Process & Thread

1. 프로세스 & 스레드 개념

- **프로세스 (Process)**

  : 운영체제로부터 자원을 할당받은 **작업의 단위**

- **스레드 (Thread)**

  : 프로세스가 할당받은 자원을 이용하는 **실행 흐름의 단위**

2. 프로그램과 프로세스

- **정적 프로그램 (Static Program)**

  : 프로그램은 윈도우의 ***.exe** 파일이나 Mac의 ***.dmg** 파일과 같은 **컴퓨터에서 실행할 수 있는 파일**을 통칭한다. 단, 아직 **파일을 실행하지 않은 상태**이기 때문에 정적 프로그램을 줄여서 프로그램이라고 부른다.

  어떠한 프로그램을 개발하기 위해서는 자바나 C언어와 같은 언어를 이용해 코드를 작성하여 완성된다. 즉, 프로그램은 쉽게 말해서 그냥 코드 덩어리인 것이다.

- **프로세스 (Process)**

  : 프로그램이 그냥 코드 덩어리이면, 프로세스는 프로그램을 실행시켜 정적인 프로그램이 동적으로 변하여 **프로그램이 돌아가고 있는 상태**를 말한다. 즉, 컴퓨터에서 작업 중인 프로그램을 의미하는 것이다.

  모든 프로그램은 운영체제가 실행되기 위한 **메모리 공간**을 할당해줘야 실행될 수 있다. 그래서 프로그램을 실행하는 순간 파일은 컴퓨터 메모리에 올라가게 되고, 운영체제로부터 시스템 자원(CPU)을 할당받아 프로그램 코드를 실행시켜 우리가 서비스를 이용할 수 있게 되는 것이다.

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

3. 스레드

- **프로세스의 한계**

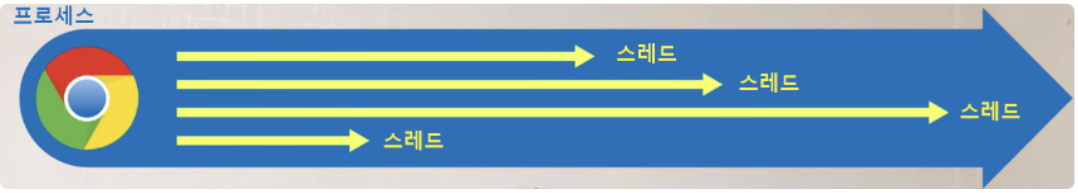
  : 과거에는 프로그램을 실행할 때 프로세스 하나만을 사용해서 이용했었다. 하지만 기술이 발전됨에 따라 프로그램이 복잡해지고 다채로워짐으로써 프로세스 작업 하나만을 사용해서 프로그램을 실행하기에는 한계가 있었다.

  오늘날 컴퓨터는 파일을 다운 받으며 다른 일을 하는 멀티 작업은 너무 당연한 기능이라고 생각할 지 모르지만, 과거에는 파일을 다운받으면 실행 시작부터 실행 끝까지 프로세스 하나만을 사용하기 때문에 다운이 완료될 때까지 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 탄생했다.

- **스레드의 개념**

  : 스레드란, **하나의 프로세스 내에서 동시에 진행되는 작업 갈래, 흐름의 단위**를 말한다.

이해하기 쉽게 비유를 들자면, 크롬 브라우저가 실행되면 프로세스가 하나가 생성될 것이다. 그런데 우리는 브라우저에서 파일을 다운 받으며 온라인 쇼핑을 하면서 게임을 하기도 한다.



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

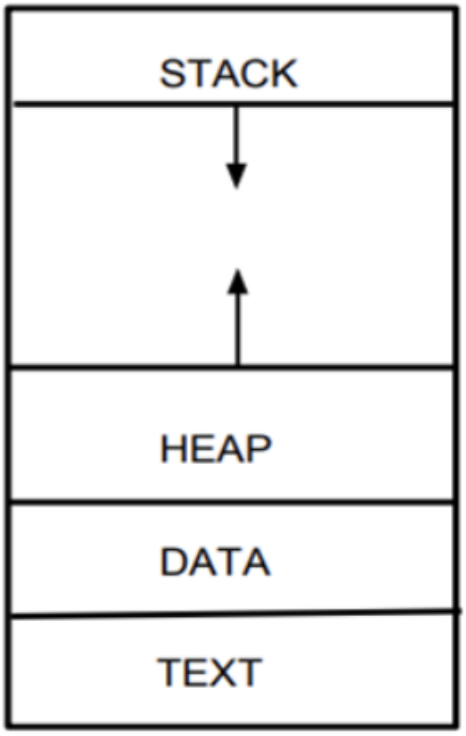
아래 그림에서 보듯이 하나의 프로세스 안에 여러개의 스레드들이 들어있다고 보면 된다. **스레드 수가 많을수록** 당연히 프로그램 속도도 동시에 하는 작업이 많아져 **성능이 올라간다.**

