😀

챕터 10. 프로세스와 스레드

```
// 챕터 목표
프로세스 제어 블록이 무엇인지 이해
문맥 교환의 정의와 과정을 학습
프로세스가 메모리에 어떻게 배치되는지 학습
프로세스 상태와 프로세스 계층 구조 학습
스레드의 개념을 이해하고, 멀티프로세스와 멀티스레드 차이를 이해
```

10-1. 프로세스 개요

프로세스란?

- 보조기억장치에 저장된 프로그램이 메모리에 적재되고 실행되면 그 프로그램을 프로세스라고 한다.

→ 즉, 프로세스는 실행 중인 프로그램.

프로세스의 종류

- **포그라운드 프로세스 (Foreground Process)**
  - 사용자가 볼 수 있는 공간에서 실행되는 프로세스.
- **백그라운드 프로세스 (Background Process)**
  - 사용자가 보지 못하는 뒤편에서 실행되는 프로세스.
- **데몬(유닉스) 또는 서비스(윈도우)**
  - 백그라운드 프로세스의 일종.
  - 사용자와 상호 작용하지 않고, 그저 묵묵히 정해진 일만 수행하는 프로세스.

프로세스 제어 블록 (PCB)

- PCB - Process Control Block
- **메모리의 커널 영역**에 생성됨.
- 프로세스 생성 시 만들어지고, 실행이 끝나면 폐기됨.

→ 운영체제가 특정한 프로세스를 식별할 수 있도록, 식별에 필요한 정보를 저장하는 자료 구조.

프로세스 제어 블록의 목적

- 모든 프로세스는 실행을 위해 CPU 자원이 필요.
- 하지만, CPU 자원은 한정되어 있기에, 돌아가면서 사용해야 함.

→ 빠르게 번갈아 수행되는 프로세스의 수행순서를 관리하고, CPU를 비롯한 자원을 배분하는데 사용하는 것이 바로 **프로세스 제어블록**

프로세스 제어 블록에 저장되는 정보

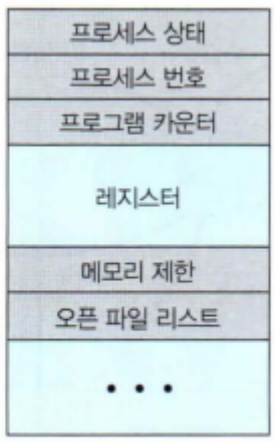


그림 3.3 프로세스 제어 블록(PCB)

- **프로세스 ID (PID)**
  - 프로세스를 식별하기 위한 고유 번호가 저장됨.
- **레지스터 값**
  - 프로세스는 자신의 실행 차례가 돌아올 때, 이전까지 사용한 레지스터의 중간 값들을 모두 복원해야 함.
  - 따라서, 해당 프로세스가 실행하며 사용했던 프로그램 카운터를 비롯한 레지스터 값들이 저장됨.
- **프로세스 상태**
  - 프로세스의 상태가 기록됨.
  - 입출력장치 사용을 위한 대기 상태 / CPU 사용을 위한 대기 상태 / CPU 이용 상태 등
- **CPU 스케줄링 정보**
  - 프로세스가 언제, 어떤 순서로 CPU 할당 받을 지에 대한 정보가 저장됨.
- **메모리 관리 정보**
  - 프로세스마다 메모리에 저장된 위치가 다름.
  - 프로세스가 어느 주소에 저장되어 있는 지에 대한 정보.
  - ex) 베이스 레지스터, 한계 레지스터 값, 페이지 테이블 정보와 같은 정보가 담김.
- **사용한 파일과 입출력 장치 목록**
  - 어떤 입출력장치가 이 프로세스에 할당되었는지, 어떤 파일을 열었는 지에 대한 정보가 저장됨.

