Ch.3정리

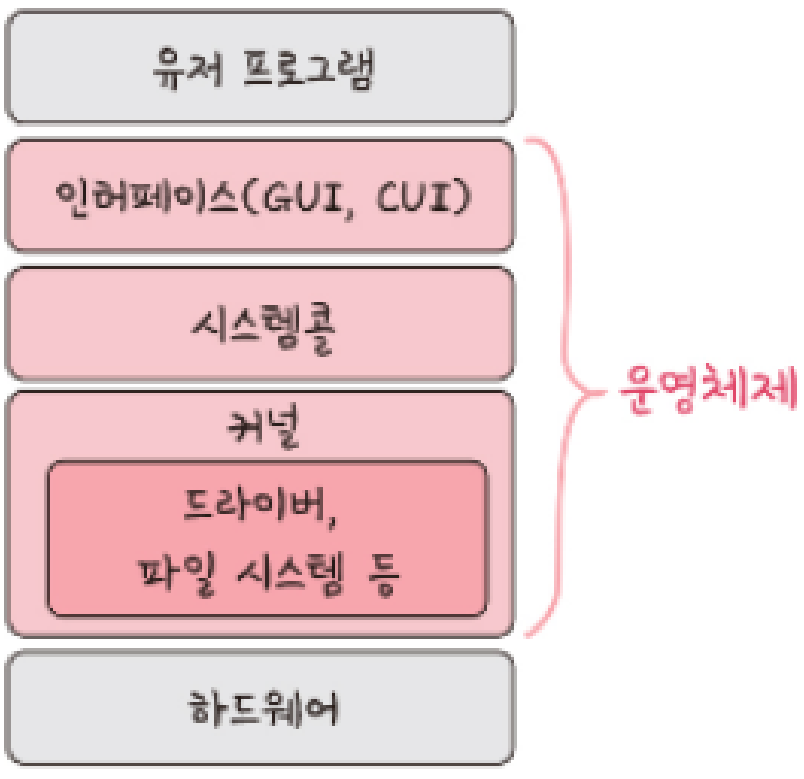
1. 운영체제와 컴퓨터

→ **운영체제란**

- 사용자가 컴퓨터를 쉽게 다루게 해주는 인터페이스(한정된 메모리나 시스템 자원을 효율적으로 분배)

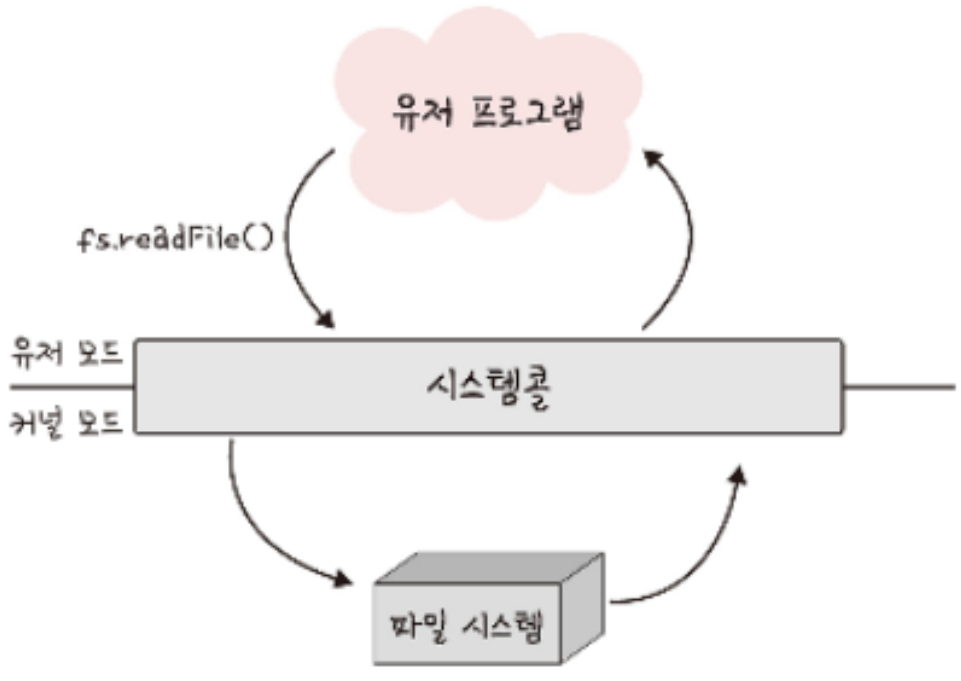
→ **운영체제의 역할**

- **CPU 스케줄링과 프로세스 관리** : CPU 소유권을 어떤 프로세스에 할당할지, 프로세스의 생성과 삭제, 자원 할당 및 반활을 관리
- **메모리 관리** : 한정된 메모리를 어떤 프로세스에 얼만큼 할당해야 하는지 관리
- **디스크 파일 관리** : 디스크 파일을 어떠한 방법으로 보관할지 관리
- **I/O 디바이스 관리** : I/O 디바이스들인 마우스, 키보드와 컴퓨터 간에 데이터를 주고 받는 것을 관리

