Process vs Thread

💡 Keyword

- **실행 단위**
  - 프로세스와 스레드를 포괄하는 개념
  - 한 cpu core에 한 순간에 한 개씩 적재되는 것
  - 즉, cpu core에서 실행하는 하나의 단위
  - 실행 단위는 프로세스일 때도 있고, 스레드일 때도 있다

- **(부연 설명이 없는) 프로세스**
  - 하나의 스레드만 가지고 있는 단일 스레드 프로세스

- **동시성**
  - 한 순간에 일어나는게 아니라, 짧은 전환으로 여러가지 일을 동시에 처리하는 것처럼 보이는 것

💡 Process & Thread

- **Program & Process**
  - 피자 집에 가서 피자를 시켰는데… 피자 레시피가 나왔다…
  - **프로그램** : 여기서 피자 레시피는 우리가 만든 코드 파일
  - **프로세스** : 프로그램을 실행시켜서 얻은 결과물

- **프로그램이 프로세스가 되는 과정**
  1. 프로세스가 필요로 하는 재료들이 메모리에 올라가야 한다.

→ 메모리는 4가지 영역으로 나뉘어져 있다.

- **Code** : 실행 명령을 포함하는 코드들
- **Data** : Static 변수 혹은 Global 변수
- **Heap** : 동적 메모리 영역
- **Stack** : 지역변수, 매개변수, 반환 값 등의 일시적인 데이터

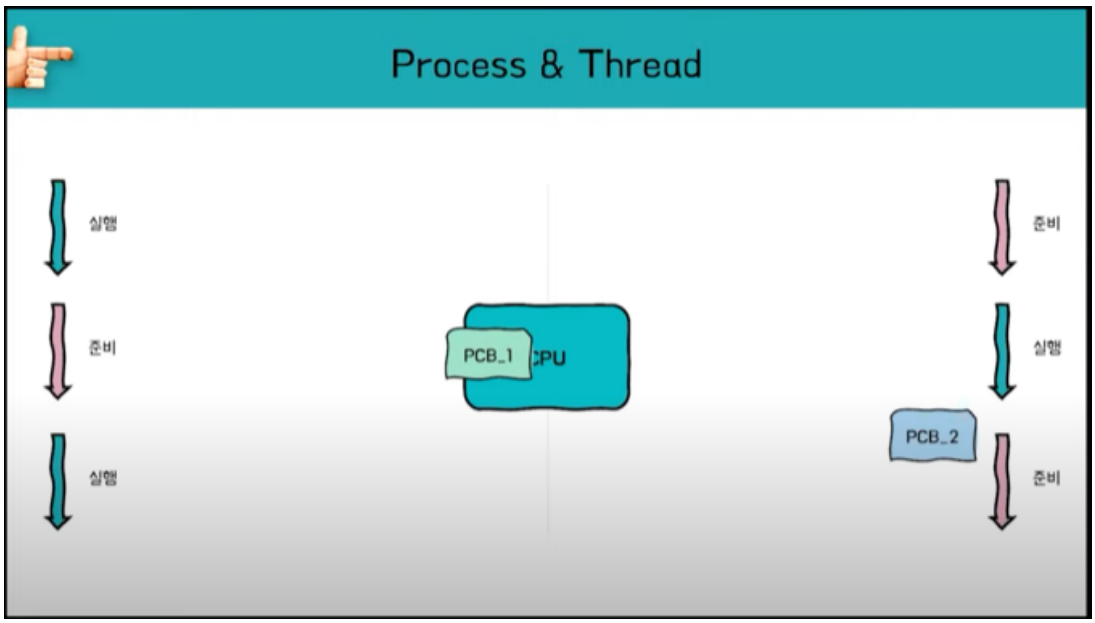
2. 해당 프로세스에 대한 정보를 담고 있는 **PCB(Process Control Block)** 가 프로세스 생성시 함께 만들어진다.

   → 안에 담긴 데이터가 너무 많아서 간단하게 몇 개만 말해보면

   - Pointer : 프로세스의 현재 위치를 저장
   - Process State : 생성, 준비, 실행, 대기, 완료 중 어떤 상태인지 저장한다
   - Process Number(ID) : PID 라는 고유 프로세스 아이디가 할당된다.
   - Program Counter : 다음 명령어 주소를 갖는다
   - 등등등등…..