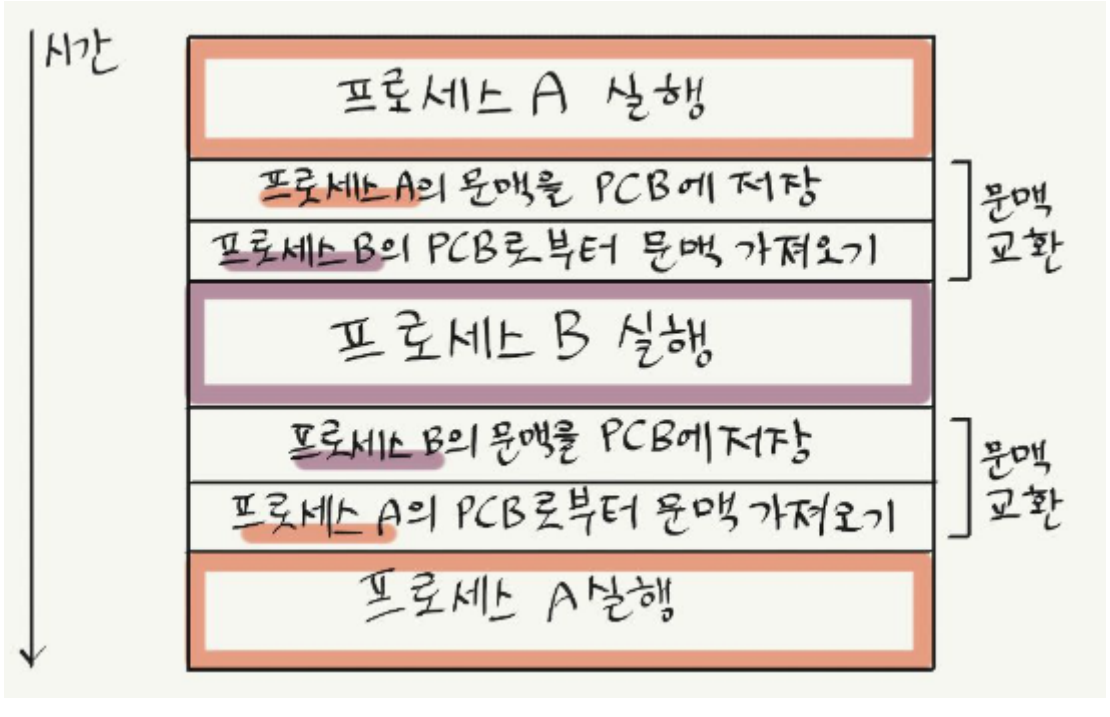
문맥 교환 (Context Switching)

문맥이란?

- CPU자원의 할당이 프로세스 A에서 B로 넘어간다고 가정.
- A는 다음 자신의 차례가 왔을 때 이전까지의 작업 내용부터 재개하기 위해, 프로그램 카운터를 비롯한 레지스터 값, 메모리 정보 등의 **중간 정보**를 백업해야 함.
- 이러한 중간 정보(문맥)은 프로세스의 PCB에 표현되어 있음.

→ **하나의 프로세스 수행을 재개하기 위해 기억해야 할 정보(중간 정보)가 바로 문맥(Context)**

문맥 교환이란?

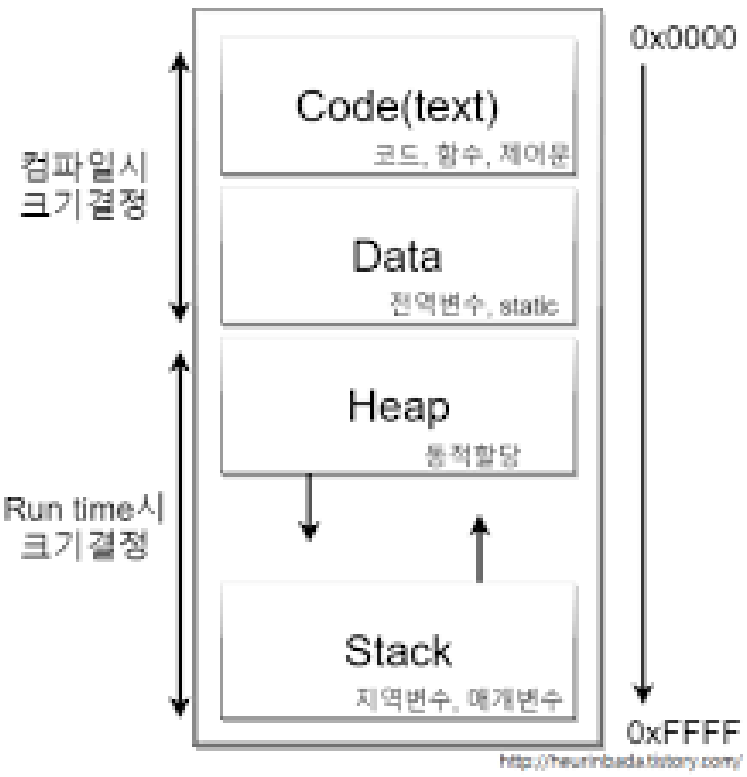


→ **<u>기존 프로세스의 문맥을 PCB에 백업</u>하고, <u>새로운 프로세스를 실행하기 위해 문맥을 PCB로부터 복구</u>하여 새로운 프로세스를 실행하는 것.**

- 여러 프로세스가 동시에 실행되는 것처럼 보이는 것은, 끊임없이 빠르게 번갈아가며 문맥 교환 하기 때문.
- 하지만, **문맥 교환**이 자주 일어나면 오버헤드가 발생해 성능 저하가 일어날 수 있다.