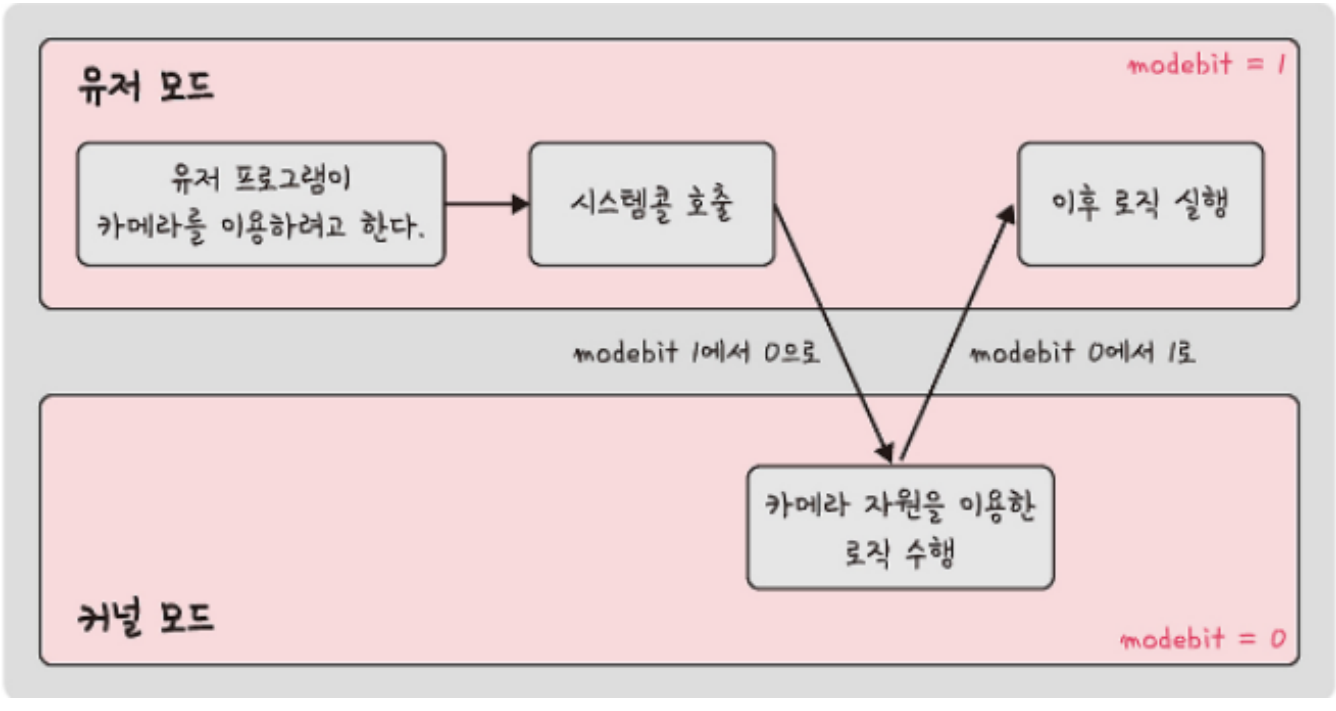


→ **시스템콜**

- 운영체제가 커널에 접근하기 위한 인터페이스ㄴ
- 유저 프로그램이 운영체제의 서비스를 받기 위해 커널 함수를 호출 할 때 사용
- 장점 : 데이터베이스와 같은 낮은 단계의 영역 처리에 대한 부분을 많이 신경 쓰지 않고 프로그램을 구현 할 수 있음

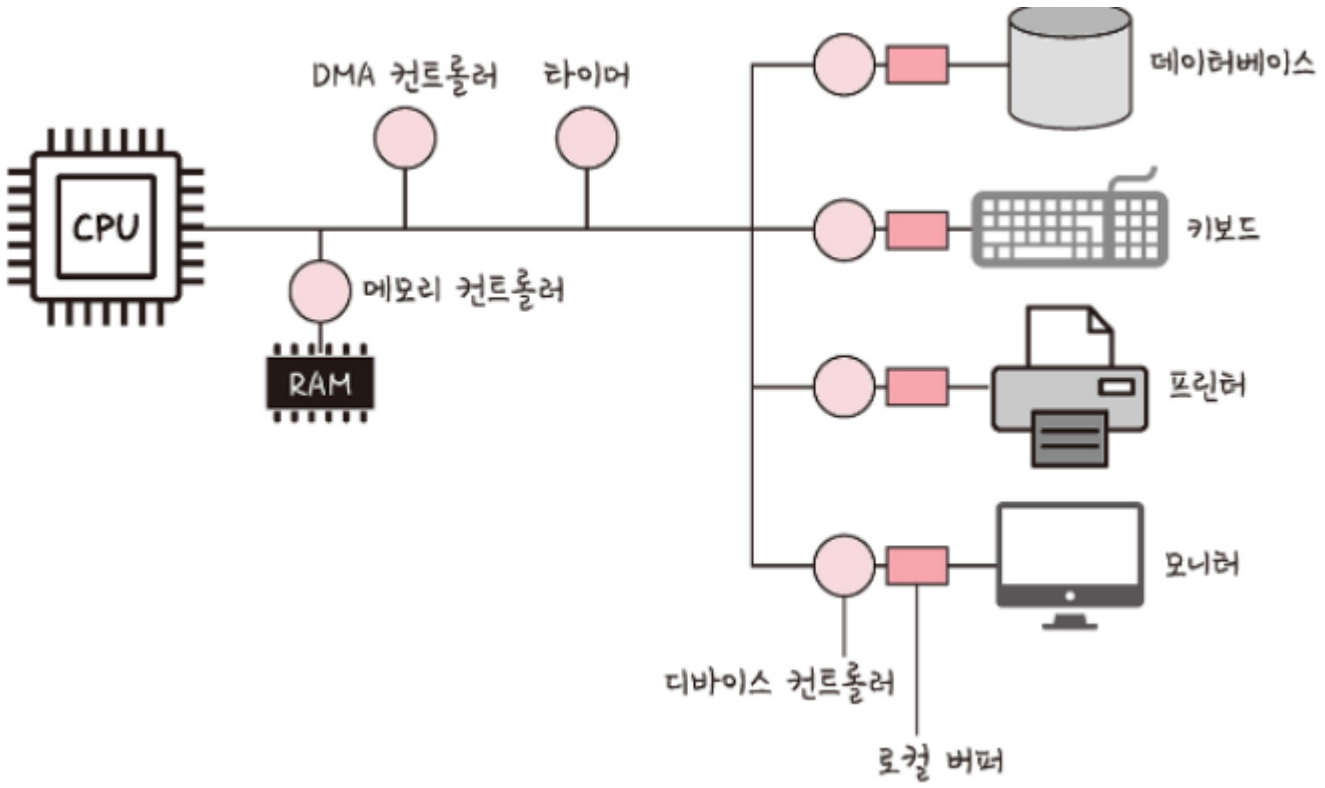


→ **modebit**



→ **컴퓨터의 요소**

- 컴퓨터는 CPU,DMA,컨트롤러,메모리,타이머,디바이스 컨트롤러로 구성 되어 있음



→ **CPU**

- 산술논리연산장치, 제어장치, 레지스터로 구성되어 있는 컴퓨터 장치이며 인터럽트에 의해 단순히 메모리에 존재하는 명령어를 해석해서 실행하는 일꾼

→ **제어장치**

- 프로세스 조작을 지시하는 CPU의 한 부품이며 입출력 장치 간 통신을 제어하고 명령어들을 읽고 해석하며 데이터 처리를 위한 순서를 결정 한다.

→ **레지스터**

- CPU 안에 있는 매우 빠른 임시기억장치를 가리키며 CPU와 직접 연결되어 있으므로 연산 속도가 메모리보다 수십 배에서 수백 배까지 빠르다.

→ **산술논리연산장치**

- 덧셈, 뺄셈, 같은 두 숫자의 산술 연산과 배타적 논리합, 논리곱 같은 논리 연산을 계산하는 디지털 회로 입니다.

→ **인터럽트**

- 어떤 신호가 들어왔을 때 CPU를 잠깐 정지 시키는 것. 키보드, 마우스 등 I/O 디바이스로 인한 인터럽트 발생

→ **DMA 컨트롤러**

- DMA 컨트롤러는 I/O 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치, CPU의 부하를 막아주며 CPU의 일을 부담하는 보조 일꾼

→ **메모리**

- 전자회로에서 데어터나 상태, 명령어 등을 기록하는 장치를 말하며, 보통 RAM을 일컬어 메모리 라고도 합니다. CPU는 계산을 담당하고, 메모리는 기억을 담당한다.

