4. 운영체제 관점에서의 스레드에 대해서 공부해봅시다

1) **프로세스**가 가지고있는 영역과, **스레드가** 가지고있는 영역을 비교해봅시다

2) 멀티 스레드 프로그래밍이 멀티 프로세스 프로그래밍에 비해 어떤 장점이 있을까요?

3) 병행성과 병렬성

4) 멀티 스레드 기반으로 프로그래밍 시 주의해야할 점

5) 스레드풀에 대해서 간단하게만 소개해주세요! (별거 없을지도..?)

이게 왜 중요할까?

=> 운영체제가 시스템 자원을 어떤 방식으로 할당하고 실제 프로그램은 이 자원을 어떤 방식으로 활용하여 작동하는지 알아야 함

프로세스와 스레드에 대한 정의

프로세스: 운영체제로부터 자원을 할당받은 **작업**의 단위.
스레드: 프로세스가 할당받은 자원을 이용하는 **실행 흐름**의 단위.

프로그램 → 프로세스 → 스레드

프로그램이란, **파일이 저장 장치에 저장되어 있지만 메모리에는 올라가 있지 않은 정적인 상태**를 말한다.
- 메모리에 올라가 있지 않은 : 아직 운영체제가 프로그램에게 독립적인 메모리 공간을 할당해주지 않았다
- 정적인 상태: 실행되지 않음

=> 코드 덩어리

실행 파일(프로그램)에게 의미를 부여하기 위해 프로그램을 실행해 보자
해당 파일은 컴퓨터 메모리에 올라가게 되고, 이 상태를 **동적(動的)인 상태**라고 하며 이 상태의 프로그램을 **프로세스**라고 한다

=>CPU를 통해 실행되고 있는 컴퓨터 프로그램

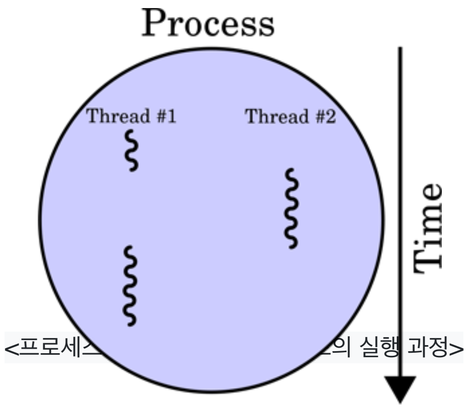
한 줄 요약: 프로그램은 코드 덩어리 파일, 그 프로그램을 실행한 게 프로세스.

프로그램 → 프로세스 → 스레드

프로세스 하나만을 사용해서 프로그램을 실행하기는 벅차게 되었다 => 여러 프로세스 ??= > 프로세스의 한계점
프로세스와는 다른 더 작은 실행 단위 개념이 필요하게 되었고, 이 개념이 바로 **스레드 => cpu의 수행 단위**

=> 스레드는 프로세스와 다르게 스레드 간 메모리를 공유하며 작동한다.
스레드끼리 프로세스의 자원을 공유하면서 프로세스 실행 흐름의 일부가 되는 것이다.

=> **한 줄 요약:** 스레드는 프로세스의 코드에 정의된 절차에 따라 실행되는 특정한 수행 경로다.