프로세스의 메모리 영역 **

- 프로세스 생성 시, 커널 영역에 PCB가 생성된다.
  - 그렇다면, 프로세스가 사용자 영역에는 어떻게 배치될까?
- 하나의 프로세스는 사용자 영역에 크게 4개의 영역으로 나뉘어 저장된다.
  - 코드 영역 / 데이터 영역 / 힙 영역 / 스택 영역



코드 영역

<u>→ 실행할 수 있는 코드, 즉 기계어로 이루어진 명령어가 저장되는 공간.</u>

- **정적 할당 영역**이다. (크기가 고정된 영역)
- 텍스트 영역이라 부르기도 함.
- 데이터가 아닌 CPU가 실행할 명령어가 저장되기에 읽기 전용. (쓰기 금지)

데이터 영역

→ 잠깐 썼다가 없앨 데이터가 아닌, 프로그램이 실행되는 동안 유지할 데이터가 저장되는 공간.

- **정적 할당 영역**이다. (크기가 고정된 영역)
- 전역변수가 대표적인 예시.

힙 영역

→ 프로그램을 만드는 사용자(프로그래머)가 직접 할당할 수 있는 저장 공간.

- 동**적 할당 영역**이다. (크기가 가변적인 영역)
- 추가로, 프로그래밍 과정에서 힙 영역에 메모리 공간을 할당했다면 반환해야 함.
  - 반환이란, 해당 공간을 더 이상 사용하지 않겠다고 말해주는 것과 같음.
  - 반환하지 않는다면, **메모리 누수 현상**이 일어남.

스택 영역

→ 데이터 영역과 달리, 잠깐 쓰다가 말 데이터를 일시적으로 저장하는 공간.

- 동**적 할당 영역**이다. (크기가 가변적인 영역)
- 함수가 끝나면 사라지는 매개변수나 지역변수가 대표적인 예시.
- 일시적으로 저장될 때는 스택에 push, 더 이상 필요하지 않을 때 pop

++ 힙 영역과 스택 영역

- 둘 다 동적 할당 영역.
- 일반적으로, **힙 영역은 낮은 주소 → 높은 주소**로 할당.
- 반대로, **스택 영역은 높은 주소 → 낮은 주소**로 할당.

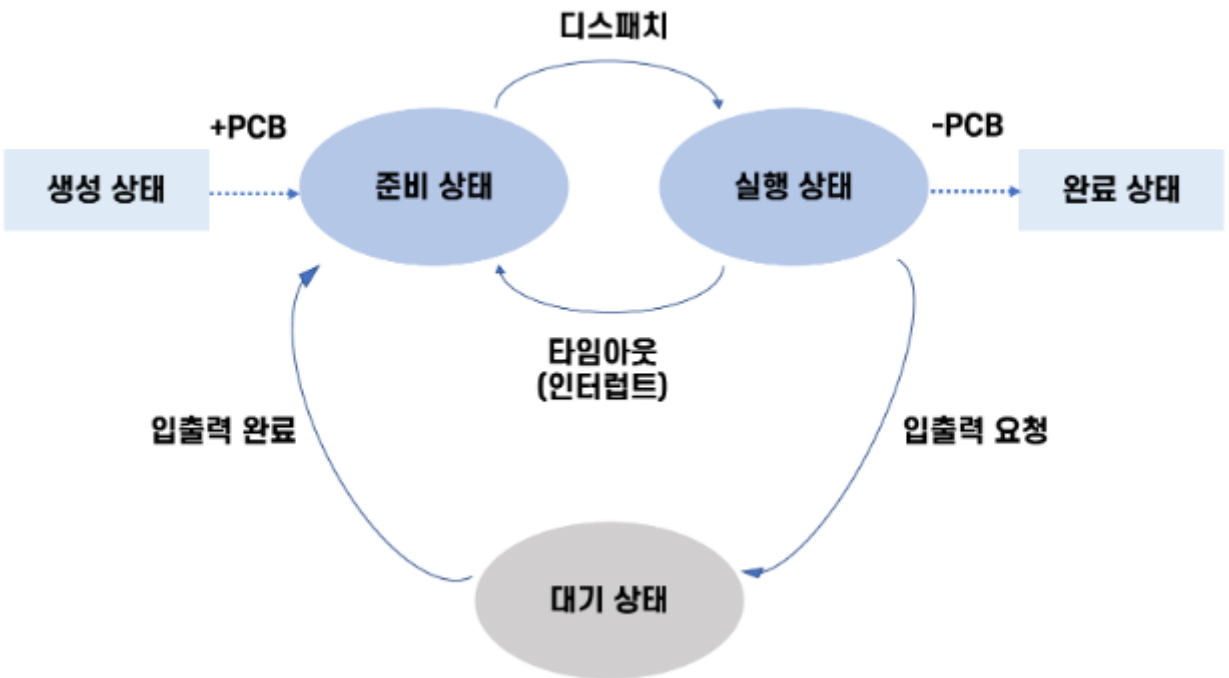
→ 따라서, 두 주소가 겹칠 가능성을 낮게 함.