→ **타이머**

- 몇 초안에는 작업이 끝나야 한다는 것을 정하고 특정 프로그램에 시간 제한을 다는 역할

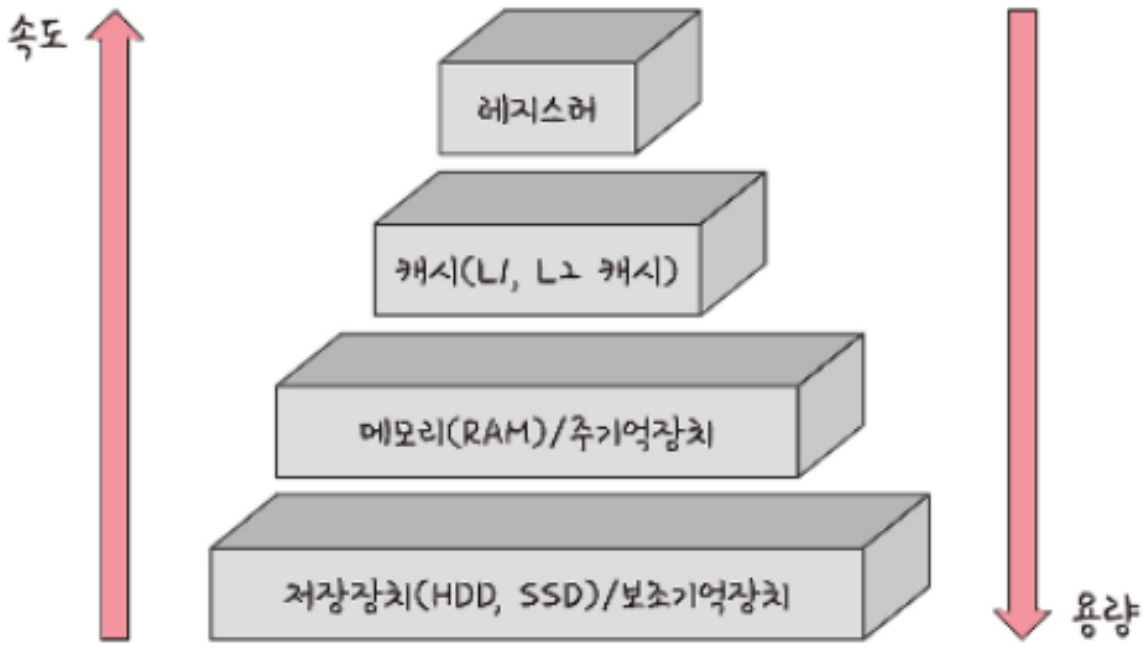
→ **디바이스 컨트롤러**

- 컴퓨터와 연결되어 있는 IO 디바이스들의 작은 CPU를 말하고 옆에 붙어 있는 로컬 버퍼는 각 디바이스에서 데이터를 임시로 저장하기 위한 작은 메모리를 뜻합니다.

2. 메모리

→ **메모리 계층**

- 메모리 계층은 레지스터, 캐시, 메모리, 저장장치로 구성되어 있다.



→ **레지스터**

- CPU 안에 있는 작은 메모리, 휘발성, 속도 가장 빠름, 기억 용량이 가장 적음

→ **캐시**

- 휘발성, 속도 빠름, 기억 용량이 적다. 참고로 L3 캐시도 있다.

→ **주기억장치**

- RAM을 가리킨다. 휘발성, 속도 보통, 기억 용량이 보통

→ **보조기억장치**

- HDD, SSD를 일컬으며 비휘발성, 속도 낮음, 기억 용량이 많다.

→ **지역성 원리**

- 자주 사용하는 데이터의 근거 : 시간 지역성과 공간 지역성

→ **시간 지역성**

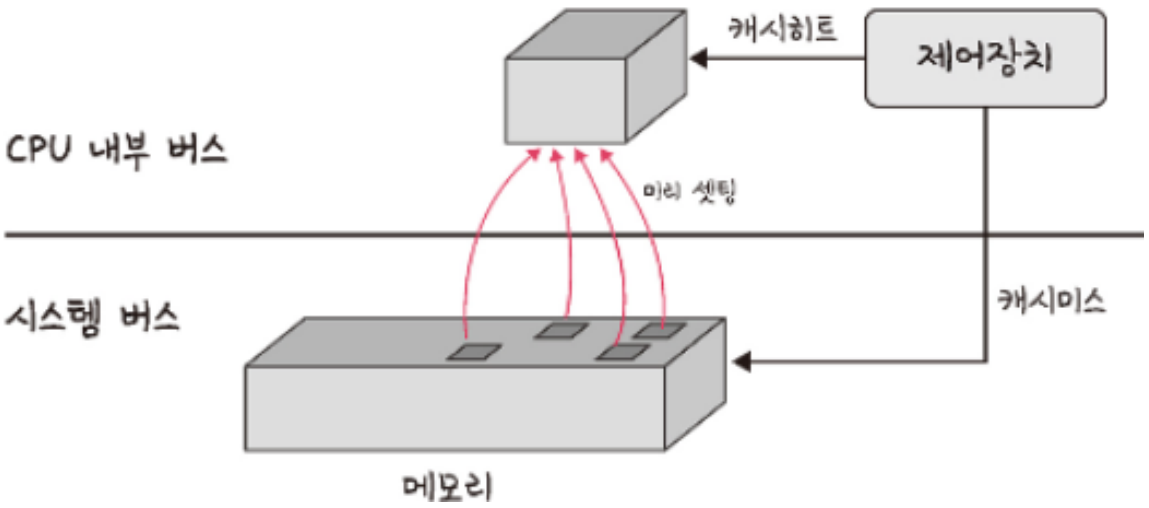
- 최근 사용한 데이터에 다시 접근하려는 특성을 말한다. ex) for문의 int i

→ **공간 지역성**

- 최근 접근한 데이터를 이루고 있는 공간이나 그 가까운 공간에 접근하는 특성을 말한다.

→ **캐시히트와 캐시미스**

- 캐시에서 원하는 데이터를 찾았다면 캐시히트
- 해당 데이터가 캐시에 없다면 주 메모리로 가서 데이터를 찾아가는 것이 캐시미스



→ **캐시매핑**

- 캐시매핑이란 캐시가 히트되기 위해 매핑하는 방법 CPU의 레지스터와 주 메모리 간에 데이터를 주고 받을 때를 기반으로 설명한다.
  - 직접 매핑 : 메모리가 1~100 캐시가 1~10이면 1:1~10, 2:1~20... 매핑하는 방법
  - 연관 매핑 : 순서를 일치시키지 않고 관련 있는 캐시와 메모리를 매핑
  - 집합 연관 매핑 : 직접 매핑과 연관 매핑을 합쳐 놓은 것

→ **웹 브라우저의 캐시**

- 소프트웨어적인 대표적인 캐시로는 웹 브라우저의 작은 저장소 쿠키, 로컬 스토리지, 세션 스토리지가 있다.

