1회차 - 운영체제

운영체제

- 운영체제 : 사용자가 컴퓨터를 쉽게 다루게 해주는 **인터페이스**
- **모든 자원은 한정되어있다** → 효율적으로 분배하는 과정이 필요

> 💡 펌웨어 : 운영체제와 유사하지만 소프트웨어를 추가로 설치할 수 없는 것

3.1 운영체제와 컴퓨터

3.1.1 운영체제의 역할과 구조

운영 체제 역할

1. CPU 스케줄링
    - CPU 소유권을 어떤 프로세스에 할당할지 결정
    - + 프로세스 관리 : 프로세스의 생성/삭제, 자원 할당/반환 관리

2. 메모리 관리
    - 메모리를 어떤 프로세스에 얼만큼 할당할지 결정
    - 포인트 : **메모리는 한정되어 있다!**

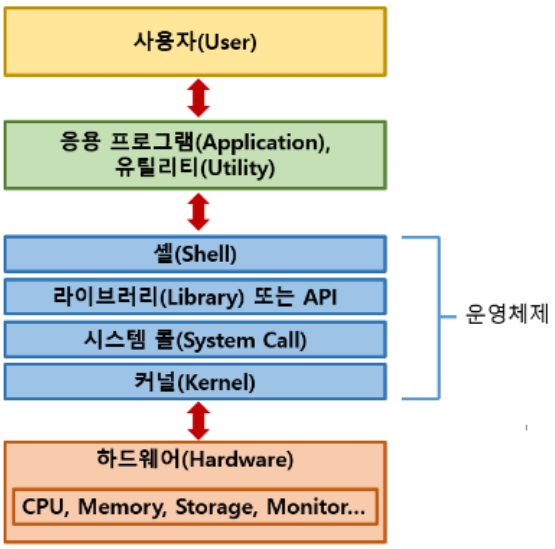
3. 디스크 파일 관리
    - 디스크 파일을 보관할 방법 결정

- 나중에 나올 파일 시스템과 연결되는 지점

4. I/O 디바이스 관리
    - I/O 디바이스와 컴퓨터의 데이터를 주고받는 과정 관리
    - I/O 디바이스 : 입출력 장치
        - 입력 : 키보드, 마우스, 카메라, 마이크, ..
        - 출력 : 모니터, 프린터, 스피커, ...

운영체제 구조

▲ 그림 3-1 운영체제의 구조



> 운영체제는 사용자와 하드웨어 사이에서 일하는 친구!

💡 GUI(Graphical User Interface) : **그래픽 사용자 인터페이스**를 뜻하며, 사용자가 컴퓨터와 상호작용할 수 있도록 시각적 요소를 사용하는 인터페이스입니다.

CUI(Command User Interface) : 키보드 입력을 통해 명령어를 실행하는 **텍스트 기반 사용자 인터페이스**입니다.

시스템 콜

시스템 콜 : 운영체제가 제공하는 기능을 응용 프로그램이 사용할 수 있도록 커널에 요청하는 **인터페이스**입니다.

- 네트워크 통신이나 데이터베이스와 같은, 낮은 단계의 영역 처리에 대한 부분을 신경쓰지 않아도 프로그램 구현이 가능!

💡 커널?

- 운영체제의 핵심이자 **시스템 콜 인터페이스를 제공**
- 보안, 메모리, 프로세스, 파일 시스템, I/O 디바이스, I/O 요청 관리 등 중요한 역할을 함

시스템 콜 과정

1. 유저 프로그램이 I/O 요청으로 트랩 발동
2. 올바른 I/O 요청인지 확인
3. 시스템 콜을 통해, **유저 모드 에서 커널 모드로 변환**

