉각적으로 반영되어야 한다.

입출력 시스템

운영체제의 목적 중의 하나는 사용자에게 특정 하드웨어 장치를 숨기는 것이다.

- 장치 드라이버 만이 관련 특정 장치의 특성을 안다.

- 입출력 시스템 구성
  - 버퍼링, 캐싱, 스풀링을 포함한 메모리 관리 구성 요소
  - 일반적인 장치 드라이버 인터페이스
  - 특정 하드웨어 장치들을 위한 드라이버

7. 보호와 보안

- 컴퓨터 시스템이 다수의 사용자를 가지며, 다수의 프로세스의 병렬 실행을 허용한다면, 데이터에 대한 접근은 반드시 규제되어야 한다.

보호

> 💡 컴퓨터 시스템의 정의한 자원에 대해 프로그램, 프로세스, 사용자들의 접근을 제어하는 기법

시행될 제어에 대한 명세와 이들을 강제 시행하기 위한 방법을 규정하는 수단을 제공해야 한다.

- 보호는 구성 요소 서브 시스템 간의 인터페이스에서 잠재적인 오류를 검출함으로써 시스템의 신뢰성을 증가시킬 수 있다.
- 보호 지향 시스템은 허가 받은 사용과 그렇지 않은 사용을 구별하는 방법을 제공한다.

보안

> 💡 외부 또는 내부의 공격을 방어하는 것

컴퓨터 시스템은 충분한 보호 기능을 가지고 있더라도 여전히 고장이 나거나, 부적절한 접근을 허용할 수 있다.

- 공격 : 바이러스, 웜, 서비스 거부 공격, 식별자 도용, 서비스 도용 등

대안

> 💡 보호와 보안을 제공하기 위해 시스템의 모든 사용자를 구분할 수 있어야 한다.

운영체제의 대부분은 사용자 이름과 연관된 사용자 식별자(user IDs)의 리스트를 유지한다. 이 식별자는 사용자마다 할당되고 시스템에서 유일한 값을 가진다. 사용자가 로그인 할 때 인증 단계어서 사용자에게 맞는 적절한 식별자를 결정한다.

8. 커널 자료 구조

- 시스템에서 데이터를 조직적으로 구성하는 방법을 다룬다.

리스트, 스택 및 큐

- 리스트 : 데이터 값들의 집단을 하나의 순서로 표시한다.
  - 배열, 연결 리스트
- 스택 : 후입선출(Last in first out, LIFO)
- 큐 : 선입선출(First in first out, FIFO)

트리

- 트리는 데이터를 계층적 구조로 구성하는데 사용된다.
  - 이진 트리(binary tree)
  - 이진 탐색 트리(binary search tree)

해시 함수와 맵

- 해시 함수 : 데이터를 입력으로 받아 이 데이터에 산술 연살을 수행하여 하나의 수를 반환하고 이 수는 데이터를 인출하기 위해 테이블의 인덱스로 사용할 수 있다.
- 해시 맵의 구현 : 해시 함수를 이용하여 `[키:값]`을 연관시킨다.

비트맵

n개의 항의 상태를 나타내는 사용 가능한 n개의 이진 비트의 스트링

- 자원의 가용 여부를 이진 비트의 값으로 표현할 수 있다.
  - 비트 맵을 사용하면 자료구조의 크기면에서 이득을 볼 수 있으므로, 대량의 자원의 가용성을 표시할 때 사용한다.

9. 계산 환경

다양한 계산 환경에서의 운영체제의 사용에 대해 살펴본다.

전통적 계산

- 컴퓨터 기술이 성숙해짐에 따라 여러 전통적인 계산 환경의 경계가 흐려지고 있다. 현재는 원격 접근과 이동성을 다양하게 제공한다.
  - 회사들은 자신의 내부 서버에 웹 접근을 제공하는 포털을 구현한다.
  - 네트워크 컴퓨터 : 웹 기반 컴퓨팅을 위한 터미널
  - 휴대용 컴퓨터 : 무선 네트워크나 이동전화 망에 연결되어 웹 포털 사용

모바일 컴퓨팅

- 휴대용 스마트폰과 태클릿 컴퓨터의 계산
  - 이동 가능하고 가볍다.
  - 화면 크기, 메모리 용량과 전체적인 기능을 포기한 대신 이메일, 웹 브라우징 등의 서비스에 대해 손에 쥘 수 있는 휴대용 접근을 얻은 것 → 휴대용 장치의 기능이 풍부해지면서 랩톱, 태블릿 사이의 기능 구분이 명확하지 않게 되었다.
- 휴대용 컴퓨팅의 두드러진 운영체제
  1. Apple iOS
  2. Google Android

분산 시스템

> 💡 물리적으로 떨어져 있는 이기종 컴퓨터들의 집합

- 분산 시스템들의 컴퓨터들은 시스템 내의 다양한 자원들을 접근할 수 있도록 네트워크로 연결되어 있다.
- 공유 자원에 대한 접근은 계산 속도와 기능, 데이터 가용성 및 신뢰성을 향상시킨다.