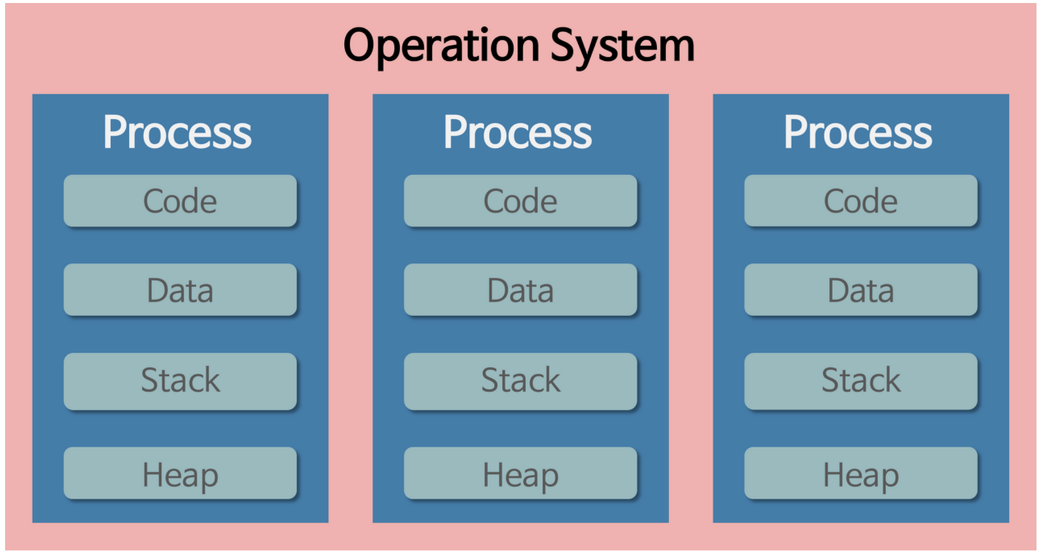
<프로세스 의 실행 과정>

<프로세스>

운영체제는 프로세스마다 각각 독립된 메모리 영역을,

Code/Data/Stack/Heap의 형식으로 할당해 준다.

각각 독립된 메모리 영역을 할당해 주기 때문에 프로세스는 다른 프로세스의 변수나 자료에 접근할 수 없다.



<스레드>

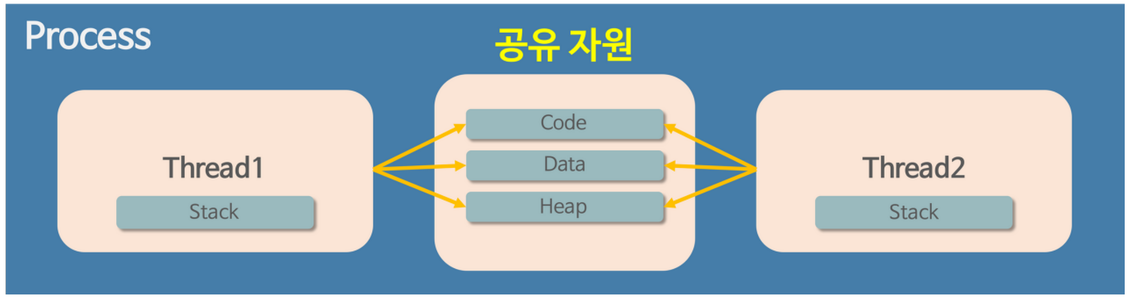
이와 다르게 스레드는 메모리를 서로 공유할 수 있다.

프로세스가 할당받은 메모리 영역 내에서

Stack 형식으로 할당된 메모리 영역은 따로 할당받고,

나머지 Code/Data/Heap 형식으로 할당된 메모리 영역을 공유한다.

따라서 각각의 스레드는 별도의 스택을 가지고 있지만 힙 메모리는 서로 읽고 쓸 수 있게 된다.



왜 멀티 프로세스로 할 수 있는 작업들을 굳이 하나의 프로세스에서 스레드로 나눠가며 할까?

- 운영체제가 시스템 자원을 효율적으로 관리하기 위해 스레드를 사용한다.
- 멀티 프로세스로 실행되는 작업을 멀티 스레드로 실행할 경우, 프로세스를 생성하여 자원을 할당하는 시스템 콜이 줄어들어 자원을 효율적으로 관리할 수 있다.
- 또한, 프로세스 간의 통신보다 스레드 간의 통신 비용이 적으므로 작업들 간 통신의 부담이 줄어든다. (처리비용 감소. 프로세스는 독립구조이기 때문)

그렇다면 무조건 멀티 스레드가 좋을까?