4. 커널 모드인 상태로 요청을 수행(예 : 파일 읽기) 후, 다시 유저 모드로 돌아감
5. 뒤에 이어지는 유저 프로그램의 로직 수행

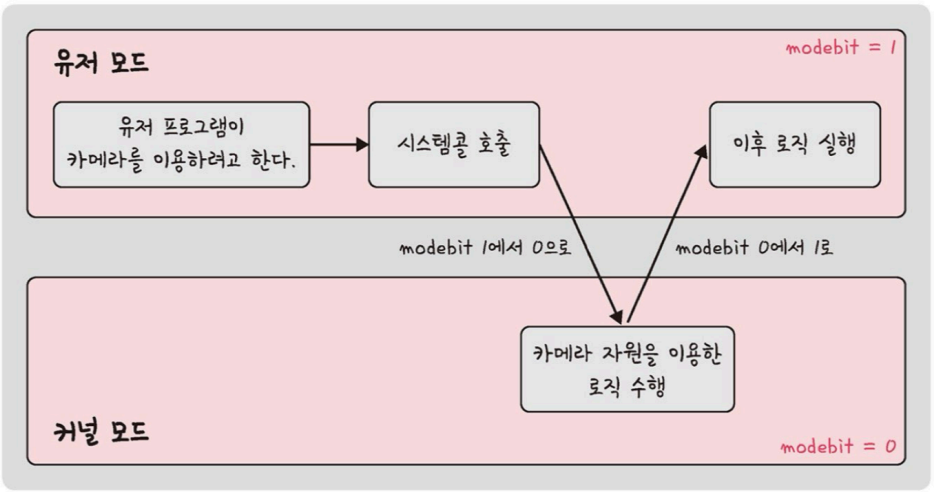
💡 커널 모드?

- 왜 번거롭게 커널모드로 변환할까? ⇒ **보안과 안정성 때문에!**
  1. 컴퓨터 자원에 대한 직접 접근 차단 → 시스템 안정성 향상
  2. 프로그램을 다른 프로그램으로부터 보호 → 프로세스 간 간섭 방지