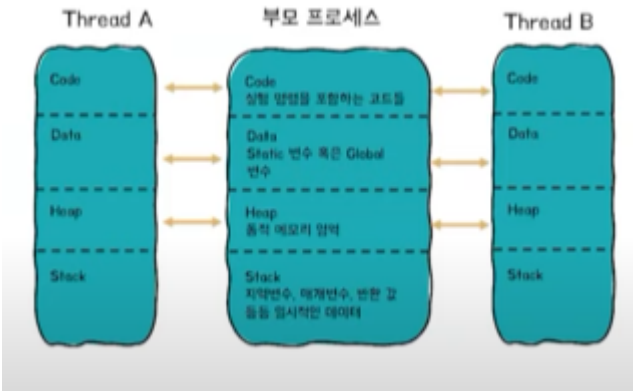
- **Process & Thread**
  - 들어가기 전에 예시를 하나 들어봅시다.
  - 회사에 출근을 하면 노래를 듣기 위해 유튜브 뮤직을 켜고, 코딩하려고 VSC 켜고, 카톡 켜고, 팀즈 켜고, 크롬 켜고 등등 참 많은 **프로세스들**을 한 번에 켜놓는데…
  - 하지만 원래 한 프로세스가 실행되기 위해 CPU를 점유하고 있으면 **다른 프로세스들은 실행 상태에 있을 수가 없다!!**
  - 유튜브 뮤직에서 노래 듣다가 VSC 켜는 순간 원래 노래가 꺼져야돼…
  - 근데 우리는 한번에 다 하고 있잖아요?
  - 이게 **동시성**!
  - 한 번에 다 실행시키는 것처럼 보이게 하기 위해 짧은 시분할로 나눠 여러개의 프로세스를 실행시킨다.

- 예를 들어 Process_1, Process_2 가 동시에 실행되고 싶다면 아래 사진과 같이 1번이 실행일 때, 2번이 준비 상태로 가고, 1번이 준비 상태로 가면 2번이 실행되고를 계속 빠르게 반복하는 것
- 이것을 **컨테스트 스위칭** 이라고 합니다.



- 근데… 2개만 해도 참 엄청나게 왔다갔다 하는데 프로세스가 계속 늘어나면 얼마나….
- 그래서 등장을 한게 경량화된 프로세스 버전인 **스레드!!**

- 다음과 같이 한 프로세스에 여러개의 스레드가 있다고 하자

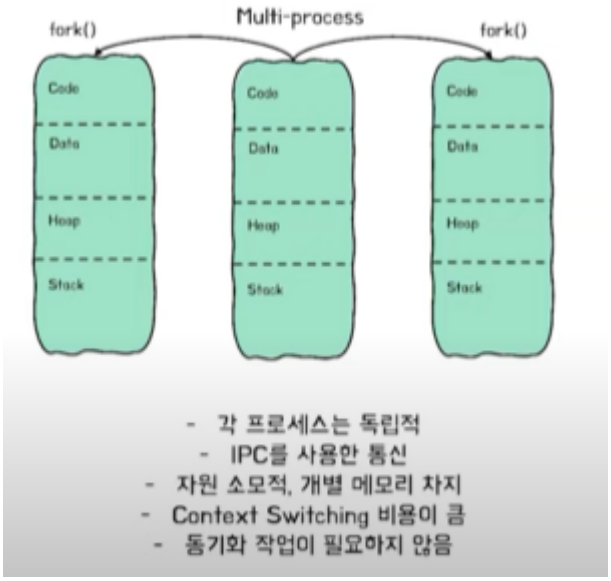


    - 각 스레드는 스택 부분만을 따로 가지고 있고 Code, Data, Heap 영역을 공통된 자원으로 사용합니다.
    - 공유되는 자원이 있기 때문에 컨텍스트 스위칭이 일어날 때 캐싱 적중률이 올라가는데 이는 그냥 필요한 부분만 넣었다 뺐다 하면 된다는 것!

    - 예를 들어 회의실 예약을 생각해보겠습니다.
    - 1팀이 먼저 회의실을 사용하고, 2팀이 1팀이 끝난 후 회의실을 사용합니다.
    - 근데 1팀이 회의실을 쓰고 안에 있던 공용 모니터, 빔프로젝터, 마이크 등등을 싹 다 갖고 나가버렸어요. 그러면 2팀은 그것들을 다시 다 가지고 와야겠고 이는 매우 비효율적
    - 각 팀들은 개인 노트북만 따로따로 가져오고 공용은 그대로 회의실에 둬야겠죠
    - 여기서 개인 노트북이 Stack, 공용으로 갖고 있는 것들이 Code, Data, Heap 입니다.
    - 훨씬 효율적입니다.

- **Multi-process & Multi-thread**
    - 두 가지 개념 모두 **처리 방식의 일종**

    1. **멀티 프로세스**
        - 여러 명의 사용자가 한 번에 로그인을 한다고 생각해보자
        - 프로세스는 한 번에 하나의 사용자만 로그인 시킬 수 있기 때문에, 프로세스는 fork()를 이용해 여러 개로 나뉘어진다.
        - 이때, 각각의 프로세스는 독립적이고 그로 인해 개별 메모리를 차지한다.