- 스레드를 활용하면 자원의 효율성이 증가하기도 하지만, 스레드 간의 자원 공유는 전역 변수를 이용하므로 동기화 문제가 발생 할 수 있으므로 프로그래머의 주의가 필요하다.

멀티 프로세스 => 안정성

1.각 프로세스들이 독립적으로 동작(자원이 서로 다르게 할당됨) 하기 때문에 안정적
2.멀티 스레드 보다 많은 메모리 공간과 CPU 시간을 차지
3.작업량이 많을수록 오버헤드가 발생하고 Context Switching으로 인한 성능 저하 우려
4.프로세스 간의 통신이 복잡 (IPC를 통해야 함)

멀티 스레드 => 효율성
스레드간 긴밀하게 연결되어 있기에,
1)공유된 자원으로 통신 비용이 절감되고,
2)메모리 사용이 효율적이다.
3)공유자원관리가 필요하고, context switching 비용이 작다

=> 하지만 스레드간 다 연결되어 있기 때문에 **한 스레드에 문제가 생기면 전체 프로세스에 영향**이 가게 된다.

* CPU에서 여러 프로세스를 돌아가면서 작업을 처리하는 데 이 과정을 Context Switching이라 한다.

멀티 태스킹(Multi-Tasking)

CPU는 한번에 하나의 프로세스만을 실행한다.

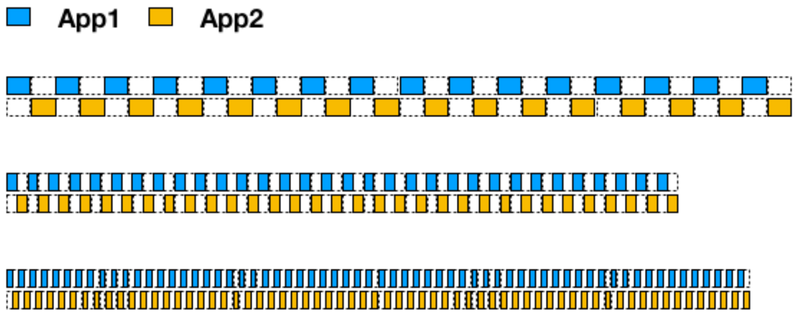
예전의 컴퓨터는 이러한 이유로 하나의 프로세스를 실행하고 있을 때, 다른 프로세스는 실행하지 못했다.

즉, 게임을 하면서 카톡을 할 수 없었다. 하지만 요즘 컴퓨터는 한번에 여러 프로세스를 실행할 수 있다.

=> 이것을 가능하게 해주는 것이 멀티 태스킹이다.

멀티 태스킹이란 하나의 CPU에서, 여러 응용 프로그램이 동시에 실행되는 것처럼 보이도록 하는 시스템이다.