  ex. I/O 디바이스는 운영체제를 통해서만 작동해야하기 때문에, 커널 모드를 거쳐서 작동됨

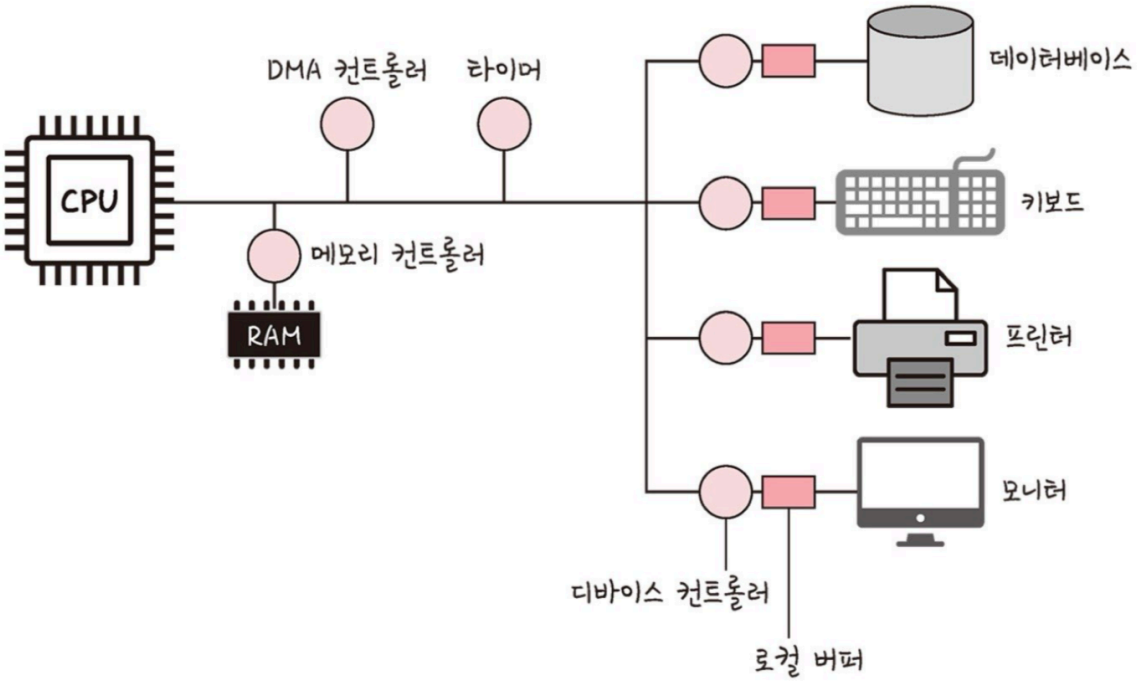
+
modebit : 유저 모드와 커널 모드를 구분하는 비트

- 0 : 커널 모드
- 1 : 유저 모드



▲ 그림 3-4 modebit의 역할

3.1.2 컴퓨터의 요소



▲ 그림 3-5 컴퓨터의 요소

CPU

CPU : 인터럽트에 의해 단순히 메모리에 존재하는 명령어를 해석, 실행하는 일꾼

- 커널이 프로그램을 메모리에 올림 → 프로세스 생성
- CPU가 메모리에 올라온 프로세스 명령어를 처리

> 💡 인터럽트 : CPU가 실행 중인 작업을 잠시 멈추고, 긴급한 작업(입출력 요청, 예외 처리 등)을 먼저 처리하도록 하는 신호입니다.

CPU의 구성

- 산술 논리 연산 장치(ALU)

실질적인 산술/논리 연산을 처리

- 제어 장치 (CU)

  역할 1. 입출력장치 간 통신 제어

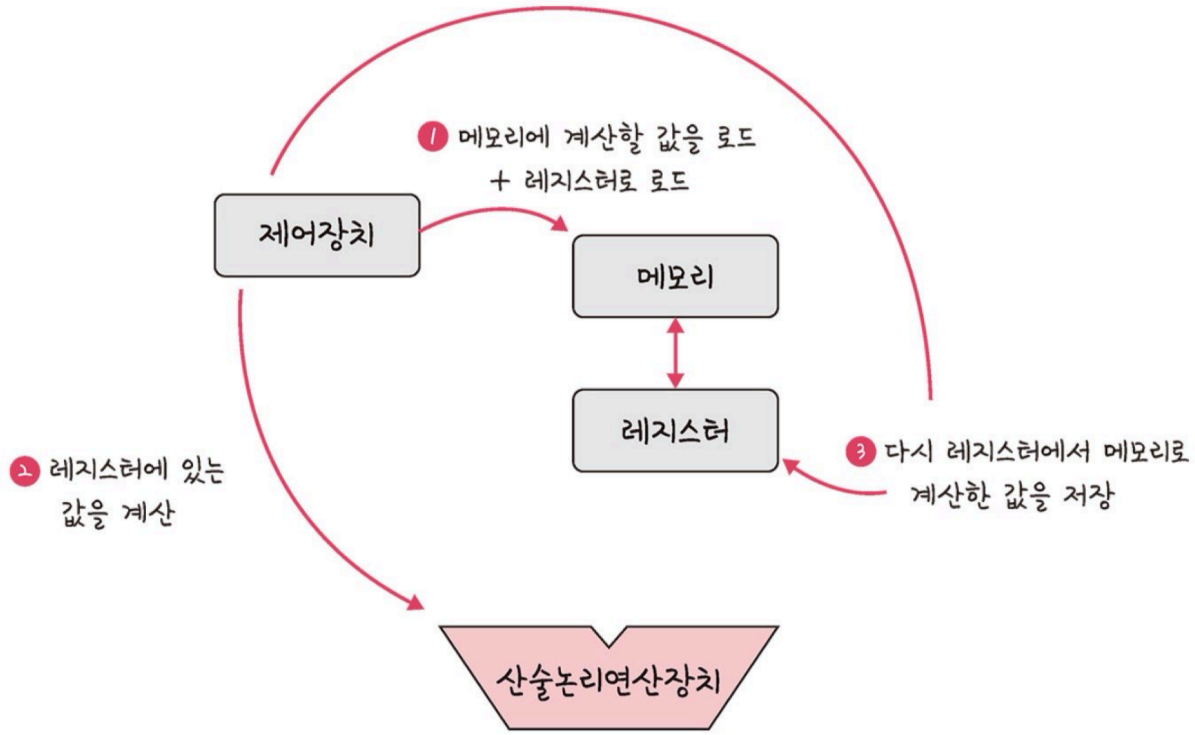
  역할 2. 명령어 읽기/해석하기

  역할 3. 데이터 처리 순서 결정

- 레지스터 (Register)

  임시 기억 장치 → 제일 빠른 기억장치임!