일반적으로 하나의 프로그램은 하나 이상의 프로세스를 가지고 있고, 하나의 프로세스는 반드시 하나 이상의 스레드를 갖는다. **즉, 프로세스를 생성하면 기본적으로 하나의 main 스레드가 생성되게 된다**. 스레드 2개, 3개 ... 는 프로그램을 개발한 개발자가 직접 프로그래밍하여 위치 시켜주어야 한다.

4. 프로세스와 스레드의 내부 작동 방식

- **프로세스& 스레드의 메모리**

  : 프로그램이 실행되어 프로세스가 만들어지면 다음 4가지의 메모리 영역으로 구성되어 할당받게 된다.



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

코드 영역과 데이터 영역은 선언할 때 그 크기가 결정되는 정적 영역이지만, **스택 영역과 힙 영역**은 프로세스가 실행되는 동안 **크기가 늘어났다 줄어들기도 하는 동적 영역**이기 때문에 이를 표현한 것이다.

프로그램이 여러개 실행된다면 메모리에 프로세스들이 담길 주소 공간이 생성되게 되고 그 안에 Code, Data, Stack, Heap 공간이 만들어지게 된다.

5. 스레드의 자원 공유

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

이때, 프로세스의 4가지 메모리 영역(Code, Data, Heap, Stack) 중 **스레드**는 **Stack**만 할당 받아 복사하고, **Code, Data, Heap**은 프로세스 내의 **다른 스레드들과 공유**된다. 따라서 각각의 스레드는 별도의 stack을 가지고 있지만, heap 메모리는 고유하기 때문에 서로 다른 스레드에서 가져와 읽고 쓸 수 있게 된다.

stack은 함수 호출 시 전달되는 인자, 되돌아갈 주소값, 함수 내에서 선언하는 변수 등을 저장하는 메모리 공간이기 때문에, 독립적인 스택을 가졌다는 것은 독립적인 함수 호출이 가능하다는 것은 독립적인 실행 흐름이 추가된다는 말이다.
즉, stack을 가짐으로써 스레드는 독립적인 실행 흐름을 가질 수 있게 되는 것이다.

> 💡 반면에 프로세스는 기본적으로 프로세스끼리 다른 프로세스의 메모리에 직접 접근할 수는 없다.

이렇게 구성한 이유는 하나의 프로세스를 다수의 실행 단위인 스레드로 구분하여 자원을 공유하고, 자원의 생성과 관리의 중복성을 최소화하여 수행 능력을 올리기 위해서다.

6. 프로세스의 자원 공유

기본적으로 각 프로세스는 메모리에 별도의 주소 공간에서 실행되기 때문에 한 프로세스는 다른 프로세스의 변수나 자료구조에 접근할 수는 없다. 그렇다면 프로세스는 영원히 다른 프로세스 정보에 접근할 수 없을까?

현재 우리가 사용하는 대부분의 컴퓨터 프로그램을 보면 다른 프로그램에 있는 정보를 가져오는 경우는 심심치 않게 볼 수 있을 것이다.

이처럼 특별한 방법을 통해 **프로세스가 다른 프로세스 정보에 접근**하는 것이 가능하다.

