Process & Thread

프로그램	프로세스
어떤 작업을 하기 위해 실행할 수 있는 파일	실행되어 작업중인 컴퓨터 프로그램
파일이 저장 장치에 있지만 메모리에는 올라가 있지 않은 정적인 상태	메모리에 적재되고 CPU 자원을 할당받아 프로그램이 실행되고 있는 상태
쉽게 말해 그냥 코드 덩어리	그 코드 덩어리를 실행한 것

프로세스(Process)

운영체제로부터 자원을 할당받은 **작업의 단위**

프로세스의 한계

과거에는 프로그램을 실행할 때 프로세스 하나만을 사용해서 이용했었다. 하지만 기술이 발전됨에 따라 프로그램이 복잡해지고 다채로워짐으로써 프로세스 작업 하나만을 사용해서 프로그램을 실행하기에는 한계가 있었다.

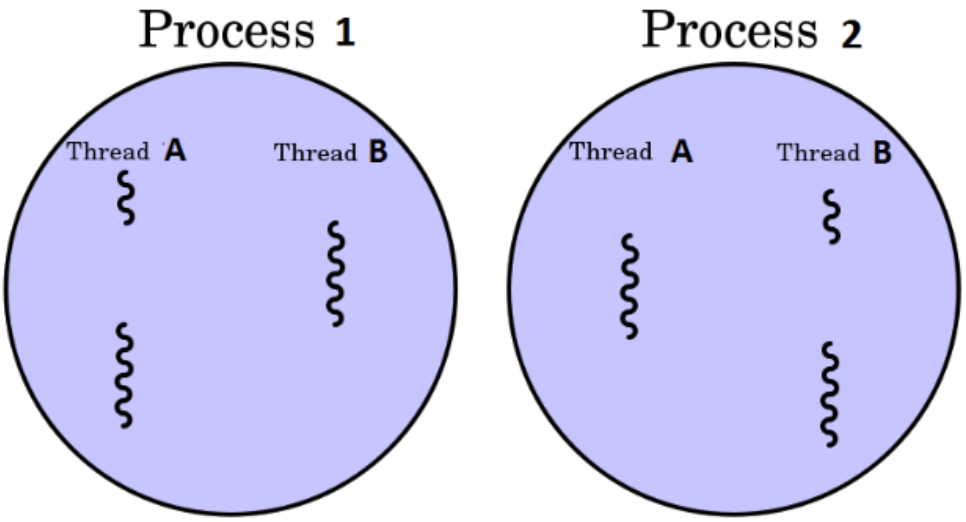
과거에는 파일을 다운받으면 실행 시작부터 실행 끝까지 프로세스 하나만을 사용하기 때문에 다운이 완료될때까지 하루종일 기다려야 했다. 그렇다고 동일한 프로그램을 여러 개의 프로세스로 만들게 되면, 그만큼 메모리를 차지하고 CPU에서 할당받는 자원이 중복되게 될 것이다. 스레드(Thread)는 이러한 프로세스 특성의 한계를 해결하기 위해 탄생 하였다.

스레드(Thread)

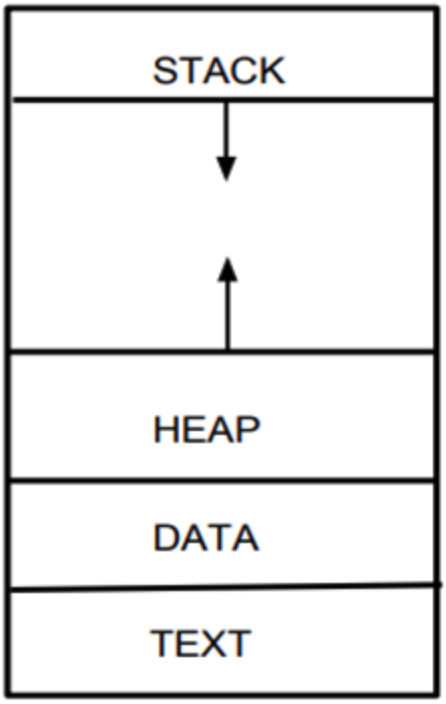
프로세스가 할당받은 자원을 이용하는 **실행 흐름의 단위**

우리는 브라우저에서 파일을 다운 받으며 온라인 쇼핑을 하면서 게임을 하기도 한다.

