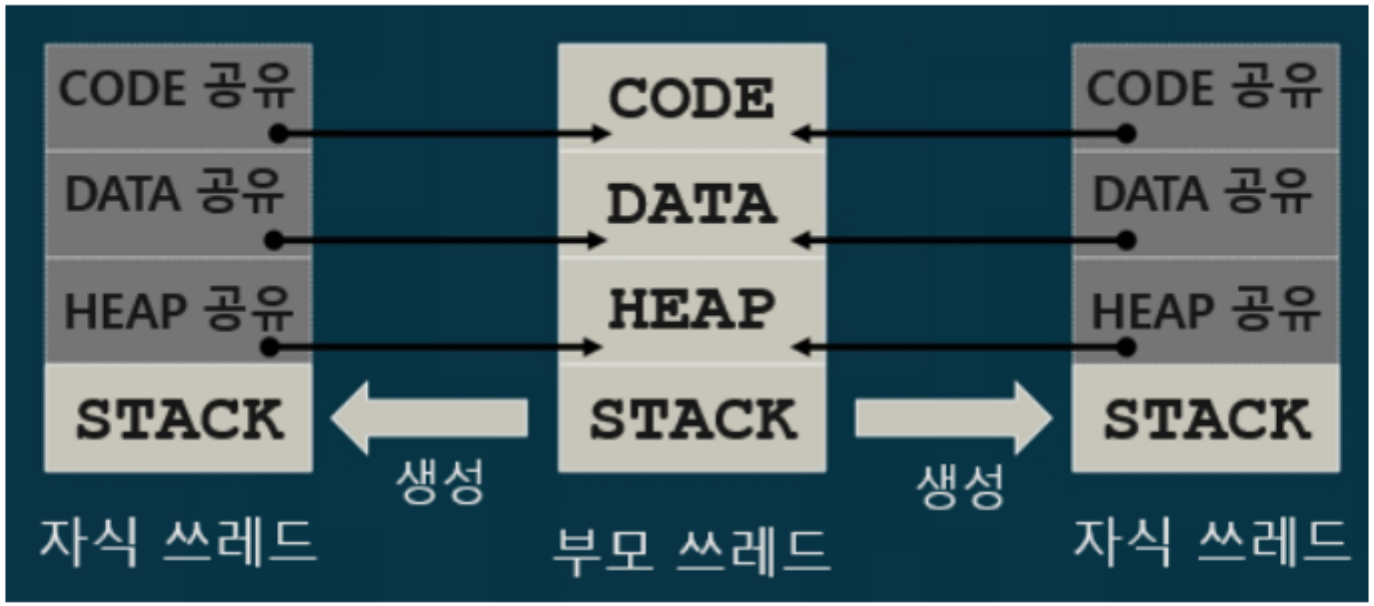
1. 프로세스와 스레드

- 프로세스는 초기에 하나의 시작 스레드를 가진다
- 스레드는 다른 스레드를 만들 수 있다
- 스레드 생성은 프로그래머가 지시한다
- 모든 스레드는 자신 고유의 스택을 가지고 있고, Data, Heap, Code 영역은 공유한다.
- 스레드는 CPU에서 하드웨어적으로 관리된다 (x86)
- ?? 옆 스레드의 stack은 접근 불가인가?? - 가능하지만 하지 않는다.

옛날에는 스레드란 개념이 없고 프로세스라는 개념만 있었다. 마치 옛날 수학 시간엔 1,2,3,4만 자연수였는데 요즘은 0까지 포함되는 것과 같다. 초반엔 스레드 없이 모든 것을 다 설명했다. 프로세스에는 스레드 하나씩 있다고 하는 게 좀 더 일관성이 있다. 프로세스는 스레드를 가지고 있다. 그러니까 스레드는 프로세스의 부분집합이다. 프로세스는 스레드 하고 다른 면목들이 모여서 프로세스를 이룬다.

컴퓨터를 실행할 때 컴퓨터를 켜놓고 보면 작업관리자에 프로세스가 많이 생기는 것을 알 수 있다. **이 프로세스는 누가 만든 것일까? 프로세스가 만든 것이다.** 어떤 실행되는 프로세스는 다른 프로세스가 CreateProcess같은 시스템 콜로 프로세스를 만든 것이다. 스레드도 마찬가지로 스레드가 여러 개 있는 게 멀티스레드인데, 그 스레드는 또 다른 스레드가 만든다. 맨 처음 태초에 하나의 스레드는 디폴트로 생기는 것이고, 다른 스레드는 CreateThread로 생기는 것이다. 즉, 이 함수를 9번 호출하면 스레드는 10개인 것이다.

착각하는 사람이 많은데, 실행파일이 있으면 그 실행파일 안에 이 파일은 4개의 스레드로 운영된다고 속성을 넣어 놓는 것이 아니다. 운영체제는 스레드가 몇 개 생길지 모른다. 실행해봐야 안다. 함수로 스레드를 만들 때만 생기기 때문이다.