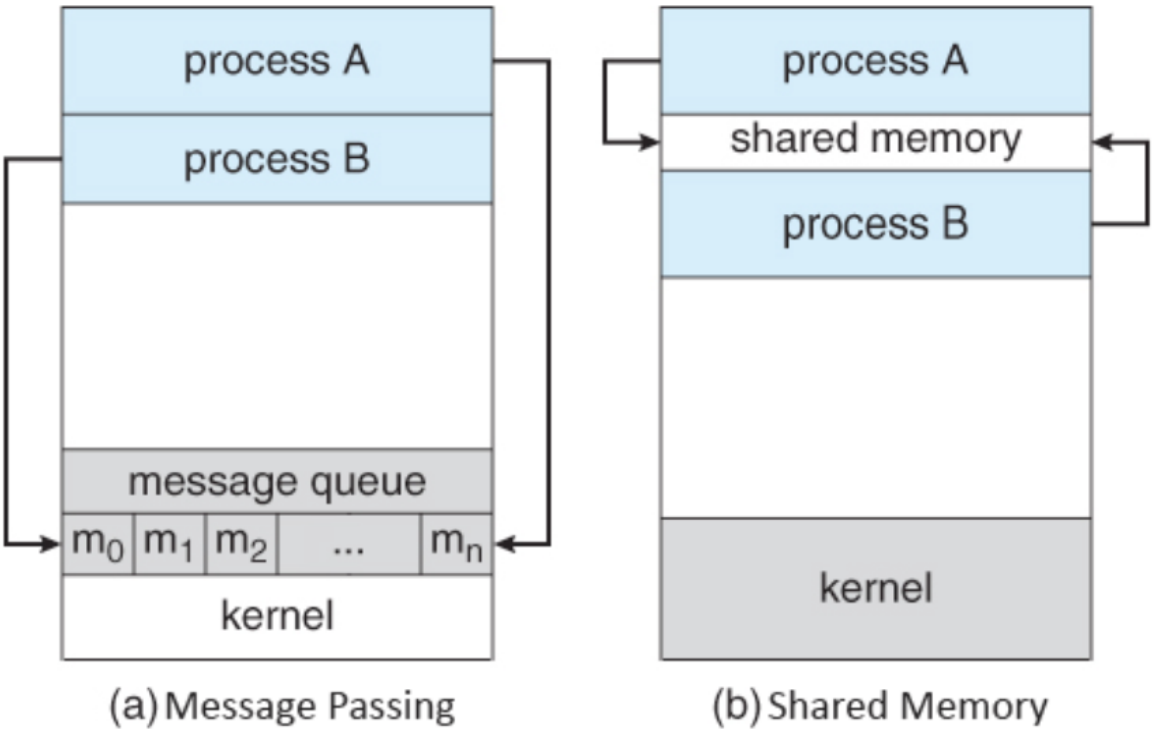
1. **IPC (Inter-Process Communication) 사용**
2. **LPC (Local inter-Process Communication) 사용**
3. **별도로 공유 메모리를 만들어서 정보를 주고받도록 설정**

💡 **컴퓨터 프로세스간 통신(IPC) | Inter-Process Communication**

:
운영체제 상에서 **실행 중인 프로세스 간에 정보를 주고 받는 것을 Inter Process Communication(IPC)**라고 한다. 프로세스는 자신에게 할당된 메모리 내의 정보만 접근할 수 있고, 이를 벗어나서 접근할 경우, C언어를 배우는 사람이라면 누구나 한 번쯤은 봤을 Segmentation Fault 등의 오류가 발생하게 된다. 이는 **안정성을 위해 운영체제에서 자기 프로세스의 메모리만 접근하도록 강제**하고 있다는 것이다.

따라서 한 프로그램에서 병렬성을 키우면서 공유되는 데이터를 사용하기 위해 메모리 공간을 공유하는 스레드를 이용하는 경우가 많다. (물론 스레드를 쓰는 이유는 이것 말고도 매우 다양하다) 하지만 이것은 하나의 프로그램에서만 의미가 있는 것이고, 서로 다른 프로그램(즉, 서로 다른 프로세스)의 데이터를 공유하려면 결국 다른 프로세스의 메모리를 접근할 필요가 발생한다. 따라서, 이때는 IPC라는 것을 사용한다.

로컬 컴퓨터 프로세스간 통신(LPC) | Local Inter-Process Communication
:
프로세스가 다른 프로세스와 정보를 주고 받을 때 빠른 속도로 데이터를 교환하기 위해 사용한다. 다만 이건 커널에서만 쓸 수 있으며 일반 프로그램은 LPC를 사용할 수 없다.