10-2. 프로세스 상태와 계층 구조

프로세스 상태

- 여러 프로세스들이 빠르게 번갈아 가면서 실행되는데,
  그 과정에서, 하나의 프로세스는 여러 상태를 거치며 실행된다.
- 이러한 상태는 PCB를 통해 인식하고 관리한다.

프로세스 상태의 종류



- **생성 상태**

  → 프로세스를 생성 중인 상태.

  - 이제 막 메모리에 적재되어 PCB를 할당받은 상태를 뜻함.
- **준비 상태**

  → CPU를 자원을 할당받을 준비가 완료되었지만, 아직 자신의 차례가 아니어서 기다리는 상태
- **실행 상태**

  → CPU를 할당받아 실행 중인 상태.

  - 할당받은 시간 동안만 CPU 사용 가능하고, 할당된 시간을 모두 사용하면(타이머 인터럽트 발생 시) 다시 준비 상태가 된다.
- **대기 상태**

  → 프로세스 실행 중, 입출력장치 사용 시, 입출력 장치의 작업을 기다리는 상태.

  - 입출력장치가 입출력을 끝낼 때까지(입출력 완료 인터럽트를 받을 때까지) 대기 중인 상태를 뜻함.
    - **입출력 완료시에는 실행 상태가 아닌 준비 상태로 돌아감.**
- **종료 상태 (완료 상태)**

  → 프로세스가 종료된 상태.

  - 종료 상태일 때, 운영체제는 PCB와 프로세스가 사용한 메모리를 정리함.

프로세스 계층 구조

- 프로세스는 실행 도중 시스템 콜(시스템 호출)을 통해 다른 프로세스 생성 가능.
- **부모 프로세스**
  - 새 프로세스를 생성한 프로세스
- **자식 프로세스**
  - 부모 프로세스에 의해 생성된 프로세스
- 자식 프로세스는 실행 과정에서 또 다른 자식 프로세스를 생성할 수도 있음.
  - 해당 과정을 도표로 그리면 트리 형태를 띠게 됨.

- 이 때, 최초의 프로세스는 모든 프로세스의 가장 위에 있는 프로세스.

  ex) 유닉스의 init, 리눅스의 systemd, macOS의 launchd 등

프로세스 생성 기법(과정)