=> 단위가 프로세스 => 스레드로 더 작아진 것

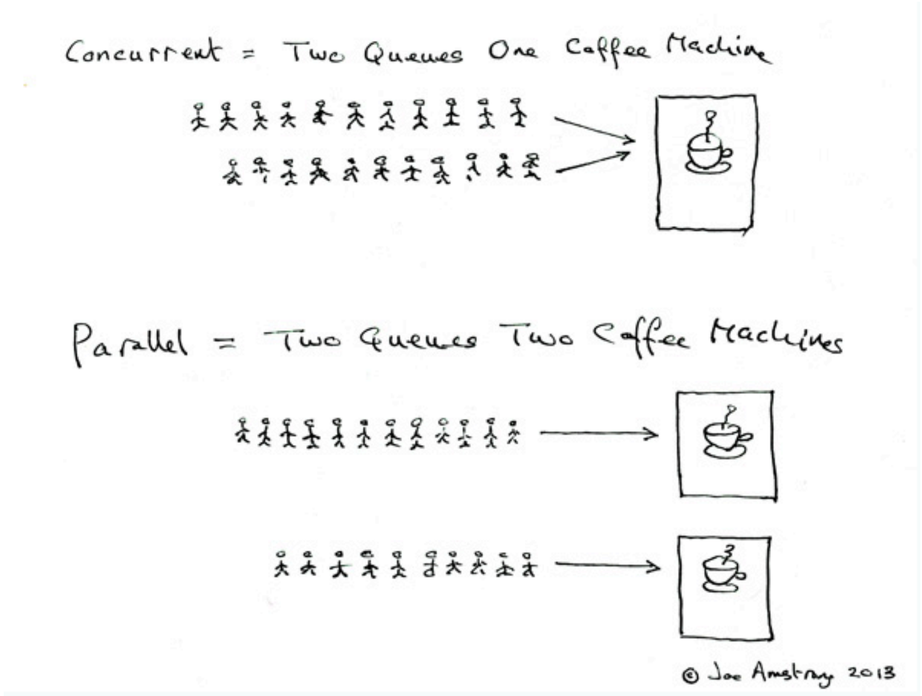


▶멀티 스레드의 실행방식: 병행성(Concurrency), 병렬성(Parallelism)

병행성(Concurrency) - 동시에 실행되는 것처럼 보이는 것.

- 보통 싱글 코어에서 멀티스레드를 동작시키기 위해 사용. 하지만 멀티코어에서도 실행 가능. (싱글, 멀티 둘다 실행 가능)
- 보기에는 스레드들을 동시에 병렬적으로 실행하는 것처럼 보이지만, 사실은 번갈아가면서 실행을 시켜 동시에 실행되는것처럼 보이게 함.

- Single Core
  - 물리적으로 병렬이 아닌 순차적으로 동작할 수 있다.
  - 실제로는 Time-sharing으로 CPU를 나눠 사용함으로써 사용자가 Concurrency를 느낄 수 있도록 한다.
- Multi Core
  - 물리적으로 병렬로 동작할 수 있다.



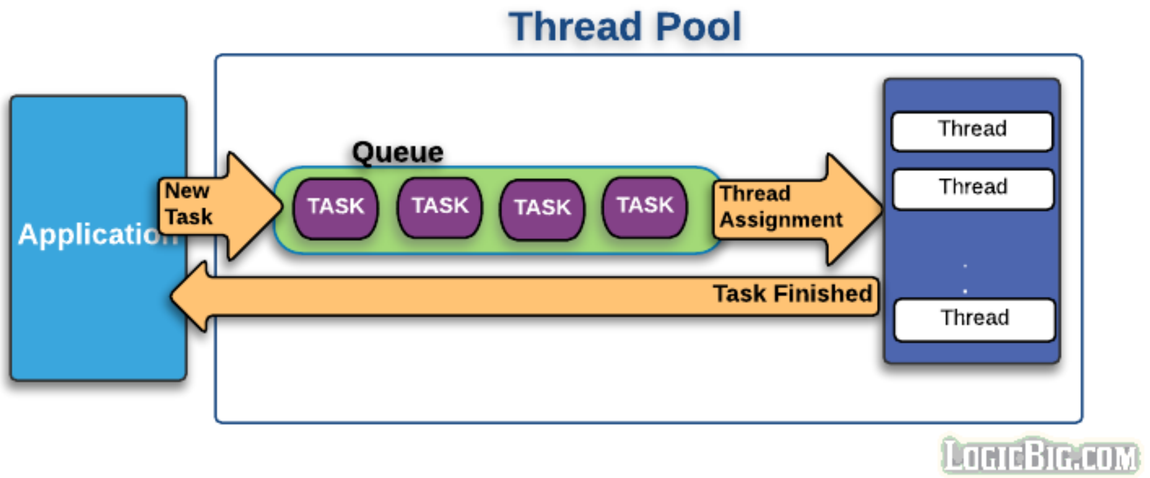
병렬성(Parallelism)

실제로 동시에 작업이 처리가 되는 것.

- Physical(Machine) Level에 속한다.
- 오직 Multi Core에서만 가능하다.