(a) Message Queue : FIFO 방식의 데이터 전송으로 동일한 key값을 이용하여 A프로세스의 데이터를 B프로세스가 받을 수 있다. 반대로, B프로세스에서도 A프로세스로 데이터를 보낼 수 있다. (양방향)

(b) Shared Memory : 커널 공간에 메모리 공간을 만들어 해당 공간을 마치 변수처럼 사용하는 방법이다. 공유 메모리 key를 사용하여 여러 프로세스 접근이 가능하다.

그러나 프로세스 자원 공유는 단순히 CPU 레지스터 교체 뿐만이 아니라 RAM과 CPU 사이의 캐시 메모리까지 초기화되기 때문에 **자원 부담이 크다는 단점**이 있다. 그래서 **다중 작업이 필요한 경우 스레드를 이용하는 것이 훨씬 효율적**이라, 현대 컴퓨터의 운영체제에선 다중 프로세싱을 지원하고 있지만 다중 스레딩을 기본으로 하고 있다.

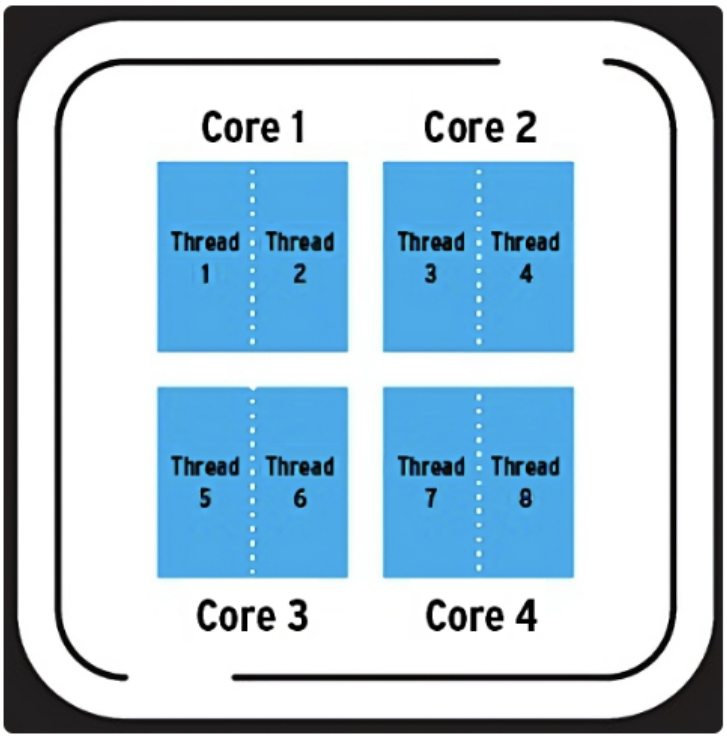
7. 프로세스 & 스레드의 동시 실행 원리

: 우리가 음악을 들으면서, 웹 서핑을 하고, 메신저의 메시지를 확인할 수 있는 이유는 컴퓨터 내부적으로 **프로세스와 스레드를 동시에 처리하는 멀티태스킹 기술** 때문이다. 하지만 여기서 동시에 처리한다는 것이 심플하게 CPU 프로세서가 프로그램들을 한꺼번에 동시에 돌리는 것으로 생각하겠지만, 내부적으로 복잡한 원리에 의해 처리가 된다.

- **멀티 코어와 스레드**

  : 한번 컴퓨터 견적을 맞춰본 경험이 있는 독자분들은 **4코어 8쓰레드 CPU**에 대한 단어를 본 적이 있을 것이다.

  CPU 한 개는 여러개의 코어를 가질 수 있다. 코어는 말 그대로 CPU 코어 유닛이다. 즉, 명령어를 메모리에서 뽑아 해석하고 실행하는 반도체 유닛이 4개가 있는 것이다. **4코어가 물리적 코어 개수**이면, **8쓰레드는 논리적 코어 개수**이다. 이 경우 **물리적 코어 하나가 스레드 두개 이상을 동시에 실행가능하다는 의미**가 된다. 즉, 운영체제가 8개의 작업을 동시에 처리할 수 있다는 뜻이다. 이를 **하이퍼스레딩 기술**이라 말한다.