하나의 프로세스 안에서 여러가지 작업들 흐름이 동시에 진행되기 때문에 가능한 것인데, 이러한 일련의 작업 흐름들을 스레드라고 하며 여러개가 있다면 이를 멀티(다중) 스레드 라고 부른다.

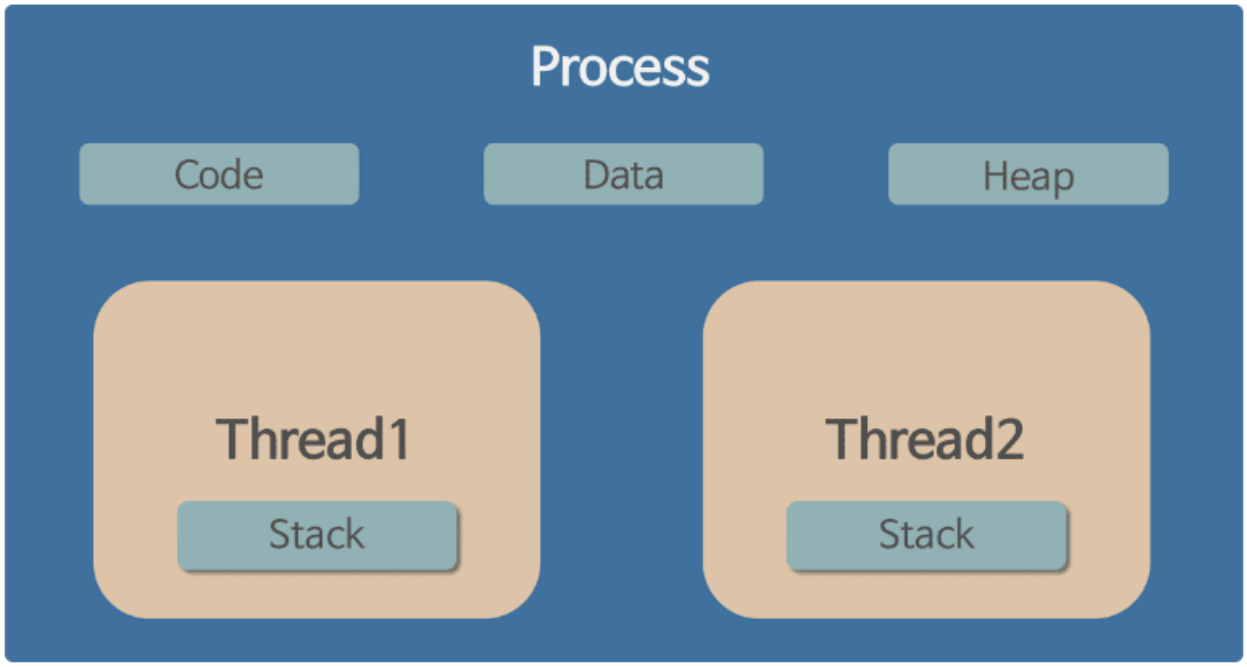


프로세스의 자원 구조



- **코드 영역(Code / Text)** : 프로그래머가 작성한 프로그램 함수들의 코드가 CPU가 해석 가능한 기계어 형태로 저장되어 있다.
- **데이터 영역(Data)** : 코드가 실행되면서 사용하는 전역 변수나 각종 데이터들이 모여있다. 데이터영역은 .data ,.rodata, .bss 영역으로 세분화 된다.
  - .data : 전역 변수 또는 static 변수 등 프로그램이 사용하는 데이터를 저장
  - .BSS : 초기값 없는 전역 변수, static 변수가 저장
  - .rodata : const같은 상수 키워드 선언 된 변수나 문자열 상수가 저장
- **스택 영역(Stack)** : 지역 변수와 같은 호출한 함수가 종료되면 되돌아올 임시적인 자료를 저장하는 독립적인 공간이다. Stack은 함수의 호출과 함께 할당되며, 함수의 호출이 완료되면 소멸한다. 만일 stack 영역을 초과하면 stack overflow 에러가 발생한다.
- **힙 영역(Heap)** : 생성자, 인스턴스와 같은 동적으로 할당되는 데이터들을 위해 존재하는 공간이다. 사용자에 의해 메모리 공간이 동적으로 할당되고 해제된다.

