**프로세스**란 운영체제 입장에서 하나의 작업 단위다.

**프로그램과 프로세스의 차이**

**프로그램**은 하드디스크 같은 **보조기억장치에 저장된 절차(코드+데이터)**다. 아직 실행되지 않은 정적인 상태이다. 폰 노이만 구조의 컴퓨터에서 프로그램을 실행하기 위해서는 메모리상에 올라와 있어야 한다. **프로그램을 메인 메모리 위에 올려 실행한 것**이 **프로세스**다. 즉 실행중인 프로그램이 프로세스다.

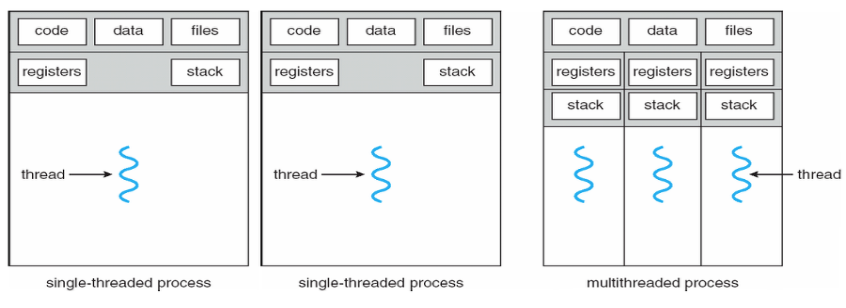
**스레드**란 프로세스의 코드에 정의된 절차에 따라 CPU에 작업을 요청하는 실행 단위, 흐름의 단위다. **프로세스와 스레드의 차이**는 운영체제 입장에서의 작업 단위는 프로세스이고 CPU 입장에서의 작업 단위는 스레드다.

**멀티태스크와 멀티스레드**

**멀티태스크**는 CPU가 시분할 시스템으로 작업을 처리하는 것으로, 이때 전달받은 프로세스는 하나의 스레드로 이루어진 프로세스다. 따라서 사실상 하나의 스레드를 전달받은 것이고, 이 스레드를 시분할 시스템(각각 처리하는 시간을 잘게 쪼개서)으로 처리한다. 때문에 여러 흐름이 동시에 진행되는 것처럼 처리된다.

하지만 프로세스를 여러 개 만들어 처리하는 방법은 코드 영역과 데이터 영역의 중복이 발생해 메모리 낭비를 발생시키게 된다. 멀티스레드를 사용할 경우 이 문제를 해결할 수 있다.

**멀티스레드**는 하나의 프로세스 안에서 작업을 여러 개의 스레드로 분할하는 방법이다. 즉, 절차지향의 프로그램보단 객체 지향적인 프로그램에 어울리는 방식이라고 할 수 있다.



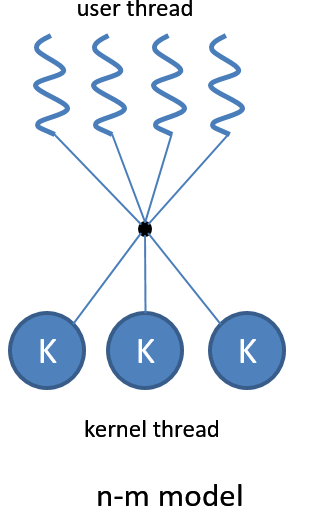
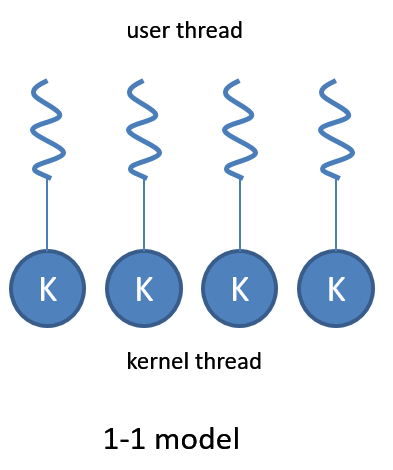
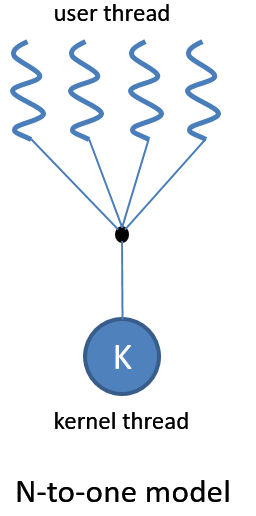
이 멀티스레드 방식은 스레드들이 프로세스의 정적 영역인 코드나 정적 데이터들을 공유하게 된다. 스레드에 의해 바뀌는 부분은 프로세스의 동적 영역(레지스터 값, 힙, 스택 등)이다. 따라서 중복되는 데이터들을 줄여 메모리 낭비를 최소화할 수 있다. 게다가 속도 면에서도 여러 개의 프로세스를 시분할 방식으로 처리하는 것 보다 멀티스레드를 처리하는 것이 효율적이고 훨씬 빠르다.