> 💡 단, 여기서 CPU의 쓰레드는 우리가 배운 프로세스의 쓰레드와는 조금 다른 개념이다. 엄밀히 말하자면 **CPU의 스레드**는 **하드웨어적 스레드**이고, **프로그램의 스레드**는 **소프트웨어적 스레드**로 구분한다.

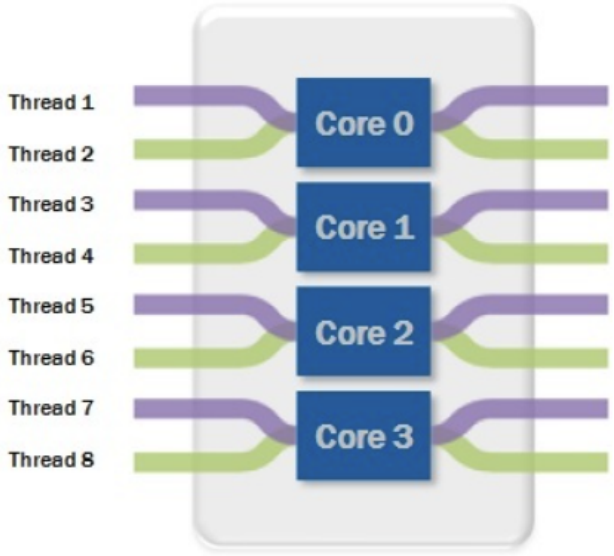
8. CPU의 작업 처리 방식

그런데 우리는 컴퓨터를 이용할 때 프로그램을 수십, 수백개를 켜놓고 이용한다. 그럼 그 수십수백개의 프로세스들을 고작 8개의 논리적인 스레드로 어떻게 처리하는 것일까?

이 원리를 알기 위해선 **병렬성(Parallelism)**과 **동시성(Concurrency)**이라는 개념을 알고 있어야 한다. 이 개념은 운영체제의 프로세스, 스레드를 이해하는데 있어 가장 핵심 골자이다.

- **병렬성(Parallelism)**

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



  듀얼 코어, 쿼드 코어, 옥타 코어 등 이런 명칭이 붙는 **멀티코어 프로세스**가 달린 컴퓨터에서 할 수 있는 방식이다.

- **동시성(Concurrency)**

  : 동시성은 **둘 이상의 작업이 동시에 실행**되는 것을 의미한다. 이 '동시'라는 의미에서 병렬성과 동시성의 한글 의미가 헷갈릴 수 있다. **Parallelism가 물리적으로 정말로 동시에 실행하는 것**이라고 하면, **Concurrency는 동시에 실행하는 것처럼 보이게 하는 것**으로 이해하면 된다.

단, 이때 작업들을 번갈아가면서 실행할 때 작업들을 **아주 작게 나누어 아주 조금씩만 작업을 수행**하고 다음 작업으로 넘어가는 식으로 동작된다. 이렇게 하는 이유는 여러 작업을 동시에 처리하는 것 처럼 보이게 만들어, 사용자에게 **더 빠른 반응성**을 제공하기 위해 **Context Switching**이라고 부른다.

- **동시성이 필요한 이유**

  : 그런데 상식적으로 생각해보면 동시성(Concurrency)은 '동시에 돌아가는 것처럼' 보이는 거지, 정말 실제로 동시에 돌아가는 것이 아니기 때문에 최종 작업이 걸리는 시간은 거의 차이가 없을 것이다. 병렬성은 정말로 각 코어에 프로세스를 나눠 실행하는거니까 듀얼 코어면 반 이상 줄어들텐데 말이다. 그렇다면 왜 이렇게 번거롭게 작업들을 스위칭하며 처리하는 것일까?

  1. **하드웨어적 한계**

     : CPU 발열 때문에 클럭으로 성능을 올리기에는 한계에 봉착했기 때문에 코어의 성능을 올리는 대신 코어를 여러개 탑재하여 쿼드 코어, 옥타 코어 CPU들을 출시하고 있다. 하지만 아무리 코어를 많이 넣어도 수십개의 코어를 넣을 순 없으니 결국 하드웨어적 제한이 걸리게 되고, 수십 수백개의 프로세스를 돌리기 위해선 결국 동시성이 필요한 것이다.

  2. **논리적 효율**

     : 4코어 8스레드의 CPU 환경에선 현재 총 16개의 작업이 있다고 가정을 해보자. 그중 8개는 오래 걸리는 작업이고, 나머지 8개는 짧은 시간을 필요로 하는 작업이라고 하자.