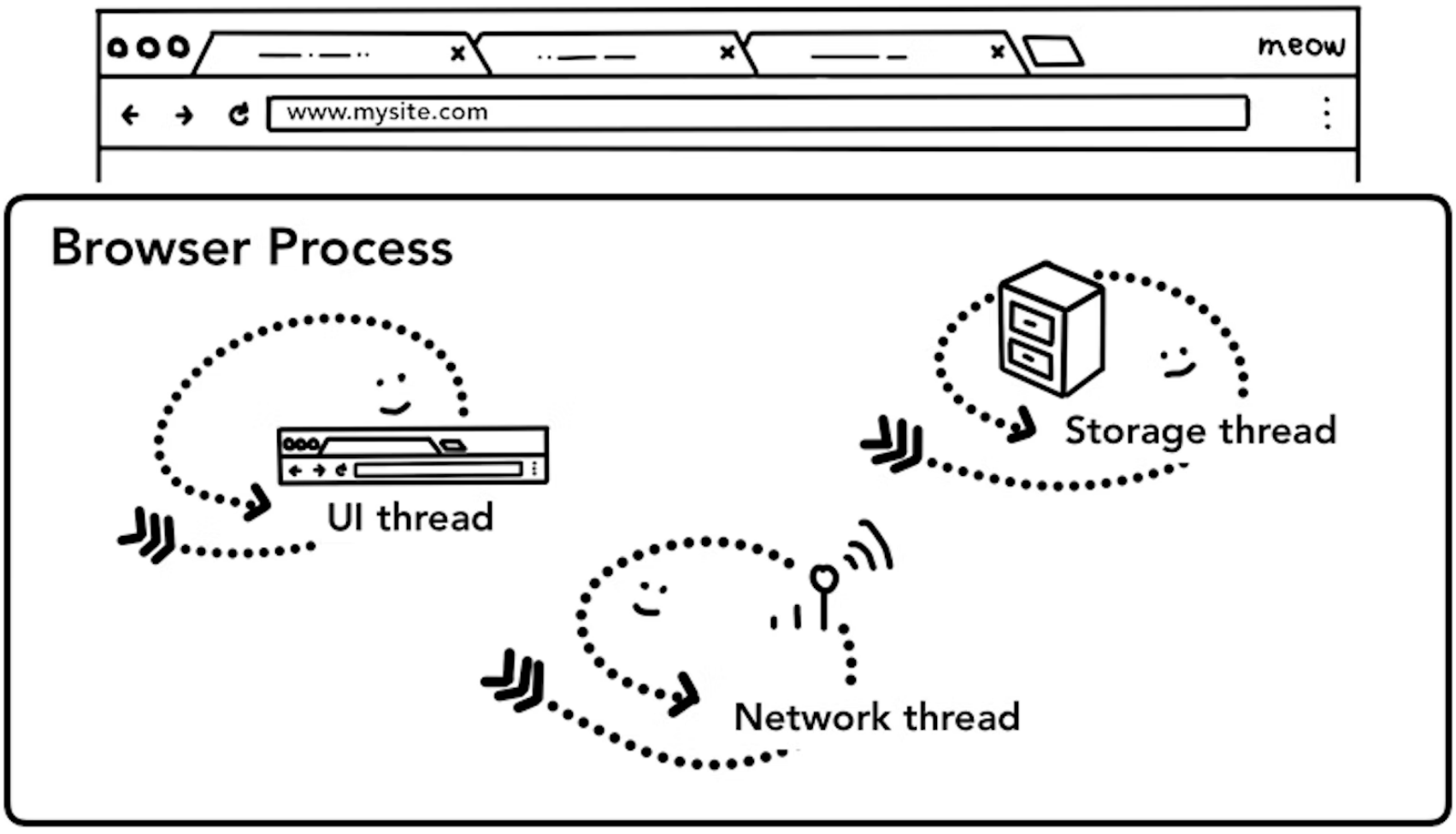
1. **부모 프로세스**가 `fork`를 통해 **자신의 복사본을 자식 프로세스로 생성**함.
2. **자식 프로세스**는 `exec`를 통해 **자신의 메모리 공간을 다른 프로그램으로 교체**함.

- 이 때, `fork`와 `exec`는 시스템 호출.
  - **fork** : 프로세스 자기 자신을 복제. 부모 프로세스의 자원, 메모리 내용, 열린 파일 목록 등이 복제본에 상속됨. **(단, 복제되었다고 해서, PID나 메모리 주소까지 같지는 않다.)**
  - **exec :** 자신의 메모리 공간을 새로운 프로그램으로 덮어쓰는 시스템 호출.
    코드 영역과 데이터 영역이 실행한 프로그램의 내용으로 바뀌고, 힙과 스택 영역은 초기화됨.

10-3. 스레드

스레드란?

- 프로세스를 구성하는 실행의 흐름 단위.
- 하나의 프로세스는 여러 개의 스레드를 가질 수 있다.
  - 이러한 스레드를 이용하면, 하나의 프로세스에서 여러 부분을 동시에 실행할 수 있다.



프로세스와 스레드

단일 스레드 프로세스

- 전통적인 관점에서, 하나의 프로세스는 한 번에 하나의 일만 처리.
- 즉, 프로세스 실행 흐름 단위(스레드)가 하나.