→ **쿠키**

- 쿠키는 만료기한이 있는 키-값 저장소

→ **로컬 스토리지**

- 로컬 스토리지는 만료기한이 없는 키-값 저장소
- 웹 브라우저를 닫아도 유지됨

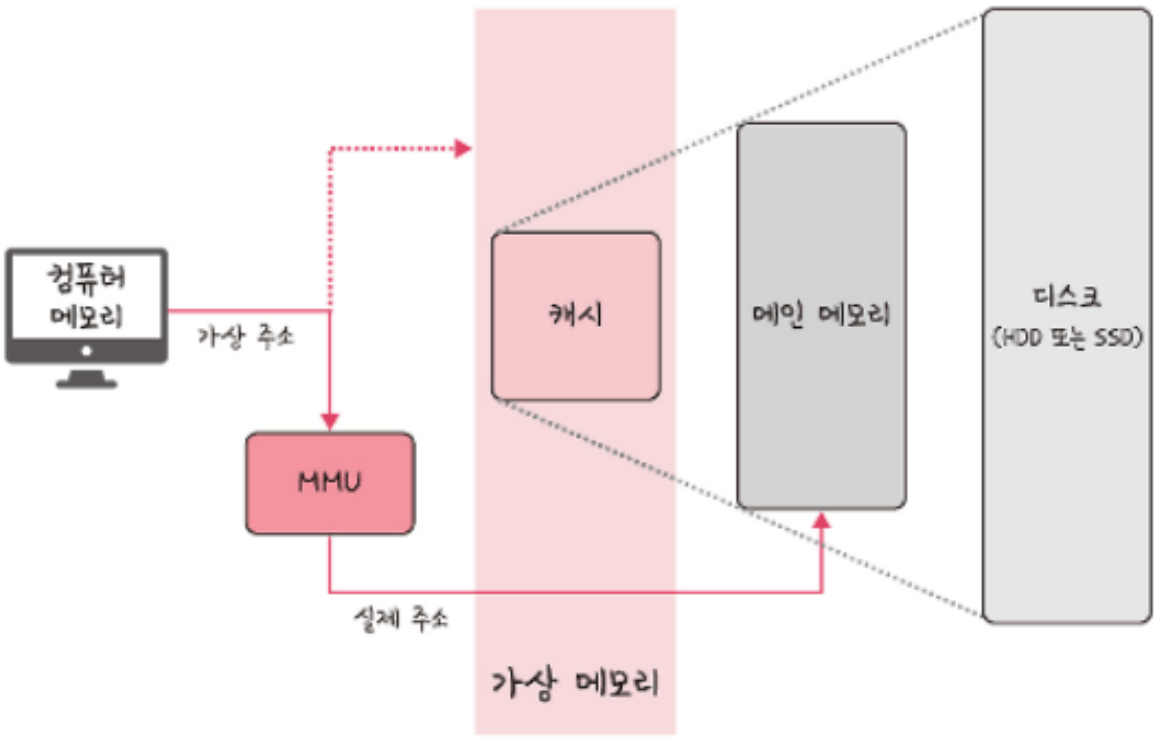
→ **세션 스토리지**

- 세션 스토리지는 만료기한이 없는 키-값 저장소 입니다.
- 탭 단위로 세션 스토리지를 생성하며, 탭을 닫을 때 해당 데이터가 삭제 된다.

2.2 메모리 관리

→ **가상 메모리**

- 메모리 관리 기법의 하나로 컴퓨터가 실제로 이용 가능한 메모리 자원을 추상화하여 이를 사용하는 사용자들에게 매우 큰 메모리로 보이게 만드는 것



- 가상적으로 주어진 주소를 가상 주소라고 하며, 실제 메모리상에 있는 주소를 실제 주소라고 합니다. 가상 주소는 메모리관리장치에 의해 실제주소로 반환
- 가상 메모리는 가상 주소와 실제 주소가 매핑되어 있고 프로세스의 주소 정보가 들어 있는 '페이지 테이블'로 관리된다. 이때 속도 향상을 위해 TLB를 쓴다.

→ **가상 메모리**

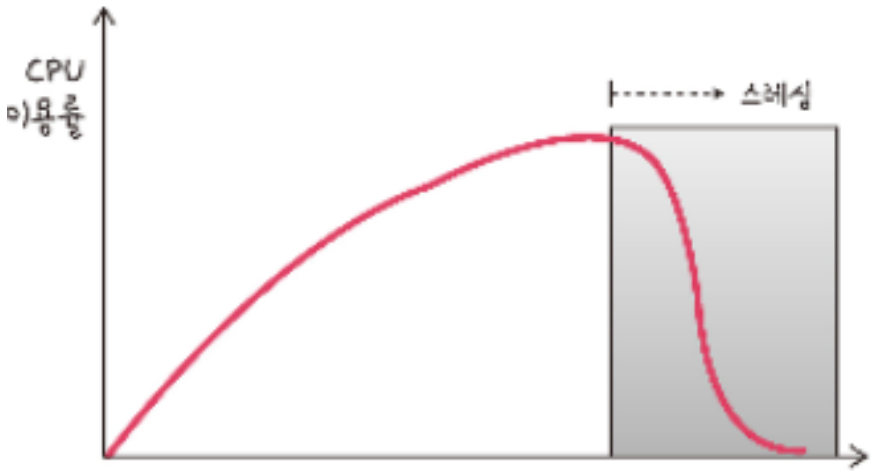
- 만약 가상 메모리에는 존재하지만 실제 메모리인 RAM에는 현재 없는 데이터나 코드에 접근할 경우 페이지 폴트가 발생
- 메모리에서 당장 사용하지 안는 영역을 하드디스크로 옮기고 하드디스크의 일부분을 마치 메모리처럼 불러와 쓰는 것을 스와핑 이라고 한다.

→ **페이지 폴트**

- 프로세스의 주소 공간에는 존재하지만 지금 이 컴퓨터의 RAM에는 없는 데이터에 접근했을 경우에 발생

→ **스레싱**

- 메모리의 페이지 폴트율이 높은 것을 의미, 심각한 성능 저하 초래



→ **메모리 할당**

- 메모리에 프로그램을 할당할 때는 시작 메모리 위치, 메모리의 할당 크기를 기반으로 할당하는데, 연속 할당과 불연속 할당으로 나뉜다.

→ **연속 할당**

- 연속 할당은 메모리에 연속적으로 공가을 할당하는 것

→ **고정 분할 방식**

- 메모리를 미리 나누어 관리하는 방식, 메모리가 미리 나뉘어 있기 때문에 융통성이 없고 내부 단편화가 발생

→ **가변 분할 방식**

- 매 시점 프로그램의 크기에 맞게 동적으로 메모리를 나눠 사용한다.
- 내부 단편화는 발생하지 않고 외부 단편화는 발생할 수 있다.
    - 최초적합
    - 최적적합
    - 최악적합

→ **불연속 할당**

- 메모리를 연속적으로 할당하지 않는 불연속 할당은 현대 운영체제가 쓰는 방법(페이징 기법)

→ **페이징**

- 동일한 크기의 페이지 단위로 나누어 메모리의 서로 다른 위치에 프로세스를 할당한다.