     논리적인 8개의 코어이니 최대 8개까지 동시에 실행할 수 있을텐데, 만일 최악의 경우 8개의 오래 걸리는 작업이 먼저 동시에 처리되기 시작했다고 하자. 이 겨웅 나머지 가벼운 8개의 작업은 처리하는데 짧은 시간이 걸리는데도 불구하고 현재 처리중인 8개의 작업이 다 끝날때까지 기다려야 할 것이다 따라서 이러한 비효율적인 면을 극복하기 위해 **작업을 아주 잘게 나눠 번갈아 가면서 처리**하는 동시성 개념을 채택한 것이다.

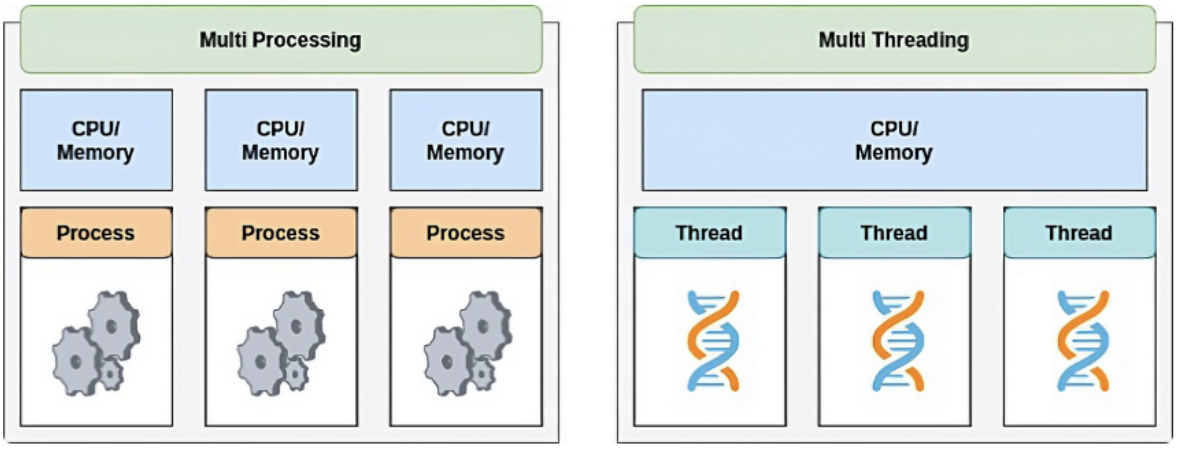
     따라서 최대 8개의 작업에 대해서 8개의 논리적인 스레드가 병렬적으로 아주 빠르게 동시적으로 작업을 하면서, 그보다 많은 수십개의 소프트웨어적 스레드가 있다면 **적절히 병렬성과 동시성**을 섞어 동시에 돌리게 된다.

9. 프로세스& 스레드의 생명 주기

: **프로세스와 스레드는 각각의 생명 주기**를 가지고 있으며, **운영체제는 이러한 생명 주기를 관리**하고, 프로세스와 스레드를 조정하여 시스템 자원을 효율적으로 사용할 수 있게 된다.

- **프로세스 스케줄링**

  : 프로세스 스케줄링(Process Scheduling)은 운영체제에서 CPU를 사용할 수 있는 프로세스를 선택하고, CPU를 할당하는 작업을 말한다. **프로세스 스케줄링은 프로세스의 우선순위, 작업량 등을 고려하여 효율적으로 배치**하여, 이를 통해 운영체제는 CPU를 효율적으로 사용하며 시스템 전반적인 성능을 향상시킨다. 그래서 스케줄링은 멀티 태스킹 작업을 만들어내는데 있어 핵심적인 부분이다.