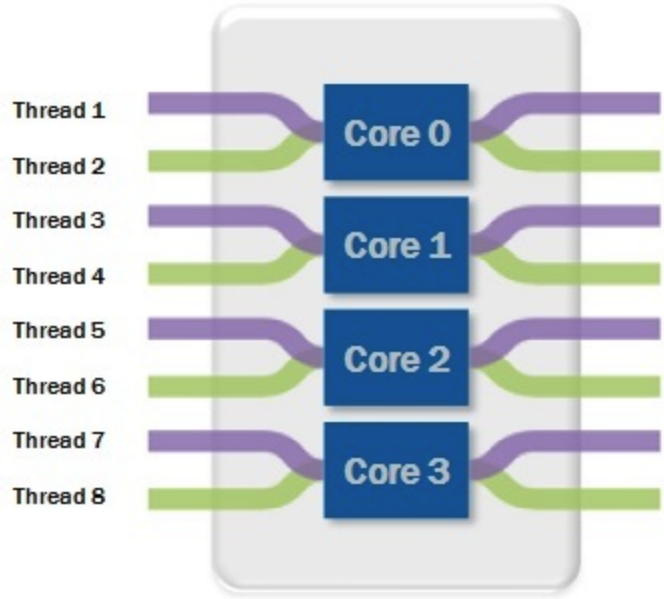
스레드의 자원 구조



스레드는 프로세스가 할당 받은 자원을 이용하는 실행의 단위로서, 스레드가 여러개 있으면 우리가 파일을 다운 받으며 동시에 웹 서핑을 할 수 있게 해준다. 스레드끼리 **프로세스의 자원을 공유**하면서 프로세스 실행 흐름의 일부가 되기 때문에 동시 작업이 가능한 것이다.

병렬성 (Parallelism)

병렬성은 직관적으로 명령어를 메모리에서 뽑아 해석하고 실행하는 반도체 유닛인 여러개의 코어에 맞춰 여러개의 프로세스, 스레드를 돌려 병렬로 작업들을 동시 수행하는 것을 말한다.

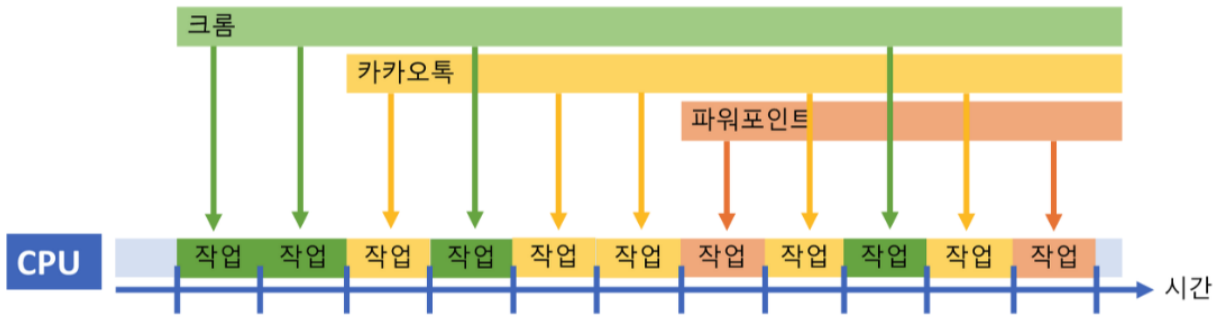


동시성 (Concurrency)

Parallelism가 물리적으로 정말로 동시에 실행하는 것이라고 하면, Concurrency는 동시에 실행하는 것처럼 보이게 하는 것으로 이해하면 된다.

즉, 1개의 코어가 있고 4개의 작업이 있다고 가정하다면, 아래 그림과 같이 프로세스들을 계속 번갈아가면서 조금씩 처리함으로써 마치 프로그램이 동시에 실행되는 것 처럼 보이는 것이다.