→ **세그멘테이션**

- 페이지 단위가 아닌 의미 단위인 세그먼트로 나누는 방법

→ **페이지드 세그멘테이션**

- 프로그램을 의미 단위인 세그먼트ㄹ 나눠 공유나 보안 측면에 강점을 두고 임의의 길이가 아닌 동일한 크기의 페이지 단위로 나누는 것을 말한다.

→ **페이지 교체 알고리즘**

- 메모리는 한정되어 있기 때문에 스와핑이 많이 일어난다.
- 스와핑은 많이 일어나지 않도록 설계되어야 하며 이는 페이지 교체 알고리즘을 기반으로 스와핑이 일어난다.
    - 오프라인 알고리즘 : 먼 미래에 참조되는 페이지와 현재 할당하는 페이지를 바꾸는 알고리즘
    - FIFO 알고리즘 : 가장 먼저 온 페이지를 교체 영역에 가장 먼저 놓는 방법을 의미
    - LRU 알고리즘 : 참조가 가장 오래된 페이지를 바꾸는 방법
    - NUR 알고리즘
    - LFU 알고리즘

3. 프로세스와 스레드

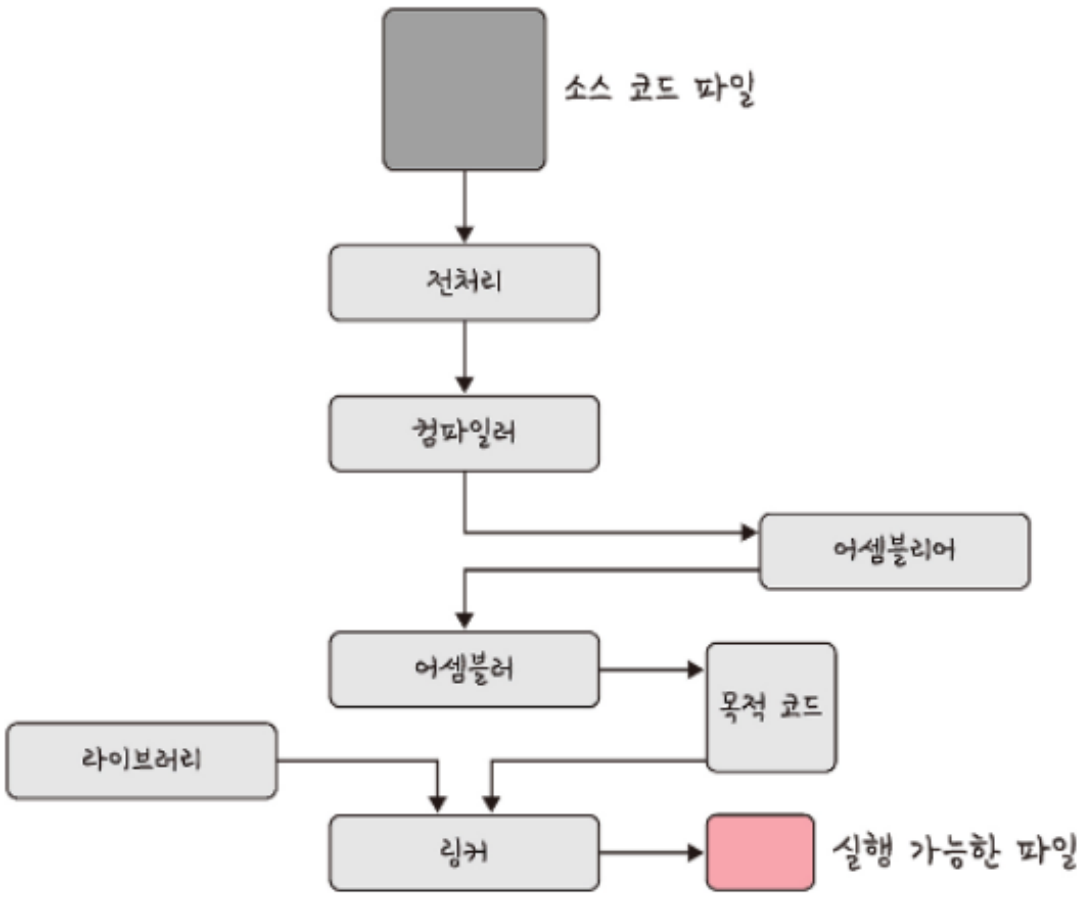
→ **프로세스**

- 컴퓨터에서 실행되고 있는프로그램을 말하며 CPU 스케줄링의 대상이 되는 작업

→ **스레드**

- 프로세스 내 작업의 흐름을 자칭

→ **프로세스와 컴파일 과정**



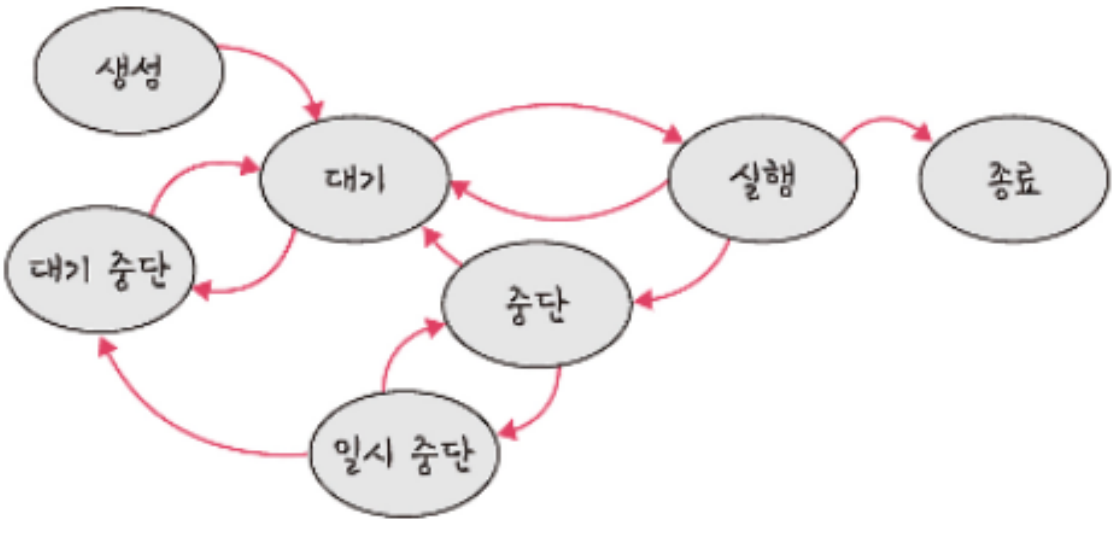
→ **어셈블러**

- 목적 코드로 변환 리눅스 → .o

→ **링커**

- 프로그램 내에 있는 라이브러리 함수 또는 다른 파일들과 목적 코드를 결합하여 실행 파일 생성 확장자는 .exe or .out

→ **프로세스의 상태**



→ **생성 상태**

- 생성 상태는 프로세스가 생성된 상태를 의미하며 fork() 또는 exec() 함수를 통해 생성 한다.
  - for() : 부모 프로세스의 주소 공간을 그대로 복사하며, 새로운 자식 프로세스를 생성하는 함수
  - exec() : 새롭게 프로세스를 생성하는 함수