스레드가 스레드를 만들면 무슨 일이 있을까? 이런 일이 있다. 시작 스레드, 즉 부모 스레드가 실행이 되고 코드 데이터 힙 스택이 사용된다. 처음 실행되는 하나의 스레드는 프로세스와 같다. 그런데 실행을 하면서 스레드 생성이라는 시스템 콜을 코드에서 호출하면 자식이 생기고, 또 호출하면 자식이 생긴다. 이랬을 때 자식 스레드는 어떤 환경에서 실행되느냐? **스택은 별도로 생성이 된다. 따로따로 스레드마다.** 그런데 코드, 데이터, 힙은 애매하다. 똑같은 메모리를 같이 쓰게 된다. 그 이야기는 부모스레드가 실행하는 프로그램과 자식 스레드가 실행하는 프로그램은 같은 프로그램이다. 코드를 복사해서 놓고 쓰는 것이 아니라, 그냥 한 장소에 있는 것을 자식과 부모가 같이 쓰는 것이다. **그래서 자식 스레드의 코드,** 데이터, **힙은 실체가 없고 모두가 공유하여 같이 쓰는 것이다.**

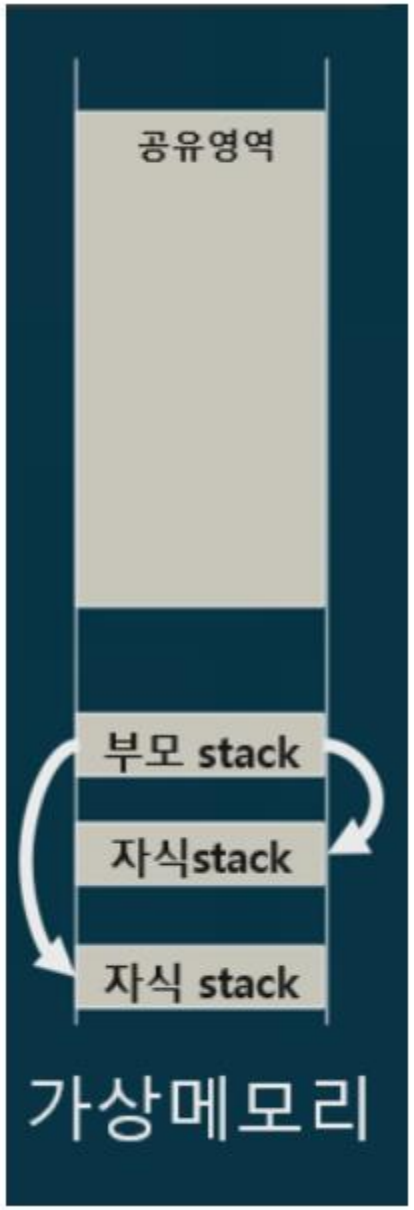
가상메모리 공간이면 원래 공유 영역에 코드, 데이터, 힙이 있다. **스레드를 만들면 자식의 스택만 새로 생기고 다른 건 공유를 해서 같이 돌아간다. 이게 스레드의 특징이다. 프로세스는 부모랑 완전히 독립**되어서 분리되는데 스레드는 아니다.

스레드는 CPU에서 하드웨어적으로 관리되는데 CPU안에 자료구조가 있다. 사실 CPU 안은 아니고 메모리 안에. 부모 스레드가 실행하고 데이터에 전역 변수 내용을 고친 뒤, 자식이 데이터를 읽으면 바뀐 값이 읽힌다. 모든 스레드가 마찬가지. 굳이 바꾼 다음에 '나 바뀌었어' 하고 알려주는 것이 아니다. 복사하는 것이 아니기 때문이다. 그러나 스택은 안 바뀐다. 지역변수의 수정은 반영되지 않는다는 뜻이다.

스레드 사이에 데이터를 주고받아야 한다. 그러려면 어떡해야 하는가? 메모리를 통해서 서로 데이터를 쓰고 읽고 해야한다. 데이터를 읽고 '그렇다면 나의 값은 이것이다' 하고 데이터를 쓰고 읽고 주고받고... 이것이 전역 변수 데이터 영역이다.

스택은 무엇인가? 학생 때 부모님과 함께 산다고 잠까지 같이 자지는 않는다. 독립된 방에서 잤다. 스레드도 마찬가지이다. 모두 공유되면 똑같은 일밖에 할 수 없다. 공유가 되긴 하는데 똑같은 일을 하게된다면 의미가 없다. 그래서 **개인작업공간을 분리시킨 것이 스택이다.** 지역변수는 서로 볼 수 없고 수정할 수 없다. 그러나 전역변수는 공유할 수 있다.

더 자세히 들어가면, 데이터 영역에 특별한 키워드를 쓰면 독립적으로 쓸 수 있다. 그러나 스택은 공유 그런 것이 없는데 억지로 자식의 방에 들어가서 컴퓨터를 살펴보겠다고 한다면 할 수는 있다. 그러나 좋은 일이 아니다. 사람들 사이에서는 관계가 틀어지고 프로그램 사이에는 버그가 나기 쉽다.



