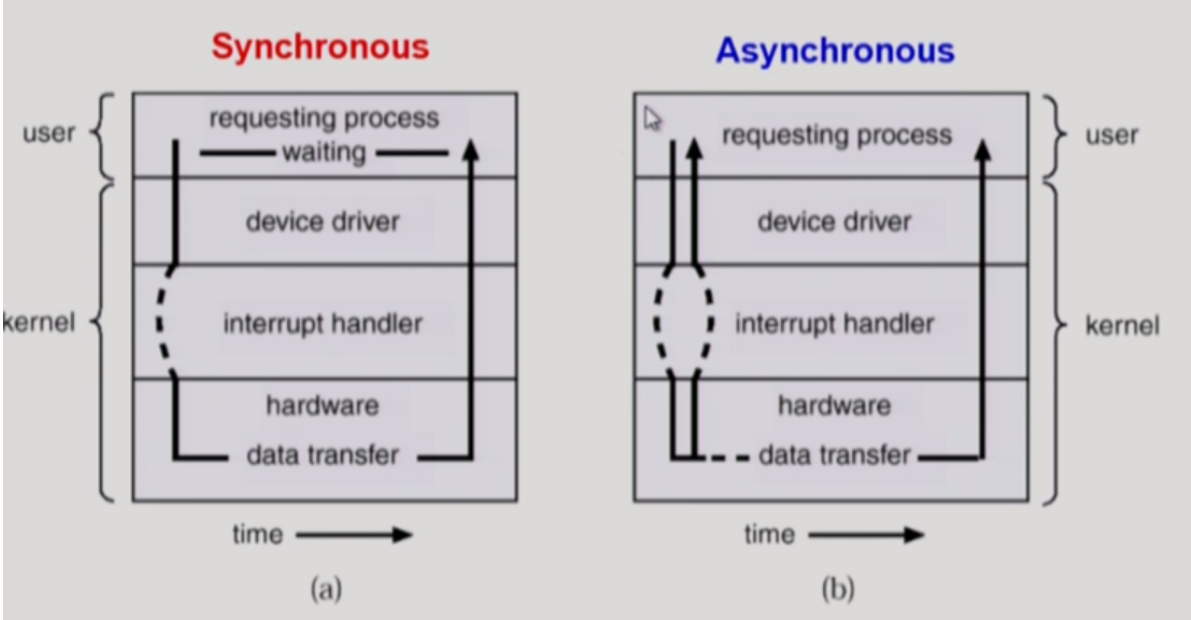
동기식 입출력과 비동기식 입출력의 차이



- 프로세스 입장에서 먼저 보면 프로세스가 입출력 요청을 먼저 했으면 본인이 집적한게 아니라 운영체제를 통해서 하기 때문에 사용자 프로세서는 운영체제에게 입출력 요청을 함
- 입출력은 오래 걸려서 수행 그 동안 입출력을 요청하는 프로세스가 입출력이 끝날 때 까지 기다려야하면 그것은 동기식 입출력

- 사용자 프로세스가 입출력 요청을 한 다음 입출력이 진행되는 동안 그 프로세스가 cup가 instruction을 실행하면 비동기식 입출력이다
- 즉 동기식과 비동기식을 비교하면 프로세스가 입출력이 진행되는 동안에 cpu를 가지고 있든 아니든은 중요하지 않지만 입출력을 실행하지않고 기다리면 입출력이 끝나는것을 보고 instruction을 실행하면 동기식 입출력 요청을 해 두고 언젠가 그 입출력이 완료가 되겠지만 그 이전에 바로 instruction을 실행하게 되면 비동기식