→ **대기상태**

- 대기 상태는 메모리 공간이 충분하면 메모리를 할당 받고 아니면 아닌 상태로 대기하고 있으며 CPU 스케줄러로 부터 CPU 소유권이 넘어오기를 기다리는 상태

→ **대기 중단 상태**

- 메모리 부족으로 일시 중단 된 상태

→ **실행 상태**

- CPU 소유권과 메모리를 할당 받고 인스트럭션을 수행 중인 상태(=CPU burst)

→ **중단 상태**

- 어떤 이벤트가 발생한 이후 기다리며 프로세스가 차단된 상태

→ **일시 중단 상태**

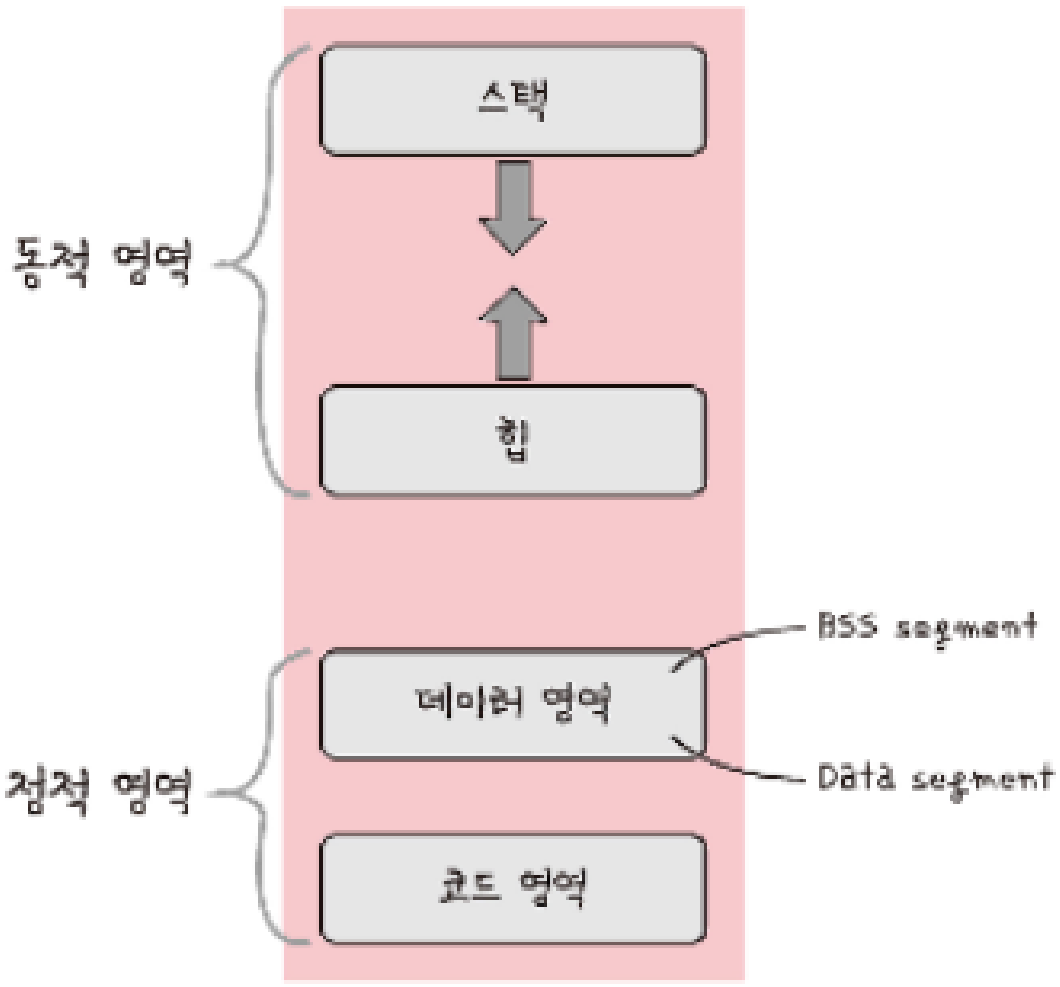
- 중단된 프로세스가 실행되려고 했지만 메모리 부족으로 인해 일시 중단된 상태