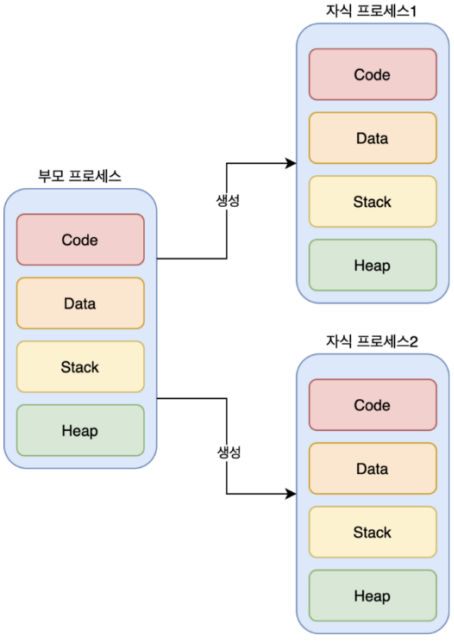
10. 멀티 프로세스

: 멀티 프로세스는 운영체제에서 하나의 응용 프로그램에 대해 동시에 여러 개의 프로세스를 실행할 수 있게하는 기술이다. 보통 하나의 프로그램 실행에 대해 하나의 프로세스가 메모리

에 생성되지만, 부가적인 기능을 위해 여러개의 프로세스를 생성하는 것이다.

> 💡 **멀티 프로세스 VS 멀티 프로세서**
> **프로세스(process)는 프로그램의 실행 상태**를 말하고, **프로세서(processor)는 CPU 코어**를 일컫는다.
> 따라서 멀티 프로세스는 하나의 프로그램에서 여러 개의 프로세스를 실행하는 것을 의미하고, 멀티 프로세서는 여러 개의 CPU 코어가 하나의 시스템에서 동시에 실행되는 것을 의미한다.

멀티 프로세스 내부를 보면, **하나의 부모 프로세스가 여러 개의 자식 프로세스를 생성함으로서 다중 프로세스를 구성**하는 구조이다. 한 프로세스는 실행되는 도중 프로세스 생성 시스템 콜을 통해 새로운 프로세스들을 생성할 수 있는데, 다른 프로세스를 생성하는 프로세스를 **부모 프로세스(Parent Process)**라고 하고, 다른 프로세스에 의해 생성된 프로세스를 **자식 프로세스(Child Process)**라 한다.



부모 프로세스와 자식 프로세스는 각각 고유한 PID (Process ID)를 가지고 있다.

- **부모 프로세스**

  : 자식 프로세스의 PID를 알고 있어, 이를 통해 자식 프로세스를 제어할 수 있다.

- **자식 프로세스**

: 부모 프로세스의 PID와 PPID(Parent Process ID)를 알고 있어, 이를 통해 부모 프로세스와 통신이 가능하다.

💡 다만, 통신이 가능할 뿐이지, 부모 프로세스와 자식 프로세스는 엄연히 서로 다른 프로세스로 독립적으로 실행되며, 독립적인 메모리 공간을 가지고 있어 서로 다른 작업을 수행한다.
대표적인 예로
웹 브라우저의 상단 탭(Tab)이나 새 창을 들 수 있다. 각 브라우저 탭은 같은 브라우저 프로그램 실행이지만, 각기 다른 사이트 실행을 행하기 때문이다.

11. 멀티 프로세스의 장점

1. **프로그램 안정성**

   : 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 들어, 크롬 브라우저에서 여러 개의 탭을 띄우고 여러 곳의 웹사이트를 방문해 서비스를 이용한다고 했을 때 한 탭의 웹사이트에서 무언가 잘못되어 먹통이 되었다.
   아주 심각한 오류가 아닌 이상, 당장 그 브라우저 탭의 웹 사이트는 이용을 못하겠지만, 다른 탭은 별 문제 없이 이용이 가능할 것이다. 이러한 이유는 자식 프로세스가 여러개 생성되어 메모리에 별도로 관리되기 때문이다.

2. **프로그램 병렬성**

   : 멀티 프로세스와 여러개의 CPU 코어를 활용하여 둘의 시너지를 합쳐, 다중 CPU 시스템에서 각 프로세스를 병렬적으로 실행하여 성능을 향상시킬 수 있다. 예를 들어, 이미지 처리나 비디오 인코딩과 같은 작업을 여러개의 코어와 CPU에 분산시켜 빠르게 처리할 수 있다.

→ 이 부분은 멀티 프로세스와 멀티 스레드 둘의 장점이 옳다.