스레드풀 ?

- 병렬 작업 처리가 많아지면 스레드 개수가 중가되고, 그에 따른 스레드 생성과 스케줄링으로 인해 CPU가 바빠져 메모리 사용량이 늘어난다. 이는 애플리케이션의 성능 저하로 이어진다.
- 병렬 작업의 폭증으로 인한 스레드의 폭증을 막으려면 스레드 풀을 사용해야 한다.
- 스레드 풀은 작업 처리에 사용되는 스레드를 제한된 개수만큼 정해 놓고 작업 큐에 들어오는 작업들을 하나씩 스레드가 맡아 처리한다.
- 작업 처리가 끝난 스레드는 다시 작업 큐에서 새로운 작업을 가져와 처리한다.
- 따라서 작업 처리 요청이 폭증해도 작업 큐라는 곳에 작업이 대기하다가 여유가 있는 스레드가 그것을 처리하므로 스레드의 전체 개수는 일정하며 애플리케이션의 성능도 저하되지 않는다.



참고 자료

[os] 프로세스와 스레드의 차이

https://velog.io/@raejoonee/%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4%EC%99%80-%EC%8A%A4%EB%A0%88%EB%93%9C%EC%9D%98-%EC%B0%A8%EC%9D%B4

https://gmlwjd9405.github.io/2018/09/14/process-vs-thread.html

https://cocoon1787.tistory.com/688

동시성, 병렬성

https://blog.naver.com/qbxlvnf11/220837131449

스레드풀

https://cheershennah.tistory.com/170

멀티 태스킹의 한계

프로그램이 고도화 될 수록 멀티 태스킹은 몇가지 단점들이 있다.

1. 하나의 프로세스가 동시에 여러 작업을 수행하지 못함.

하나의 프로세스가 하나의 작업 밖에 하지 못하므로 속도의 아쉬움이 있다. 물론 하나의 프로그램을 여러 프로세스로 띄워서 성능을 높일 수 있겠지만 이 다음과 같은 한계는 극복하지 못한다.

2. 프로세스의 컨텍스트 스위칭은 무거운 작업이다.

컨텍스트 스위칭이 되면, 이전의 프로세스의 상태를 PCB에 보관하고, 새로운 프로세스의 상태를 PCB에서 읽어와 CPU 내에 있는 레지스터에 적재해야한다. 그러므로 이는 시스템에 많은 부담을 준다.

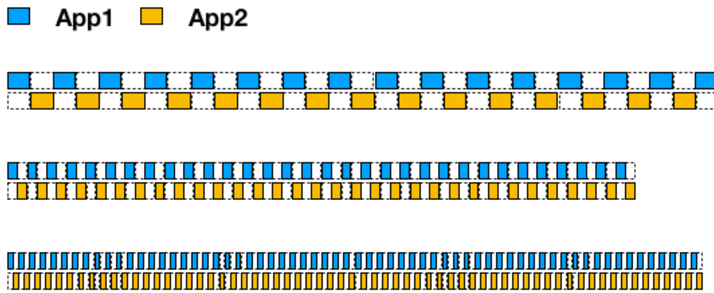
3. 프로세스끼리 데이터 통신이 까다로움.

프로세스는 독립적인 메모리 공간을 가진다. 만약 하나의 프로그램을 여러 프로세스로 실행 시킨다면 프로세스끼리 데이터를 통신을 해야하는 경우가 반드시 있다. 물론 IPC라는 프로세스 간 데이터 통신 기법을 사용할 수 있지만 이것도 굉장히 번거로운 작업이다.

4. 듀얼 코어 CPU가 등장했지만 잘 활용하지 못함.

하드웨어적으로는 듀얼 코어가 나왔지만 기존 멀티 태스킹 방식으로는 듀얼 코어의 장점을 100% 살리지 못함.

이러한 단점들을 보완하기 위해 나온 것이 스레드라는 개념이다.