→ **종료 상태**

- 메모리와 CPU 소유권을 모두 놓고 가는 상태

→ **프로세스의 메모리 구조**

- 운영체제는 프로세스에 적절한 메모리를 할당하는데 다음 구조를 기반으로 할당

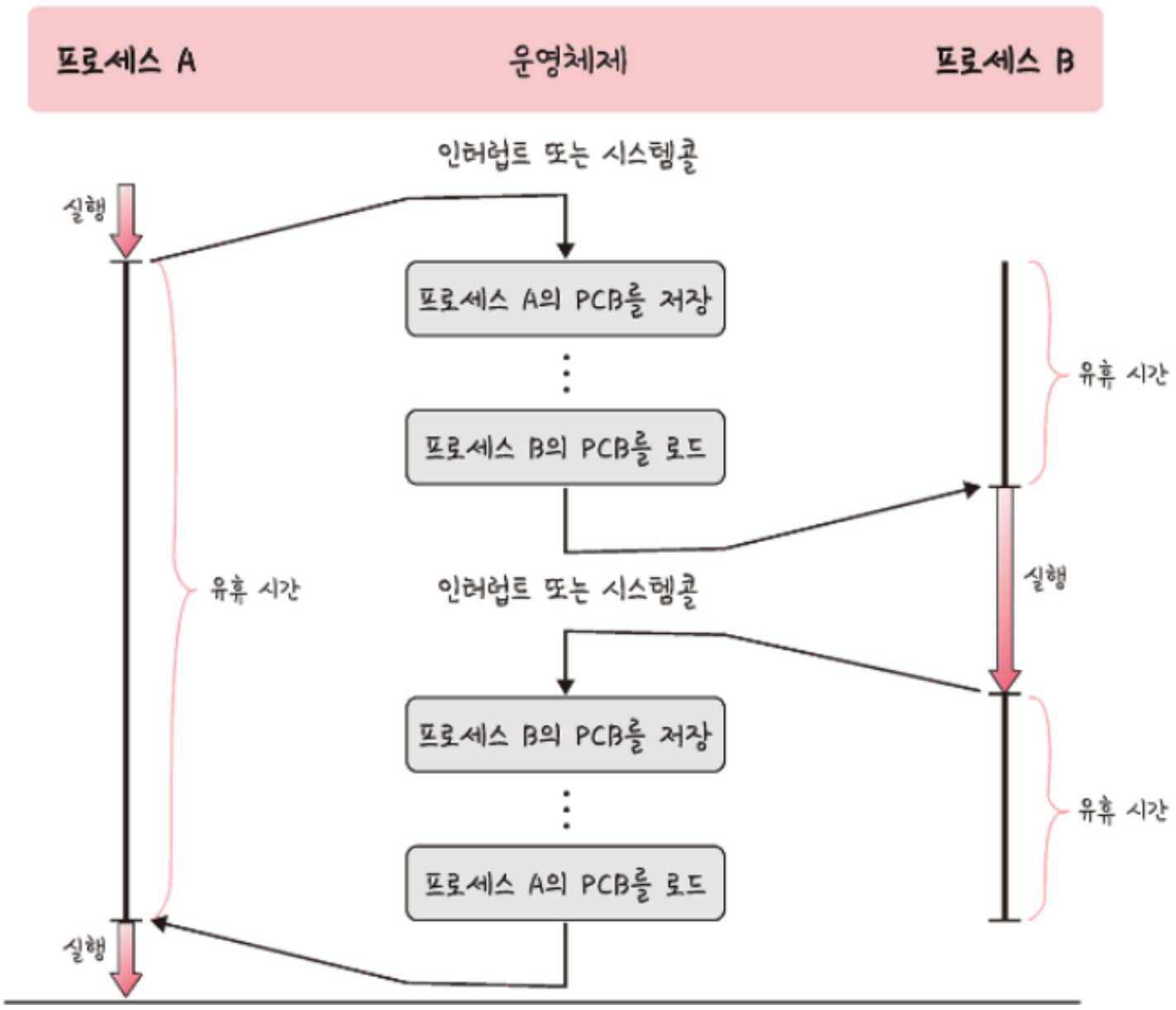


→ **PCB**

- 운영체제에서 프로세스에 대한 메타데이터를 저장한 '데이터'를 말한다.(=프로세스 제어블록)
- 구조
  - 프로세스 스케줄링 상태 :
  - 프로세스 ID
  - 프로세스 권한
  - 프로그램 카운터
  - CPU 레지스터
  - CPU 스케줄링 정보
  - 계정 정보
  - I/O 상태 정보

→ **컨텍스트 스위칭**

- 앞서 설명한 PCB를 기반으로 프로세스의 상태를 저장하고 로드시키는 과정을 말한다.



→ **IPC**

- IPC가 가능하며 IPC는 프로세스끼리 데이터를 주고받고 공유 데이터를 관리하는 메커니즘

→ **공유 메모리**

- 여러 프로세스에 동일한 메모리 블록에 대한 접근 권한이 부여되어 프로세스가 서로 통신 할 수 있도록 공유 메모리를 생성해서 통신하는 것

→ **파일**

- 디스크에 저장된 데이터 또는 파일 서버에서 제공한 데이터를 말한다.

→ **소켓**

- 동일한 컴퓨터의 다른 프로세스나 네트워크의 다른 컴퓨터로 네트워크 인터페이스를 통해 전송하는 데이터

→ **익명 파이프**

- 프로세스 간에 FIFO 방식으로 읽히는 임시 공간인 파이프를 기반으로 데이터를 주고 받으며, 단 방향 방식의 읽기 전용, 쓰기 전용 파이프를 만들어서 작동하는 방식

→ **명명된 파이프**

- 파이프 서버와 하나 이상의 파이프 클라이언트 간의 통신을 위한 명명된 단방향 또는 양방향 파이프

→ **메시지 큐**

- 메시지를 큐 데이터 구조 형태로 관리하는 것을 의미 다른 IPC 방식에 비해서 사용 방법이 매우 직관적이고 간단하다.

→ **멀티스레딩**

- 멀티스레딩은 프로세스 내 작업을 여러 개의 스레드, 멀티스레드로 처리하는 기법이며 스레드끼리 서로 자원을 공유하기 때문에 효율성이 높다.

→ **공유 자원**

- 시스템 안에서 각 프로세스, 스레드가 함께 접근할 수 있는 모니터, 프린터, 메모리, 파일, 데이터 등의 자원이나 변수 등을 의미
- 공유 자원을 두개 이상의 프로세스가 동시에 읽거나 쓰는 상황 → 경쟁 상태

→ **임계 영역**

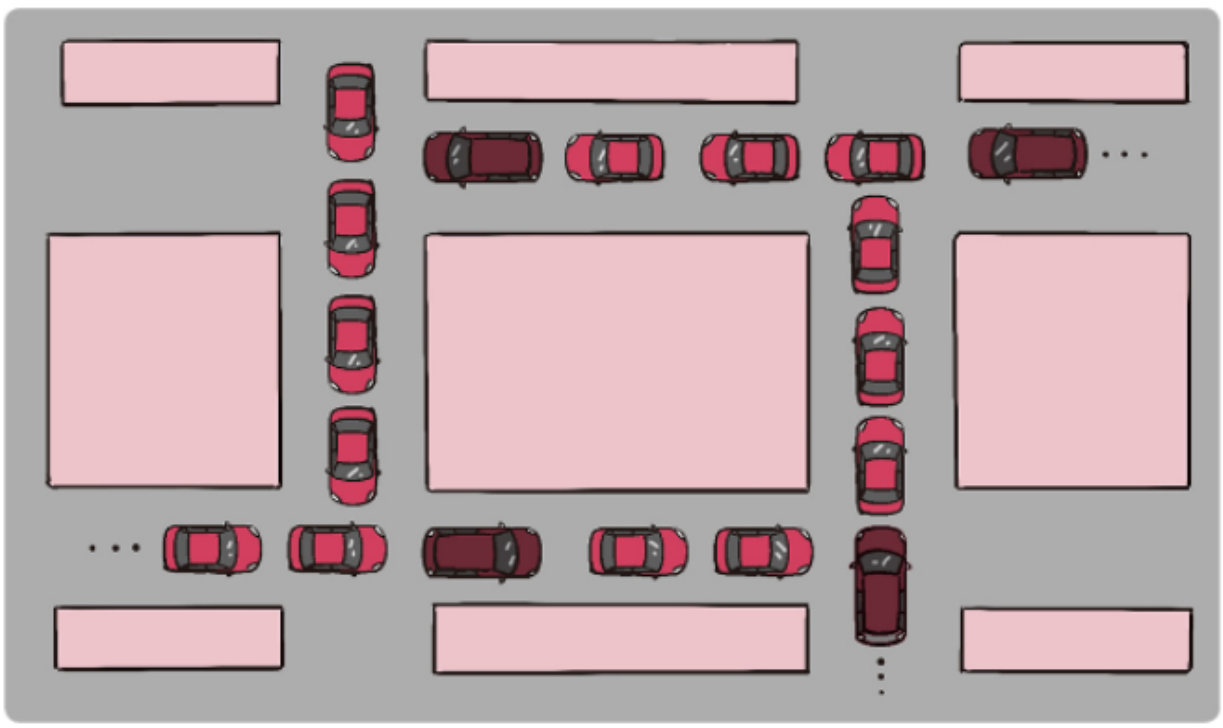
- 둘 이상의 프로세스, 스레드가 공유 자원에 접근할 때 순서 등의 이유로 결과가 달라지는 코드 영역
- 해결 방법 :
  - 뮤텍스 : 프로세스나 스레드가 공유 자원을 lock()을 통해 잠금을 설정하고 사용한 후에는 unlock()을 통해 잠금 해제
  - 세마포어 : 일반화된 뮤텍스, wait 및 signal로 공유 자원에 접근
  - 모니터 : 둘 이상의 스레드나 프로세스가 공유 자원에 안전하게 접근할 수 있도록 공유 자원을 숨기고 해당 접근에 대해 인터페이만 제공

→ **임계 영역의 3가지 조건**

- 상호 배제 : 한 프로세스가 임계 영역에 들어 갔을 때 다른 프로세스는 들어 갈 수 없다.
- 한정 대기 : 특정 프로세스가 영원히 임계 영역에 들어가지 못하면 안됨
- 융통성 : 어떠한 프로세스도 임계 영역을 사용하지 않는다면 임계 영역 외부의 어떠한 프로세스도 들어갈 수 있고 프로세스끼리 서로 방해하지 않는다.

