**프로세스** : 프로그램을 구동하여 프로그램 자체와 프로그램의 상태가 메모리 상에서 실행되는 작업 단위를 지칭한다. 즉, 저장장치에 저장되어 있는 정적인 상태의 프로그램이 실행을 위해 메모리에 올라온 동적인 상태가 된 것을 말한다.

**스레드** : 프로세스의 코드에 정의된 절차에 따라 CPU에 작업 요청을 하는 실행 단위이다.

**프로세스와 스레드의 차이점**

|  |  |  |
| --- | --- | --- |
|  | 프로세스 | 스레드 |
| 전환 속도 | 스레드 보다 느리다 | 프로세스 보다 빠르다 |
| 메모리 | 독립적 실행 & 개별 메모리 차지 | 프로세스의 내의 메모리 공유 |

**스레드 사용 이유 및 스레드 모델**

스레드는 CPU가 여러 개일 경우에 각각의 CPU가 스레드 하나씩을 담당하는 방법으로 속도를 높일 수 있으며 실제 시간상 동시에 수행될 수 있다. 즉, 효율성과 응답성의 향상, 자원 공유의 이점이 있다.

* **사용자 레벨 스레드(= 다대일 스레드 맵핑)**
* 사용자 프로세스 내에 여러 개의 스레드가 커널의 스레드 하나와 연결
* 라이브러리가 직접 스케줄링을 하고 작업에 필요한 정보를 처리하기 때문에 문맥 교환이 필요 없음
* 커널 스레드가 입출력 작업을 위해 대기 상태에 들어가면 모든 사용자 스레드가 같이 대기하게 됨
* 한 프로세스의 타임 슬라이스를 여러 스레드가 공유하기 때문에 여러 개의 CPU를 동시에 사용할 수 없음
* **커널 레벨 스레드(= 일대일 스레드 맵핑)**
* 하나의 사용자 스레드가 하나의 커널 스레드와 연결
* 독립적으로 스케줄링이 되므로 특정 스레드가 대기 상태에 들어가도 다른 스레드는 작업을 계속할 수 있음
* 커널 레벨에서 모든 작업을 지원 -> 멀티 CPU 사용 가능
* 하나의 스레드가 대기 상태에 있어도 다른 스레드는 작업을 계속할 수 있음
* 보안에 강하고 안정적으로 작동 but 문맥 교환 시 오버헤드로 느리게 작동
* **멀티 레벨 스레드(= 다대다 스레드 맵핑)**
* 사용자 레벨 스레드 + 커널 레벨 스레드 혼합
* 커널 스레드가 대기 상태에 들어가면 다른 커널 스레드가 대신 작업을 하여 사용자 레벨 스레드보다 유연하게 작업을 처리할 수 있음
* 커널 레벨 스레드를 같이 사용하기 때문에 문맥 교환 시 오버헤드로 느리게 작동
* 상황에 따라 선택적으로 스레드 사용

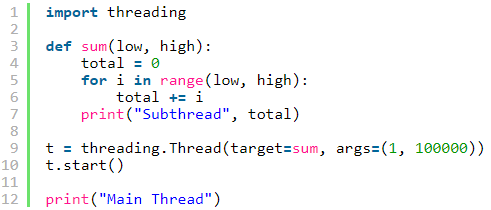
Ex) 빠르게 움직여야 하는 스레드 -> 사용자 레벨 스레드로 작동

안정적으로 움직여야 하는 스레드 -> 커널 레벨 스레드로 작동

**Python에서 프로세스 및 스레드의 사용법**

* Python에서 threading 혹은 thread 모듈을 이용하여 스레드를 사용 가능하며, 일반적으로 threading 모듈을 이용하고 thread 모듈은 거의 사용하지 않는다.
* daemon속성을 True or False로 설정 가능하다. 데몬스레드는 백그라운드에서 실행되는 스레드로 메인스레드가 종료되면 즉시 종료된다. 기본값은 False이다.

Ex) t1.daemon = True



<스레드 사용 예시>

* Python에서 multiprocessing을 이용하여 프로세스의 이용이 가능하다.
* 멀티 프로세싱을 활용하면 별도의 프로세스를 생성 후 병렬처리하기 때문에 보다 빠른 응답처리 속도를 기대할 수 있다.



<프로세스 사용 예시>

**Python에서 Thread와 Process의 차이**

파이썬 프로그램은 기본적으로 Single Thread에서 실행된다. 파이썬은 GIL(Global Interpreter Lock)때문에 특정 시점에서 하나의 파이썬 코드만을 실행하게 되며, 이로 인해 파이썬은 다중 CPU환경에서 동시에 여러 파이썬 코드를 병렬로 실행할 수 없다. 따라서 thread는 병렬 처리에 적합하지 않으며, thread를 1개만 사용하였을 때가 2개를 사용한 경우보다 속도가 빠르다. 이런 GIL을 피해서 해결하는 것이 multiprocessing이며 병렬 처리를 통해 속도를 비약적으로 향상시킬 수 있다.