3. **시스템 확장성**

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

   예를 들어, 대규모 웹 서비스에서는 수많은 요청을 동시에 처리하기 위해 여러대의 서버를 두고 로드 밸런서와 같은 장비를 사용하여 클라이언트 요청 트래픽을 분산시킨다. 이때 여러 대의 서버는 컴퓨터를 여러개 말하는 것일 수도 있고, **하나의 성능 좋은 컴퓨터에 여러개의 서버 프로세스를 두는 것**을 말하기도 한다. 멀티 프로세스를 사용하여 여러 대의 서버에 요청을 분산시켜 처리함으로써, 시스템의 규모를 쉽게 확장할 수 있으며, 부가로 서버의 당애나 다운타임을 최소화할 수 있게 되는 것이다.

12. 멀티 프로세스의 단점

1. **Context Switching Overhead 발생**

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

   따라서, 멀티 프로세스 환경에서는 Context Switching Overhead를 최소화하기 위해 다양한 방법을 고려한다.

   1. **프로세스 수를 적정하게 유지**
   2. **I/O 바운드 작업이 많은 프로세스와 CPU 바운드 작업이 많은 프로세스를 분리하여 관리**
   3. **CPU 캐시를 효율적으로 활용**

2. **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 Communication)을 사용해야 한다.

   > 💡 IPC란 프로세스 간에 정보를 주고 받는 메커니즘을 말한다. 이를 위해 파이프, 소켓, 메시지 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.

13. 멀티 스레드

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

멀티 프로세스	멀티 스레드
웹 브라우저에서의 여러 탭이나 여러 창	웹 브라우저의 단일 탭 또는 창 내에서 브라우저 이벤트 루프, 네트워크 처리, I/O 및 기타 작업을 관리하고 처리

14. 멀티 스레드의 장점

: 윈도우, 리눅스 등 많은 운영체제들이 멀티 프로세싱을 지원하고 있지만 멀티 스레딩을 기본으로 하고 있다.

1. **스레드는 프로세스보다 가벼움**

`: 스레드는 프로세스보다 용량이 가볍고, 프로세스 내에서 생성되기 때문에 스레드의 실행 환경을 설정하는 작업이 매우 간단하여 생성 및 종료가 빠르다.

또한, 스레드는 코드, 데이터, 스택 영역을 서로 공유하기 때문에 기본적으로 내장되어 있는 데이터 용량이 프로세스보다 당연히 작다. 스레드를 생성하고 제거할 때, 프로세스 내부의 자원만을 관리하면 되기 때문에 프로세스 생성, 제거보다 훨씬 빠른 것이다.

2. **자원의 효율성**

: IPC를 사용하지 않고도, 데이터를 공유할 수 있기 때문에 자원의 효율적인 활용이 가능하다.

3. **Context Switching 비용 감소**

: 스레드에도 컨텍스트 스위칭 오버헤드가 존재하지만, 스레드 컨텍스트 스위칭 비용은 스레드 간에 공유하는 자원을 제외한 **스레드 정보(stack, register)만을 교체하면 되므로** 상대적으로 비용이 낮다.

4. **응답 시간 단축**

: 컨텍스트 스위칭 오버헤드가 작아 멀티 프로세스보다 응답 시간이 빠르다.

15. 멀티 스레드의 단점

1. **안정성 문제**

: 멀티 스레드 모델은 하나의 스레드에서 문제가 발생하면 다른 스레드들도 영향을 받아 전체 프로그램이 종료될 수 있다.

2. **동기화로 인한 성능 저하**

: 멀티 스레드 모델은 여러 개의 스레드가 공유 자원에 동시에 접근할 수 있기 때문에 동기화 문제가 발생할 수 있다. 따라서 스레드간 동기화가 필수적이다.

동기화 작업은 여러 스레드들이 자원에 대한 접근을 순차적으로 통제하는 것이므로 **병목 현상**이 일어나 성능이 저하될 수 있다.

→ 이를 해결하기 위해 임계 영역(Critical Section)에 대하여 **뮤텍스**, 또는 **세마포어** 방식을 활용한다.

3. **데드락 (교착 상태)**

: 여러 개의 스레드가 서로 대기하면서 무한정 기다리게 되는 증상이다.

4. **디버깅이 어려움**

: 코드를 디버깅하는 도중에 다른 스레드가 실행되어 예기치 못한 결과가 발생할 수 있다. 또한 각 스레드의 동작을 추적하기 어려울 수 있다.