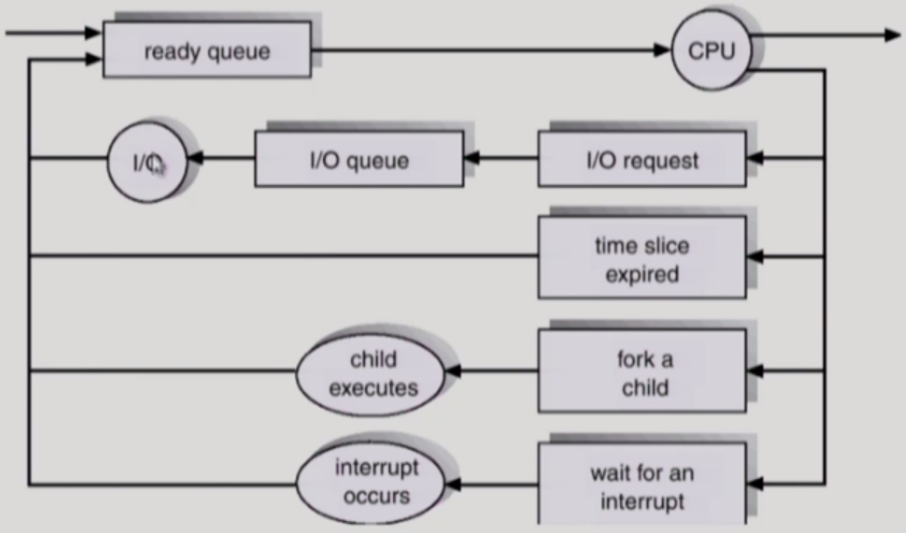
동기식 구현 방법에 관한 설명

동기식 입출력과 비동기식 입출력

- 동기식 입출력 (synchronous I/O)
  - ✓ I/O 요청 후 입출력 작업이 완료된 후에야 제어가 사용자 프로그램에 넘어감
  - ✓ 구현 방법 1
    - I/O가 끝날 때까지 CPU를 낭비시킴
    - 매시점 하나의 I/O만 일어날 수 있음
  - ✓ 구현 방법 2
    - I/O가 완료될 때까지 해당 프로그램에게서 CPU를 빼앗음
    - I/O 처리를 기다리는 줄에 그 프로그램을 줄 세움
    - 다른 프로그램에게 CPU를 줌
- 비동기식 입출력 (asynchronous I/O)
  - ✓ I/O가 시작된 후 입출력 작업이 끝나기를 기다리지 않고 제어가 사용자 프로그램에 즉시 넘어감

- 동기식은 프로세스가 I/O요청을 하고 완료될때까지는 일을 못함 근데 그 못하는 동안 CPU를 가지고 기다리면 구현 방법 1
- 어짜피 일 못할거 CPU는 다른 프로세스한테 당장 CPU가 있다면 일을 할 수 있는 프로세스한테 넘겨주자 그럼 구현 방법 2, 좀 더 효율적으로 사용하기 위해서 사용