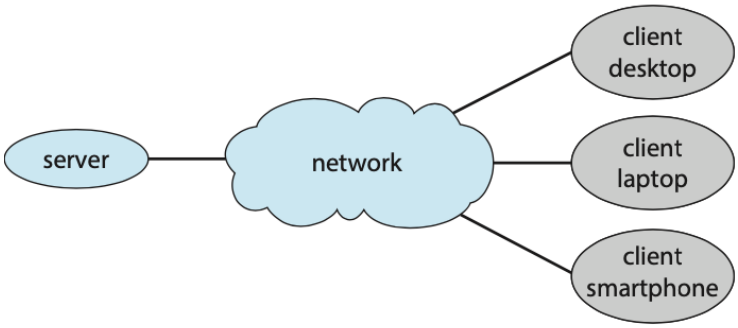
- 네트워크는 노드 간의 거리에 의해 유형이 결정된다.
  - 근거리 통신망(LAN), 원거리 통신망(WAN), MAN, PAN ...
- 운영체제에서는?
  - 어떤 운영체제에서는 네트워킹의 자세한 사항을 네트워크 인터페이스의 장치 드라이버에 포함시킴으로써 네트워크에 대한 접근을 파일 접근 형태로 일반화한다. → NFS
  - 다른 운영체제들은 사용자가 특정 네트워크 기능을 명시할 수 있게 한다. → FTP
  - 어떤 운영체제들은 네트워크 연결을 제공하는 타원을 넘어 네트워크와 분산 시스템의 개념을 더 확대한다. → 네트워크 운영체제
    - 네트워크 운영체제 : 네트워크를 통한 파일의 공유, 다른 컴퓨터 상에 존재하는 다른 프로세스들까지의 메시지 교환을 제공하는 운영체제

클라이언트 서버 컴퓨팅

> Client Server Computing

- 중앙식 시스템에 연결된 터미널은 PC에 의해 대체되고 있다.
  - 중앙 시스템에 의해 직접 취급되던 사용자 인터페이스 기능들도 PC에 의해 처리된다.
- 클라이언트 - 서버 시스템
  - 시스템은 클라이언트 시스템이 생성하는 요구를 만족시키기 위한 서버 시스템으로 동작한다.
  - 계산 - 서버 시스템
    - 클라이언트가 작업을 요청할 수 있는 인터페이스 제공
  - 파일 - 서버 시스템
    - 클라이언트가 파일을 생성, 갱신, 읽기 및 제거를 할 수 있는 파일 시스템 인터페이스 제공

▼ <Figure> 클라이언트-서버 시스템의 일반적인 구조

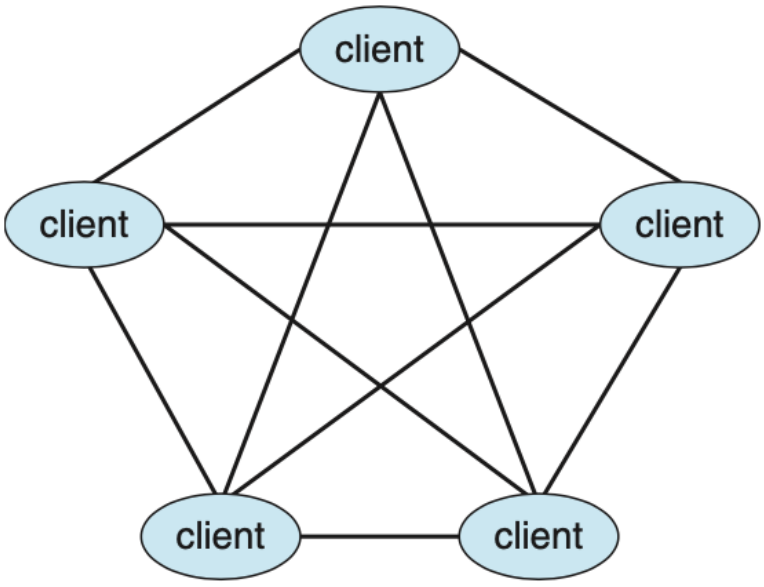


피어 간 계산

> Peer to peer Computing

💡 클라이언트와 서버가 구별되지 않고 대신 시스템 상의 모든 노드가 피어로 간주되고 각 피어는 서비스를 요청하느냐 제공하느냐에 따라 클라이언트 및 서버로 동작한다.