2. 프로세스에 대한 스레드의 장점과 단점

- 장점
  - 생성 Overhead가 적다
  - Context Switch Overhead가 적다 (Virtual memory (TBL교체 오버헤드))
  - Thread 간의 통신이 간단하다.
- 단점
  - 하나의 스레드에서 발생한 문제가 전체 프로세스를 멈추게 한다
  - 디버깅이 어렵다

프로세스는 할 수 없고 스레드만 할 수 있는 것이 있는가? 있다고 할 수 있지만, 절대 그런 것은 아니다. 절대 스레드만 할 수 있고, 절대 프로세스는 할 수 없다 이런 것은 없다. 스레드가 할 수 있는 것은 프로세스도 할 수 있고, 프로세스가 할 수 있는 것은 스레드도 다 할 수 있다.

그럼 다 멀티프로세스로 해서 성능을 올리지 왜 멀티스레드를 쓰는가?
이런 **장점**이 있다. **첫번째로 생성 오버헤드가 적다.** 스레드를 만들 때 시스템 콜을 해야 하는 데 걸리는 시간이 프로세스보다 적다. 금방 만들 수 있다. 오버헤드는 자원, 즉 메모리를 줘야 한다. 메모리를 할당시켜줘야 하고 파일 i/o자원도 할당을 해주어야 한다. 왜냐하면 printf scanf 하는 것도 자원이기 때문이다.
그러나 스레드는 그렇지 않다. 같은 집에서 살기 때문에 새 집, 즉 메모리를 줄 필요가 없다. 그래서 오버헤드가 적다.

둘째로 스레드와 스레드 간의 context switch가 가볍다. 빠르다는 이야기이다. 왜? virtual memory이기 때문이다. 가상 메모리 맵핑을 딱딱 바꾸어야 한다. 그러나 스레드는 그럴 필요가 없다. 스레드는 가상 메모리 맵핑 그대로 공유하니까 교체하지 않고 그대로 실행하면 된다. 교체 오버헤드도 적고, 스위치를 하면 실행되는 프로세스가 다른 프로세스를 실행하니까 전에 프로세스가 사용하는 것은 완전히 다른 메모리이다. 이럴 경우 캐시 미스(Cache Miss)가 난다. 여태까지 사용한 데이터는 날아가고 새로 해서 캐시 미스가 계속 난다. 그런데 멀티스레드를 하면 캐시 미스가 덜 일어난다. 그래서 스위치 오버헤드가 적다.

셋째로 가장 큰 이유는 스레드간 통신이 간단하다. 얼마나 간단한가?
다른 스레드간 통신을 하고 싶다 그러면 IPC(Inter Process Communication)이 된다. 프로그래밍 언어에는 없고 운영체제마다 다 다른 API를 제공한다. 윈도우는 윈도우에 있는 IPC를 써야 하고, 리눅스는 리눅스에 있는 IPC를 써야 한다. 프로세스끼리 데이터를 주고받고 싶으면 소켓을 만들어서 소켓 IPC를 해야 한다. 그러면 같은 컴퓨터에 프로세스가 있지 않아도 된다. 다른 컴퓨터에 있는 프로세스끼리도 통신할 수 있다. 그런 엄청난 장점이 있지만, 오버헤드가 크고 느리다.

그렇다면 스레드 통신은 어떻게 하냐? 통신이 아니다. 스레드A가 스레드 B에게 3이라는 데이터를 보내고 싶으면 A는 3을 보낸다. 그럼 A에선 num = 3; 그리고 B에서는 cout << num; 하면 받을 수 있다. **여러 세팅이 필요한 것이 아니다.**

그러나 **단점**도 있다.
첫번째로 멀티 프로세스로 했을 때 한 프로세스가 죽으면 다른 프로세스들이 다 죽지 않는다. 그리고 프로세스가 실행될 때 그 옆 프로세스가 죽으면 다시 깨우고 연결해서 실행하면 된다. 그러나 스레드는 그렇지 않다. **스레드가 실행되다 죽는다면? 그러면 스레드 전체가 죽는다**. 어떤 스레드에서라도 크러쉬가 나서 종료한다고 하면 전체 스레드가 죽는 것이다. 다시 살릴 방법이 없다. 문제가 발생했을 때 파급효과 페널티가 치명적이다. 이것이 단점이다.

두 번째 단점은 디버깅이 어렵다는 점이다. 왜 어려울까? 프로그램은 하나인데 이 안에서 프로그램 안에 여러 군데가 동시에 실행되고 있기 때문에, 에러가 나면 이 위에서 무슨 일이 벌어졌나 살펴보게 된다. 그런데 그게 불가능하다. 왜냐하면 내가 온 길은 보이지만, 옆에 스레드가 어디를 실행하고 있는지 보이지 않는다. 억지로 볼 수는 있겠지만 명령어가 어떤 순서로 일어나는지 알 수 없다. A에 3이 들어가며 안되는데 들어갔다? 그럼 누가 3을 넣었는지 알 수가 없다. 이게 제일 큰 단점이다. 이것만 아니라면 멀티스레드가 널리 퍼져서 C나 C++ 배울 때 쓰라고 했을 텐데 그러지 못하는 이유는 디버깅이 너무 어렵기 때문이다.