게다가 2개 이상의 CPU를 가진 컴퓨터에서 멀티스레드를 사용하면 다중 CPU가 멀티스레드를 동시에 처리하여 CPU 사용량이 증가하고 프로세스의 처리 시간이 단축된다. 단, 멀티스레드의 경우 모든 자원을 공유하기 때문에 한 스레드에 문제가 생기면 전체 프로세스에 영향을 미칠 수 있다.

**멀티스레드 모델**

커널 스레드 : 커널이 직접 생성하고 관리하는 스레드

사용자 스레드 : 라이브러리에 의해 구현된 일반적인 스레드



**사용자 레벨 스레드**는 라이브러리에서 커널이 지원하는 스케줄링, 동기화, 정보처리 등을 구현하여 처리한다. 사용자 프로세스 내에는 여러 개의 스레드가 존재하지만, 커널의 입장에서는 이를 하나의 프로세스로 인식하기 때문에 커널의 스레드 하나와 연결된다. 때문에 **1 to N 모델** 이라고도 한다.

**장점** : 라이브러리 내에서 스레드를 직접 스케줄링하고 제어하기 때문에 커널 모드로의 문맥 교환(context switching)을 할 필요가 없어지게 되어 처리 속도가 빨라진다. 커널 모드에서 스레드를 스케줄링 하는 것이 아니기 때문에 라이브러리만 있으면 어떠한 운영체제에서도 커널에 종속되지 않고 프로세스를 실행할 수 있다. = 이식성이 좋다.

**단점** : 이 모델에는 치명적인 단점이 있다. 커널이 프로세스의 멀티 스레드를 인식하지 못하고 하나의 프로세스로 인식하기 때문에 여러 개의 CPU를 가지고 있는 커널이더라도 스레드를 나누어 처리하지 못한다. 즉 멀티 프로세싱이 불가하다. 또한 여러 개의 스레드를 커널이 하나의 스레드처럼 처리하기 때문에 사용자 스레드 하나가 I/O 처리를 위한 블록상태에 들어가면 나머지 모든 사용자 스레드들도 블록된다. 보안에 있어서도 커널 모드보다 유저 모드의 라이브러리에서 구현되는 것이 더 취약하다.

**커널 레벨 스레드**는 커널이 멀티 스레드를 지원하므로 하나의 사용자 스레드가 하나의 커널 스레드와 연결되는 **1 to 1 모델**이다. 각각의 사용자 스레드를 커널 스레드가 맡아 직접 스케줄링하고 제어하기 때문에 병렬적인 처리가 가능하다. 현재 대부분의 운영체제에서 지원하고 사용하는 모델이다.

**장점** : 특정 스레드가 블록상태에 들어가더라도 다른 작업을 계속할 수 있다. 멀티 프로세싱이 가능하다. 커널이 제공하는 보안 서비스를 받을 수 있다.

**단점** : 유저 모드와 커널 모드를 왔다 갔다 하면서 문맥 교환을 해야 하기 때문에 오버헤드가 발생한다. 또한 상대적으로 커널이 하는 일이 많아져 부담을 주기 때문에 시스템 성능이 떨어질 수 있다. (하지만 지금의 컴퓨터 성능으로는 이러한 단점을 충분히 커버할 만큼이기 때문에 커널 스레드를 선호한다)

**Python에서 Thread 사용법**

**threading 모듈**에서 threading.Thread() 함수를 호출하면 Thread 객체를 반환한다. 그런 다음 Thread 객체의 **start() 메소드**를 호출하면 스레드 실행 가능.

import threading

import time

def myThread(index):

for i in range(index):

print("My thread")

time.sleep(2)

t = threading.Thread(target = myThread, args=(3,))

t.start()

for i in range(3):

print("main")

time.sleep(1)

print("---end---")

threading.Thread()에 target 인자에 실행할 메소드 이름을 넣어주고, 메소드의 인자는 args에 튜플 형식으로 전달해준다. 그런 다음 start() 메소드를 호출하면 스레드가 실행된다. 이 때 동시에 main()에서도 어떤 문장을 출력하는 코드를 넣어주고, time.sleep()으로 대기 시간을 설정해 주면 두 함수를 멀티 스레드처럼 작동하는 것을 눈으로 확인할 수 있다. 출력 결과는 myThread가 다 실행되고 main()이 실행되는 것이 아니라 time.sleep()에 따라 왔다 갔다 하면서 실행된다.

위의 경우 부모 스레드인 main이 먼저 종료되어 버리는데, 이를 방지하기 위한 방법으로 join() 메소드를 사용할 수 있다.

혹은 클래스를 하나 만들어서 threading.Thread를 상속받게 한 후, 그 안에서 run() 메소드를 재정의 하는 방법도 있다. (run() 메소드는 스레드가 실제 실행하는 메소드로 start()메소드에서 내부적으로 run()을 호출한다.)