→ **교착 상태**

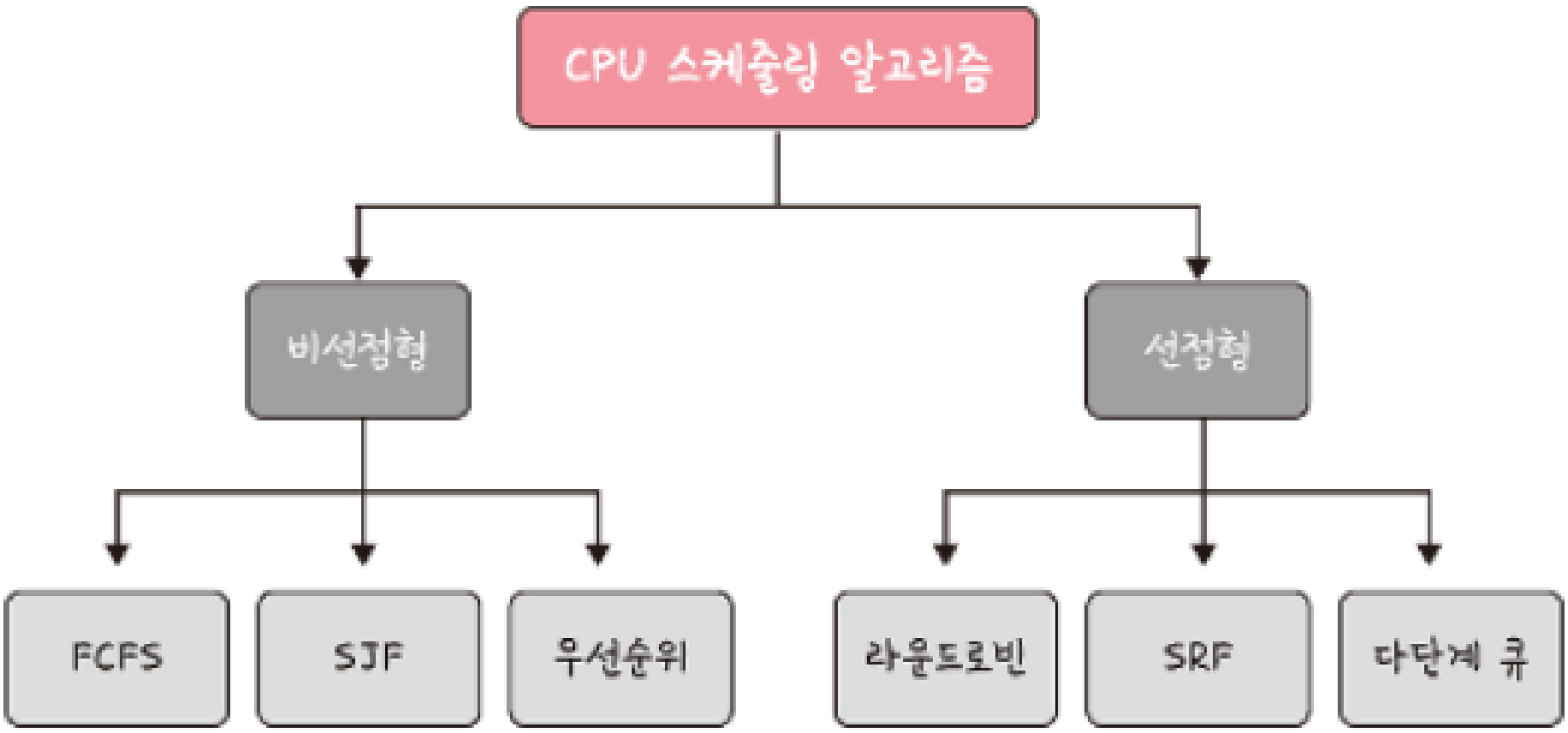
- 둘 이상의 프로세스들이 서로가 가진 자원을 기다리며 중단된 상태
- 교착 상태의 원인 :
  - 상호 배제
  - 점유 대기
  - 비선점
  - 환형 대기



→ **교착 상태 해결 방법**

1. 자원을 할당 할 때 애초에 조건이 성립되지 않도록 설계
2. 교착 상태 가능성이 없을 때만 자원 할당되며, 프로세스당 요청할 자원들의 최대치를 통해 자원 할당 가능 여부를 파악하는 '은행원 알고리즘' 사용
3. 교착 상태가 발생하면 사이클이 있는지 찾아보고 이에 관련된 프로세스를 한 개씩 지운다.
4. 교착 상태는 매우 드물게 일어나기 때문에 이를 처리하는 비용이 더 커서 교착 상태가 발생하면 사용자가 작업 종료

3.4 CPU 스케줄링 알고리즘



→ **비선점형 방식**

- 프로세스가 스스로 CPU 소유권을 포기하는 방식, 강제로 프로세스를 중지 하지 않는다.

→ **FCFS**

- 가장 먼저 온 것을 가장 먼저 처리하는 알고리즘 → convoy effect 발생 할 수 있음

→ **SJF**

- 실행 시간이 가장 짧은 프로세스를 가장 먼저 실행하는 알고리즘 → 긴 시간을 가진 프로세스가 실행되지 않는 현상 + 평균 대기 시간이 가장 짧음

→ **우선 순위**

- 오래된 작업일 수록 '우선순위를 높이는 방법'을 사용한 알고리즘

→ **선점형 방식**

- 현대 운영체제가 쓰는 방식으로 지금 사용하고 있는 프로세스를 알고리즘에 의해 중단시키고 강제로 다른 프로세스에 CPU 소유권을 할당하는 방식

→ **라운드 로빈**

- 현대 컴퓨터가 쓰는 섬점형 알고리즘 스케줄링 방법 각 프로세스는 동일한 할당 시간을 주고 그 시간 안에 끝나지 않으면 다시 준비 큐의 뒤로 가는 알고리즘

→ **SRF**

- 중간에 더 짧은 작업이 들어오면 수행하던 프로세를 중지하고 해당 프로세스를 수행하는 알고리즘

→ **다단계 큐**

- 우선순위에 따른 준비 큐를 여러개 사용하고 큐마다 라운드 로빈이나 FCFS 등 다른 스케줄링 알고리즘을 적용한 것