▼ <Figure> 파이 간 시스템 구조

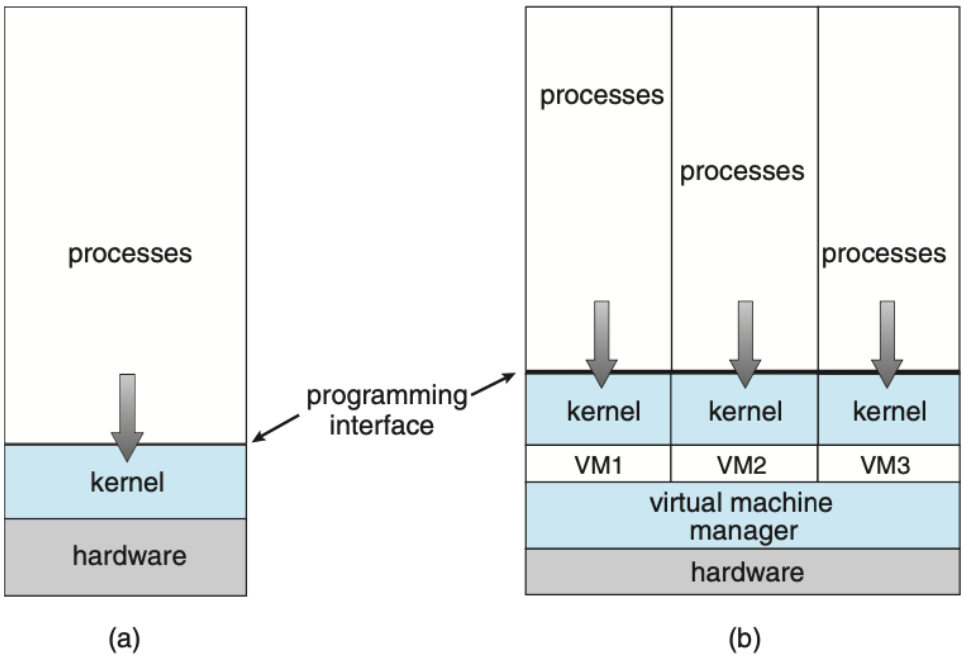


가상화

> 💡 가상화는 운영체제가 다른 운영체제의 응용처럼 실행될 수 있게 한다.
> 단일 컴퓨터의 하드웨어를 여러 가지 실행 환경으로 추상화하여 개별 환경이 자신만의 컴퓨터에서 실행되고 있다는 환상을 만들 수 있는 기술이다.

- 가상 머신을 통해 사용자는 단일 운영체제에서 동시에 실행되는 다양한 프로세스 간의 전환할 수 있는 것과 동일한 방식으로 다양한 운영체제 간에 전환할 수 있다.
- 부가적인 기능을 제공하는 것이 아니라 하드웨어 자체를 사용할 때와 동일한 인터페이스를 제공하는 것

▼ <Figure> 가상 머신을 사용하는 컴퓨터



→ 극히 일부분에 대한 내용이고 이후 장에서 더 다룸

클라우드 컴퓨팅

> 💡 계산, 저장장치는 물론 응용조차도 네트워크를 통한 서비스로 제공하는 계산 유형

- 가상화를 그 기능의 기반으로 사용하기 때문에 가상화의 논리적 확장으로 볼 수 있다.

여러 유형의 클라우드 컴퓨팅이 존재한다.

- Public / Private / Hybrid cloud
- 소프트웨어 서비스, Software as a Service (SaaS)
- 플랫폼 서비스, Platform as a service (PaaS)
- 기간시설 서비스, Infrastructure as a Service (IaaS)

실시간 내장형 시스템

현재 가장 유행하는 컴퓨터의 형태이다.

- 이 장치들은 아주 특정한 작업만을 실행하는 경향이 있다.

내장형 시스템은 대부분 실시간 운영체제를 실행한다.

- 실시간 시스템 : 처리기의 작동이나 데이터의 흐름이 엄격한 시간 제약이 있을 때 사용
  - 전용 응용에서 제어 장치로 사용된다.
  - 실시간 시스템은 잘 정의된 고정된 시간 제약을 가진다.

10. 오픈소스 운영체제

- 컴파일 된 이진 형태보다는 소스 코드 형태로 받을 수 있는 운영체제를 의미한다.
- 관심 있는 프로그래머들의 공동체를 들 수 있으며, 이 프로그래머들은 디버깅에 도움을 주고, 코드를 분석하고, 지원을 제공하고 수정 사항을 제안함으로써 코드 발전에 기여한다.