  CPU에 전달되는 모든 데이터는 레지스터를 거쳐 전달됨



▲ 그림 3-7 CPU 연산 처리

인터럽트

- CPU 너 멈춰!!
- 종류 : IO 디바이스로 인한 인터럽트, 산술 연산에서의 인터럽트, 프로세스 오류
  - 하드웨어 인터럽트 : IO 디바이스에서 발생하는 인터럽트

    - 소프트웨어 인터럽트 : 프로세스 오류 등으로 프로세스가 시스템콜을 호출할 때 발동 (trap 이라고도 부른다)

- 인터럽트 발생 과정
    1. 인터럽트 발생
    2. 인터럽트 벡터로 이동
    3. 인터럽트 벡터에 있는 인터럽트 핸들러 함수 실행

    > 💡 **인터럽트 벡터** : 인터럽트 핸들러 함수가 모여 있는 공간
    >
    > **인터럽트 핸들러 함수** : 인터럽트를 핸들링하기 위한 함수. 커널 내부의 **IRQ**를 통해 호출된다!

- **인터럽트에는 우선 순위가 있음!!**
- 하드웨어 인터럽트 : IO 디바이스에서 발생하는 인터럽트

  예시 : 키보드 'A' 입력
    1. 입력 발생 → 키보드가 CPU에게 인터럽트 신호를 보낸다
    2. CPU가 인터럽트 신호를 받음 → 하던 일을 잠시 멈추고, OS가 인터럽트를 처리하도록 넘긴다
    3. OS가 인터럽트를 처리
        a. 어떤 장치가 인터럽트를 발생시켰는지 확인
        b. 키보드의 '로컬 버퍼'에서 입력된 데이터(A)를 가져옴
        c. 입력 데이터를 적절한 곳(예 : 모니터 화면)에 전달 후 인터럽트 처리 마무리
    4. 인터럽트 처리 완료 → CPU는 원래 하던 일을 마저 한다

    - 왜 중요할까? → CPU가 **IO 디바이스를 계속 확인할 필요 없이, 디바이스가 먼저 신호를 보내는 방식!**

DMA 컨트롤러

DMA 컨트롤러 : **CPU의 개입 없이 I/O 장치와 메모리 간 데이터를 직접 전송**하는 하드웨어 장치

- CPU 대신 메모리로 데이터를 옮겨주는 친구!

왜 필요해?

- CPU에만 너무 많은 인터럽트 요청이 들어오면 → CPU 부하 발생
- CPU 일을 도와주는 보조 일꾼
- CPU의 일을 덜어준다 → 필수인 것은 아니지만, 없다면 효율이 떨어진다!

DMA 컨트롤러와 CPU가 하나의 작업을 동시에 하는 것을 방지한다?

- 사실 DMA가 하는 일은 CPU도 할 수 있다
- 근데 둘이 하나의 작업을 동시에 하면? ⇒ 데이터 충돌 OR 다른 값 저장
- 그래서 **CPU가 일하는 동안에는 DMA가 대기하고, DMA가 일하는 동안에는 CPU가 개입하지 않도록 제어하는 것**이 핵심

메모리

메모리 : 데이터나 상태, 명령어 등을 기록하는 장치

- CPU는 계산 담당, 메모리는 기억 담당!

보통 RAM을 메모리라고 부른다!

타이머

타이머 : 특정 프로그램에 시간 제한을 거는 역할

디바이스 컨트롤러

디바이스 컨트롤러 : I/O 디바이스들의 작은 CPU

로컬 버퍼 : 각 디바이스에서 데이터를 임시로 저장하기 위한 작은 메모리