단, 이때 작업들을 번갈아가면서 실행할때 작업들을 아주 잘게 나누어 아주 조금씩만 작업을 수행하고 다음 작업으로 넘어가는 식으로 동작된다. 이렇게 하는 이유는 여러 작업을 동시에 처리하는 것처럼 보이게 만들어, 사용자에게 더 빠른 반응성을 제공하기 위해서다. 그리고 이렇게 진행 중인 작업들을 A → B → C → D 로 번갈아 바꾸는 것을 **Context Switching** 이라고 부른다.



동시성이 필요한 이유

첫번째는 **하드웨어적 한계** 때문이라고 할 수있다. CPU 발열 때문에 깡 클럭으로 성능을 올리기에는 한계에 봉착됐기 때문에 코어의 성능을 올리는 대신 코어를 여러개 탑재하여 쿼드 코어, 옥타 코어 CPU들을 출시하고 있다. 하지만 아무리 코어를 많이 넣어도 수십개의 코어를 넣은순 없으니 결국 하드웨어적 제한이 걸리게 되고 수십수백개의 프로세스를 돌리기 위해선 결국 동시성이 필요한 것이다.

두번째는 보다 **논리적인 효율**적인 이유에서이다. 4코어 8스레드의 CPU 환경에서 현재 총 16개의 작업이 있다고 가정을 해보자. 그중 8개는 오래 걸리는 작업이고, 나머지 8개는 짧은 시간을 필요로 하는 작업이라고 한다. 논리적인 8개의 코어이니 최대 8개까지 동시에 실행할수 있을텐데, 만일 최악의 경우 8개의 오래 걸리는 작업이 먼저 동시에 처리되기 시작했다고 하자. 이 경우 나머지 가벼운 8개의 작업은 처리하는데 짧은 시간이 걸리는 데에도 불구하고 현재 처리중인 8개의 작업이 다 끝날때 까지 기다려야 할 것이다. 따라서 이러한 비효율적인 면을 극복하기 위해 작업을 아주 잘게 나눠 번갈아 가면서 처리하는 동시성 개념을 채택한 것이다.

Multi Process

멀티 프로세스는 운영체제에서 하나의 응용 프로그램에 대해 동시에 여러 개의 프로세스를 실행할 수 있게 하는 기술을 말한다. 보통 하나의 프로그램 실행에 대해 하나의 프로세스가 메모리에 생성되지만, 부가적인 기능을 위해 여러개의 프로세스를 생성하는 것이다.

멀티 프로세스의 장점

프로그램 안전성

멀티 프로세스는 각 프로세스가 독립적인 메모리 공간을 가지므로, 한 프로세스가 비정상적으로 종료되어도 다른 프로세스에 영향을 주지 않는다. 그래서 프로그램 전체의 안전성을 확보할 수 있다는 장점이 있다.

예를 들자면 크롬 브라우저에서 여러개의 탭을 띄우고 여러곳의 웹사이트를 방문해 서비스를 이용한다고 하자. 이때 어느 한 탭의 웹사이트에서 무언가 잘못되어 먹통이 되었다.

시스템 확장성

멀티 프로세스는 각 프로세스가 독립적이므로, 새로운 기능이나 모듈을 추가하거나 수정할때 다른 프로세스에 영향을 주지 않는다. 그래서 시스템의 규모를 쉽게 확장할 수 있다.

멀티 프로세스의 단점

Context Switching Overhead

멀티 태스킹을 구성하는데 핵심 기술인 컨텍스트 스위칭(context switching) 과정에서 성능 저하가 올 수 있다. 특히나 프로세스를 컨텍스트 스위칭 하면, CPU는 다음 프로세스의 정보를 불러오기 위해 메모리를 검색하고, CPU 캐시 메모리를 초기화하며, 프로세스 상태를 저장하고, 불러올 데이터를 준비해야 하기 때문에, 이로 인한 빈번한 Context Switching 작업으로 인해 비용 오버헤드가 발생할 수 있게 된다. 반면 스레드를 컨텍스트 스위칭하면 프로세스 스위칭 보다 가벼워 훨씬 빠르고 좋다.

자원 공유 비효율성

멀티 프로세스는 각 프로세스가 독립적인 메모리 공간을 가지므로, 결과적으로 메모리 사용량이 증가하게 된다.

