프로세스 & 쓰레드

프로세스 vs 프로그램

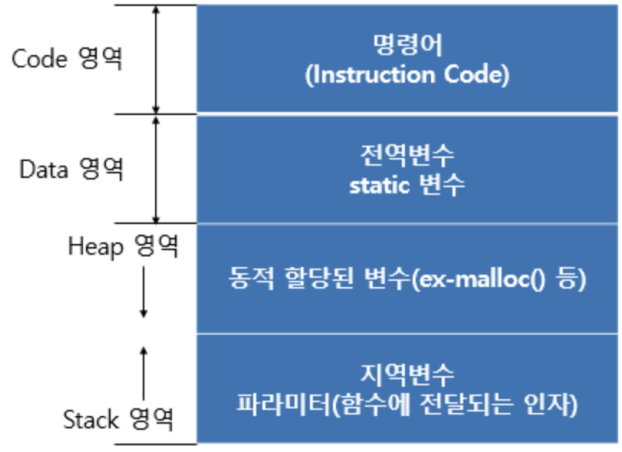
- **프로그램**: 프로그램은 컴퓨터에서 실행 가능한 명령어들의 집합으로, 특정 작업을 수행하기 위한 코드와 데이터의 모음. 프로그램은 주로 디스크나 저장 매체에 저장되어 있으며, 실행을 위해서는 메모리에 로드되어야 한다.

→ **우리가 작성한 코드들은 프로그램**

- **프로세스**: 프로세스는 **실행 중인 프로그램의 인스턴스**로, 프로그램이 메모리에 적재되어 CPU의 할당을 받을 수 있는 것을 말함. 프로세스는 실행 중인 프로그램의 상태와 데이터를 저장하며, 독립적인 실행 단위로 여러 개의 프로세스가 동시에 실행될 수 있다. 프로세스는 작업을 위해 CPU Time, Memory, Files, I/O Devices와 같은 자원이 필요하다.

→ **프로그램을 실행시키면 프로세스**

프로세스 구성

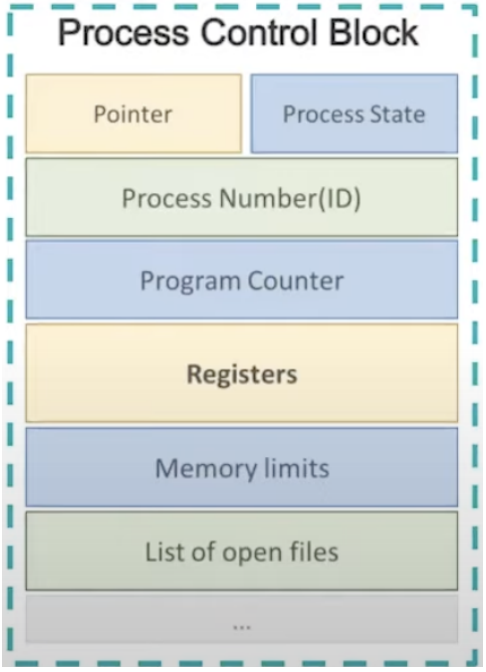


프로세스의 메모리 공간

- Code 영역 : 실행할 프로그램의 코드가 저장되는 텍스트 영역이다. CPU는 코드영역에서 저장된 명령어를 하나씩 가져가서 처리한다.
- Data 영역 : **전역변수**와 **정적변수**가 이해 해당된다. 프로그램의 시작과 함께 할당되며 프로그램이 종료되면 소멸된다.
- Heap 영역 : 힙 영역은 **사용자가 직접 관리할 수 있는 메모리 영역**이다. 힙 영역은 사용자에 의해 메모리공간이 동적으로 할당되고 해제된다

- Stack 영역 : 스택영역은 함수의 호출과 관계되는 **지역변수**와 **매개변수가** 저장되는 영역이다. 함수의 호출과 함께 할당되며, 함수의 호출이 종료될때 해제된다.

서로 다른 프로세스들은 독립된 메모리 공간을 가지며, 한 프로세스가 다른 프로세스에 직접 접근할 수 없다.



Pointer: 준비상태나 대기상태의 큐를 구현하기 위해 필요한 포인터

Process State: 현재 프로세스 상태

Process Number(PID): 프로세스의 고유 번호

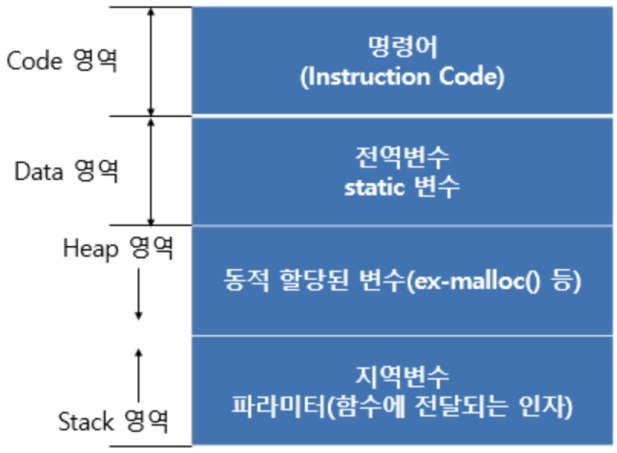
Program Counter: 다음에 실행할 명령어의 주소를 가리킴

쓰레드

쓰레드는 프로세스에서의 실행 단위 즉 하나의 프로세스에서 여러 개의 스레드가 실행될 수 있다.

쓰레드는 프로세스와 유사한 구조를 가지기 때문에 경량화된 프로세스라 할 수 있다.