멀티 스레드 프로세스

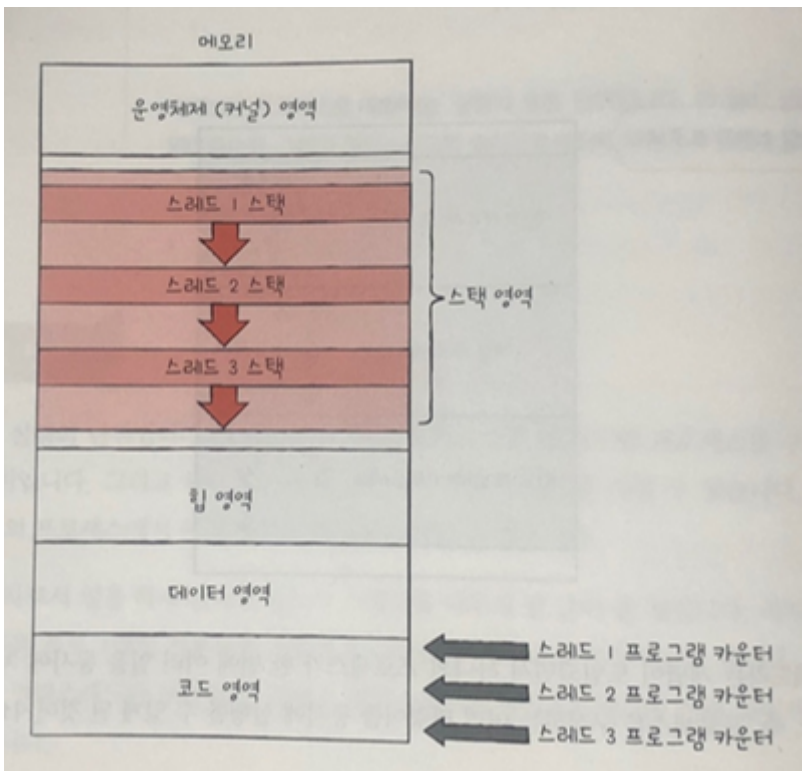
- 하나의 프로세스가 여러 일을 동시에 처리 가능.
- 즉, 프로세스를 구성하는 여러 명령어를 동시에 실행 가능.

스레드의 구성요소

- **스레드 ID**
- **프로그램 카운터 값을 비롯한 레지스터 값**
- **스택**

스레드의 특징

- 스레드의 구성요소와 같이, 각각의 레지스터 값, 스택을 가지고 있어서 스레드마다 각기 다른 코드 실행 가능.
- 실행에 필요한 최소한의 정보(프로그램 카운터를 포함한 레지스터 값, 스택)만을 유지한 채,
  프로세스 자원을 공유하며 실행됨.
    - 스레드 각각의 스택 영역은 별도로 존재하나, 코드/데이터/힙 영역이 별도로 있는 것은 아님.



멀티 프로세스와 멀티 스레드

멀티 프로세스

- 여러 프로세스를 동시에 실행하는 것

멀티 스레드

- 한 프로세스 내에서, 여러 스레드로 프로세스를 동시에 실행하는 것.

멀티 프로세스 vs 멀티 스레드

- **자원 공유**
  - **프로세스**끼리는 기본적으로 자원 공유를 하지 않지만,
    스레드끼리는 같은 프로세스 내의 자원을 공유할 수 있다.
- **메모리 측면에서의 장/단점**
  - **멀티 프로세스**에서 프로세스를 fork했을 때, 자원 복제 후 메모리에 적재 됨
    → PID와 메모리 주소를 제외하고는 모든 것이 동일하기 때문에,
    결과적으로 동일한 프로세스가 메모리에 적재되므로 **메모리 낭비**.
  - 반면, **멀티스레드**에서는 fork 방식을 이용하지 않고, 공통 영역(데이터/힙/코드)에서 자원을 공유하는 방식을 사용 → **메모리 효율 증대**
- **다른 프로세스, 스레드에 미치는 영향**
  - 멀티 프로세스는 문제가 생겨도 다른 프로세스에 지장이 적거나 없음.
  - 하지만, 멀티 스레드는 모든 스레드가 프로세스의 자원을 공유하기 때문에, 하나의 스레드에 문제가 생기면 다른 스레드도 영향을 받을 수 있음.