만일 각 프로세스간에 자원 공유가 필요할 경우 프로세스 사이의 어렵고 복잡한 통신 기법인 IPC(Inter-Process Commnuication)을 사용하여야 한다.

IPC란 운영체제 상에서 실행 중인 프로세스 간에 정보를 주고받는 메커니즘을 말한다. 이를 위해 파이프, 소켓, 메세지 큐 등 다양한 방법이 사용된다. 그런데 IPC 자체로 오버헤드가 발생한다. 예를 들어, 파이프나 소켓과 같은 IPC 기법은 데이터를 복사하거나 버퍼링하는 과정에서 성능 저하가 발생할 수 있기 때문이다. 또한 코드의 복잡도를 증가시킨다.

Multi Thread

스레드는 하나의 프로세스 내에 있는 실행 흐름이다. 그리고 멀티 스레드는 하나의 프로세스 안에 여러개의 스레드가 있는 것을 말한다. 따라서 하나의 프로그램에서 두가지 이상의 동작을 동시에 처리하도록 하는 행위가 가능해진다.

웹 서버는 대표적인 멀티 스레드 응용 프로그램이다. 사용자가 서버 데이터베이스에 자료를 요청하는 동안 브라우저의 다른 기능을 이용할 수 있는 이유도 바로 멀티 스레드 기능 덕분인 것이다. 즉, 하나의 스레드가 지연되더라도, 다른 스레드는 작업을 지속할 수 있게 된다.

멀티 스레드의 장점

윈도우, 리눅스 등 많은 운영체제들이 멀티 프로세싱을 지원하고 있지만 멀티 스레딩을 기본으로 하고 있다. **왜 멀티 프로세스 보다 멀티 스레드로 프로그램을 돌리는 것이 유리한지** 그 이유에 대해 알아보자. (이는 스레드 자체의 장점이기도 하다)

스레드는 프로세스보다 가벼움

일단 스레드는 프로세스 보다 용량이 가볍다. 그도 그럴게 스레드는 프로세스 내에서 생성되기 때문에 스레드의 실행 환경을 설정하는 작업이 매우 간단하여 생성 및 종료가 빠르다. 또한 스레드는 프로세스와 달리, 코드, 데이터, 스택 영역을 제외한 나머지 자원을 서로 공유하기 때문에 기본적으로 내장되어 있는 데이터 용량이 프로세스보다 당연히 작다. 그래서 스레드를 생성하고 제거할 때, 프로세스 내부의 자원만을 관리하면 되기 때문에 프로세스 생성, 제거 보다 훨씬 빠른 것이다.

자원의 효율성

멀티 스레드는 하나의 프로세스 내에서 여러 개의 스레드를 생성되기 때문에, heap 영역과 같은 공유 메모리에 대해 스레드 간에 자원을 공유가 가능하다. 이를 통해, 프로세스 간 통신(IPC)을 사용하지 않고도 데이터를 공유할 수 있기 때문에, 자원의 효율적인 활용이 가능해 시스템 자원 소모가 줄어든다

Context Switching 비용 감소

스레드에도 컨텍스트 스위칭 오버헤드가 존재한다. 하지만 상대적으로 프로세스 컨텍스트 스위칭 오버헤드보다 훨씬 낮아 비용이 낮다는 장점이 있다.

멀티 스레드의 단점

안정성 문제

멀티 프로세스 모델에서는 각 프로세스가 독립적으로 동작하므로 하나의 프로세스에서 문제가 발생해도 다른 프로세스들은 영향을 받지 않기 때문에 프로그램이 죽지 않고 계속 동작할 수 있다. 그러나 멀티 스레드 모델에서는 기본적으로 하나의 스레드에서 문제가 발생하면 다른 스레드들도 영향을 받아 전체 프로그램이 종료될 수 있다.

데드락 (교착 상태)

Deadlock 이란, 다수의 프로세스나 스레드가 서로 자원을 점유하고, 다른 프로세스나 스레드가 점유한 자원을 기다리는 상황에서 발생하는 교착 상태를 말한다. 여러 개의 스레드가 서로 대기하면서 무한정 기다리게되는 무한 루프와 같은 증상이라고 보면된다.