다만 프로세스와의 가장 큰 차이는 **쓰레드 끼리는 자원을 공유한다는 점**



쓰레드 끼리는 코드,데이터,힙 영역을 서로 공통된 자원으로 사용한다. 다만 스택 영역은 공유하지 않는다.

또한 PC(Program Counter)도 공유하지 않는다.

- **스택을 쓰레드마다 독립적으로 할당하는 이유**

  스택은 함수 호출 시 전달되는 인자, 되돌아갈 주소값 및 함수 내에서 선언하는 지역변수를 저장하기 위해 사용되는 메모리 공간이므로 쓰레드마다 독립적인 함수 호출이 가능하게 하기 위해서 스택영역은 공유하지 않는다.

- **PC(Program Counter) 를 쓰레드마다 독립적으로 할당하는 이유**

  PC 값은 쓰레드의 명령어가 어디까지 수행되었는지를 나타낸다. 쓰레드 사이에서는 컨텍스트 스위칭이 일어나기 때문에 명령어가 연속적으로 수행되지 못하고 어디까지 실행되었는지 기억해야한다. 따라서 PC 또한 공유되지 않고 독립적으로 할당된다.

멀티 프로세싱 vs 멀티 쓰레드

단순히 여러 프로그램을 띄워 놓는게 아닌 한 어플리케이션에 대한 두 가지 처리 방식이라 생각하면 비교가 쉬움

- 컨텍스트 스위치(Context Switch)

이 두 멀티 작업을 진행하기 위해서는 컨텍스트 스위칭이 필요.

우선 프로세스 끼리 컨텍스트 스위칭이 일어난 뒤, CPU를 점유하고 있는 프로세스 안에 여러 쓰레드가 있다면 그 안에서 쓰레드 끼리 컨텍스트 스위칭이 발생한다.

쓰레드 끼리는 공유하는 자원이 많기 때문에 캐싱 적중률이 높아 스위칭 비용이 상대적으로 적다. 프로세스 끼리는 독립적이기 때문에 스위칭 비용이 상대적으로 크다.

그러나 멀티 쓰레드 환경에서는 하나의 쓰레드에 문제가 생기면 다른 쓰레드에 영향을 끼친다는 단점이 있다!

비용이 적지만 책임 범위가 큰 멀티 쓰레드!

비용이 크지만 독립적인 멀티 프로세싱!

대상 시스템의 특징에 따라 적합한 동작 방식을 선택하고 적용해야 한다!

예시: 인터넷 브라우저의 각 탭

explore 브라우저는 멀티 쓰레드 방식을 택해서 한 탭에 문제가 생기면 모든 탭이 꺼지는 문제가 발생

크롬 브라우저는 멀티 프로세스 방식을 택해 하나의 탭이 문제가 생겨도 그 탭만 재시작 해도 됨

크롬 브라우저의 각 탭 안에서는 렌더링, 네트워킹, JavaScript 엔진, 사용자 인터페이스 와 같은 작업들이 멀티 쓰레딩으로 실행되고 있음.

멀티 XXX

1. **멀티 프로그래밍 (Multiprogramming) → CPU 스케쥴링**:
    - 멀티 프로그래밍은 하나의 프로세서가 하나의 프로세스를 수행하는 동안 입출력과 같이 기다리는 시간이 있을 때 다른 프로세스로 전환하여 해당 프로세스의 작업을 진행할 수 있도록 하는 방법! CPU가 항상 어떤 작업을 처리하도록 유지하여 낭비 없이 최대한 효율적으로 쓰기 위한 방법
2. **멀티 태스킹 (Multitasking) → 멀티 프로세싱, 멀티 쓰레딩을 통해 가능:**
    - 목표: 사용자 경험을 향상시키고 작업 효율성을 높이기 위해 여러 작업을 동시에 실행하는 것
    - 동작 방식: 하나의 컴퓨터 시스템에서 여러 응용 프로그램이 동시에 실행되며, CPU 시간을 각 작업에 분배하여 각 작업이 동시에 진행되는 것처럼 보이게 함
3. **멀티 프로세싱 (Multiprocessing):**

- 목표: 여러 개의 독립적인 프로세서 또는 CPU를 사용하여 작업을 병렬로 처리
- 동작 방식: 다수의 프로세서가 동시에 여러 작업을 처리하며, 각 프로세서는 독립적으로 작업을 실행

4. **멀티 쓰레딩 (Multithreading):**
    - 목표: 하나의 프로세스 내에서 여러 스레드를 사용하여 작업을 병렬로 처리하여 성능을 향상시키는 것
    - 동작 방식: 하나의 프로세스 내에서 다수의 스레드가 공유된 자원과 메모리를 사용하며, 스레드 간의 작업을 조율하여 병렬로 실행
5. **멀티 코어 (Multicore) → 병렬프로그래밍:**
    - 동작 방식: 다수의 실행단위를 한 순간에 병렬처리로 실행! 즉 둘 이상의 코어에서 각각 실행단위가 병렬적으로 진행될 수 있게 해주는 방법.
    - 예시: 10개의 멀티 쓰레드 환경에서 코어가 1개일 때는 CPU 할당을 받은 1개의 쓰레드만 실행, 2개일때는 10개 중 CPU 할당을 받은 2개의 쓰레드가 실행!
    - 암달의 법칙: 코어의 개수를 아무리 많이 늘려도(병렬 처리를 늘려도) 병렬 처리를 통한 성능 향상에 한계가 있다! 병렬화 가능한 부분이 많을 수록 전체 성능이 늘어남. 즉! 전체 프로그램을 전부 병렬화 가능한게 아니므로 코어를 아무리 늘려도 성능이 코어의 개수에 비례해서 증가하지 않음을 나타냄!