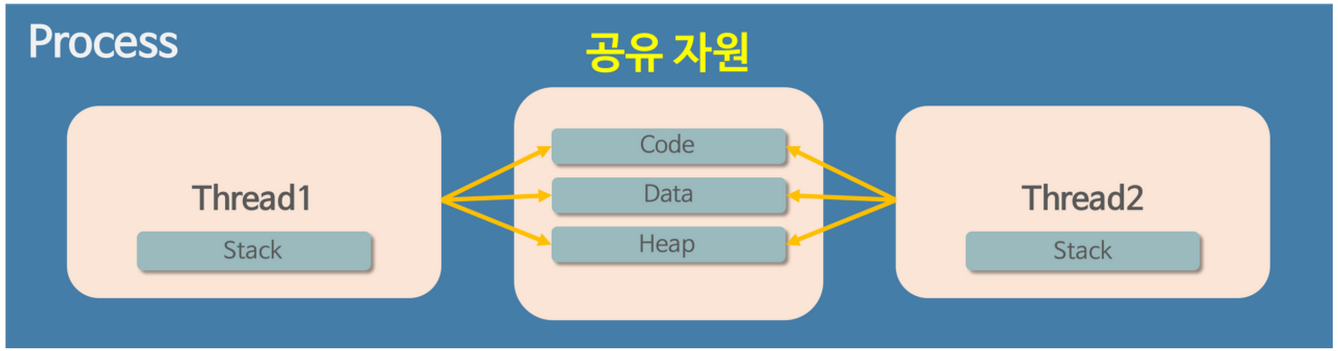
프로세스와 스레드의 중요한 차이

만약 한 프로세스를 실행하다가 오류가 발생해서 프로세스가 강제로 종료된다면, 다른 프로세스에게 어떤 영향이 있을까?
=> 공유하고 있는 파일을 손상시키는 경우가 아니라면 아무런 영향을 주지 않는다.

스레드는 Code/Data/Heap 메모리 영역의 내용을 공유하기 때문에
어떤 스레드 하나에서 오류가 발생한다면 같은 프로세스 내의 다른 스레드 모두가 강제로 종료된다.

이와 다르게 스레드는 메모리를 서로 공유할 수 있다.

프로세스가 할당받은 메모리 영역 내에서 Stack 형식으로 할당된 메모리 영역은 따로 할당받고,
나머지 Code/Data/Heap 형식으로 할당된 메모리 영역을 공유한다.
따라서 각각의 스레드는 별도의 스택을 가지고 있지만 힙 메모리는 서로 읽고 쓸 수 있게 된다.



스레드들이 프로세스의 Code/Data/Heap 메모리 영역을 공유하는 모습
(이미지 출처: Heee's Development Blog)

스레드(Thread)

적은 범위에서 얘기하면 **스레드(thread)**는 프로세스 내에서 실행되는 **작업 흐름의 단위**를 말한다. 더 확장되어 **CPU에서 실행되는 단위**이다.
위의 프로세스의 단점을 많이 보완하기위해 나왔고, **병렬적인 작업**을 가능하게 해준다.

스레드의 특징

1. 프로세스는 두 개 이상의 스레드를 가질 수 있다.=> 그래서 병렬적으로 작업을 처리할 수 있다.
2. 같은 프로세스에 있는 스레드들끼리 컨텍스트 스위칭이 가볍다.
스레드는 아래 사진처럼 data 등을 공유하고 있고 stack, register block 영역만 나눠져있기 때문에 그 부분만을 CPU 레지스터에 교체해주면 된다. 그러므로 프로세스 간의 컨텍스트 스위칭보다 비용이 훨씬 적다.
3. 스레드간 데이터 통신이 쉽다.
위 사진처럼 스레드들은 자신들이 속한 프로세스의 메모리 영역 (data, code 영역 등)을 공유하고 있기 때문에 통신할 데이터를 이 공유 영역에 올리기만 하면 된다. 프로세스처럼 IPC같은 특별한 기법이 필요하지 않다.