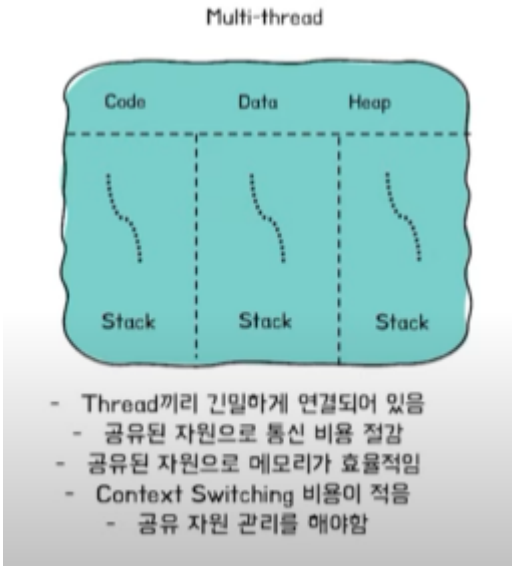


- 위에서 볼 수 있듯이 각 프로세스가 독립적이라서 가지는 여러 내용들이 특징이다. (좀만 생각해보면 당연한 내용들이다.)
- IPC(Inter-Process Communication)를 사용해 각 프로세스들이 통신을 한다고 했는데 간단하게 두 명의 개발자가 서로 다른 사무실에서 일하고 있다가 논의할 일이 생기면 옥상으로 가서 담배 한대 피면서 논의하고 다시 내려와서 작업하고 이런 느낌이다.

2. **멀티 스레드**
    - VSC라는 하나의 프로세스를 사용해서 코딩중이라고 생각해보자.
    - 내가 아래와 같은 코드를 치고 있다.

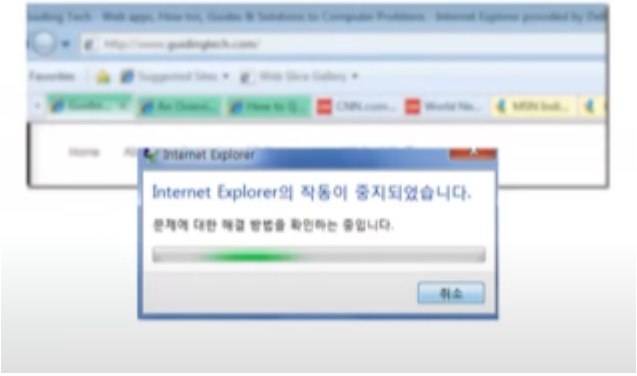
```
let testData: string[] = ["abc", "bcd", ...]

testData.map(() => { ... })
```

    - testData. 을 치자마자 여러 개의 추천 코드들을 보여준다.
    - 즉, 코드를 치는 동작과 추천 코드를 보여주는 동작이 동시에 일어나고 있는 것이다.
    - 이때 사용되는게 **멀티 스레드**!



- 멀티 스레드는 한 프로세스 내에서 공유 자원을 가지고 각자의 스택만 가지는 구조이기 때문에 각 스레드는 위와 같은 당연한 특징들을 가지고 있다.
- 그냥 한 사무실 내에서 여러 개발자가 코딩하면서 논의를 할 수 있는 것이다.
- 물론, 모든 개발자들은 동일한 깃헙 코드를 가지고 개발을 진행하고 있기 때문에 이런 공유 자원에 대한 관리는 철저히 해야합니다.

- 이렇게 보면… 멀티 스레드가 훠어어얼씬 좋아보이는데 왜 멀티 프로세스를 쓸까?
- 가장 좋은 예시는 IE와 크롬의 멀티탭이다.
- 예전 IE가 잘나갈때를 생각해보면… 한 개의 탭에만 문제가 생겨도 걍 다 에러 뜨고 다 껐어야 됐다.



- 개극혐이었다.

- 반면, 크롬은 각각의 탭이 서로에게 영향을 주지 않기 때문에 하나가 에러가 뜨면 그 탭만 끄면 된다.
- 이게 멀티 프로세스와 멀티 스레드의 차이이다.
- 크롬은 탭 설계가 멀티 프로세스로 이루어져 있는 반면 IE는 멀티 스레드 방식으로 탭을 설계했는데 멀티 스레드 방식은 공유 자원이 너무나 많다 보니 하나만 무너져도 와르르 무너졌다.

- **Multi-core**
  - 멀티 프로세스/스레드와 달리 처리 방식의 느낌이 아니다.
  - 멀티 코어는 좀 더 **하드웨어적**이다.
  - 멀티 코어와 관련된 키워드는 **동시성**과 **병렬처리**이다.
  - CPU 하나에 코어가 하나밖에 없다면 하나의 코어에서 하나 이상의 프로세스(혹은 스레드)가 번갈아가면서 진행되어야 한다.
  - 하지만 CPU에 코어가 여러개라면 둘 이상의 코어에서 동시에 하나 이상의 프로세스가 한꺼번에 진행될 수 있다.