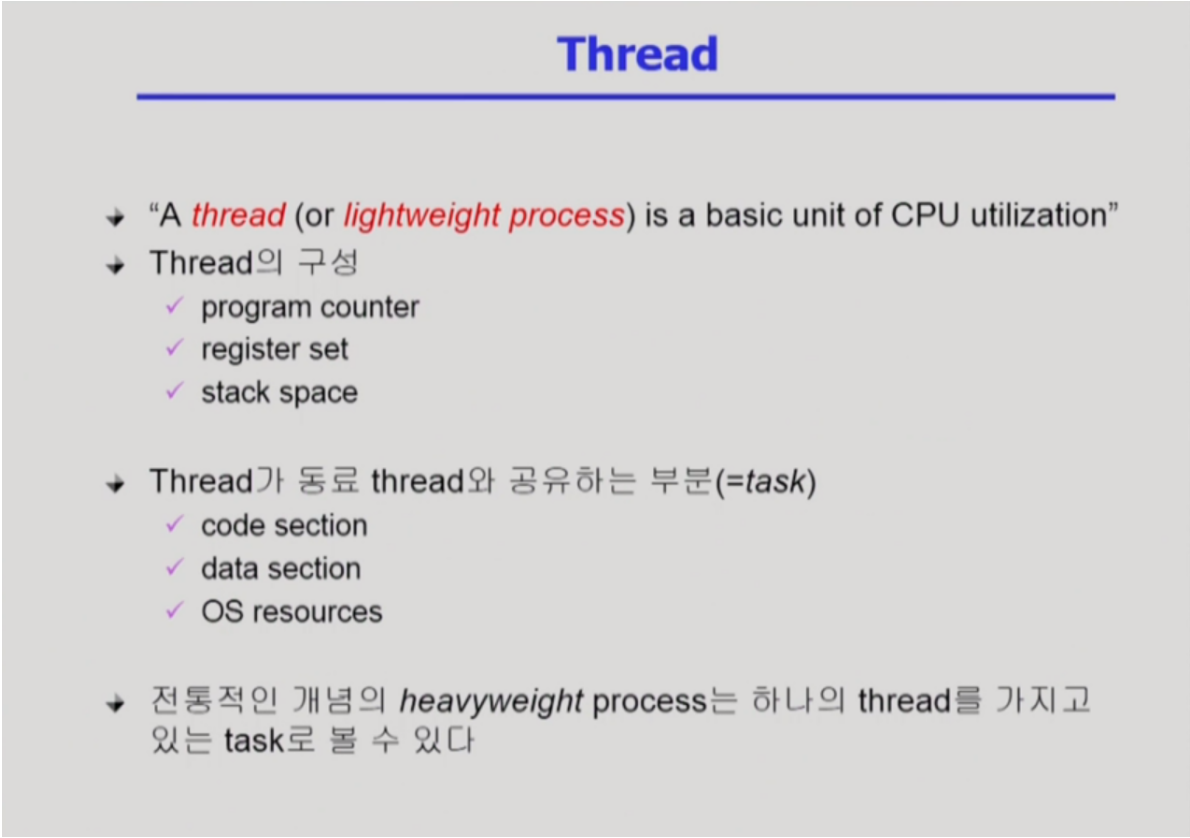
프로세스 스케줄링 큐의 모습

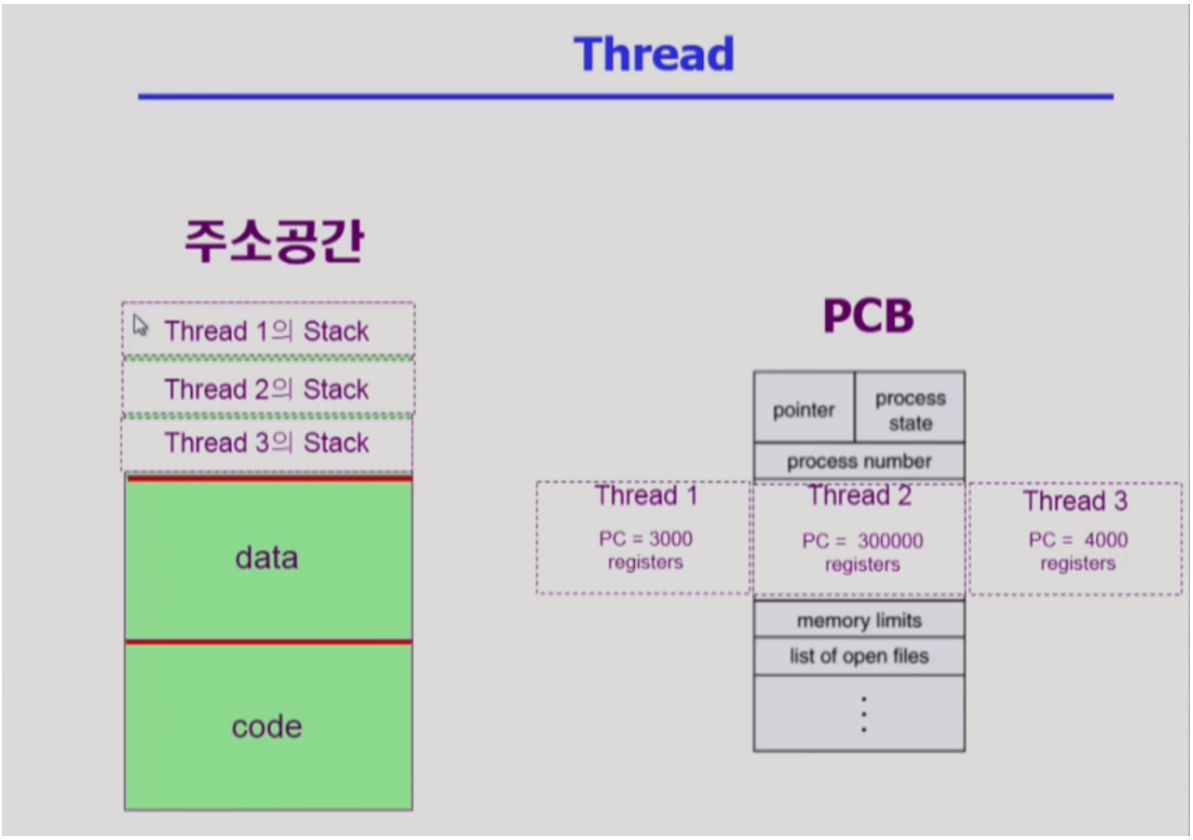


- CPU를 가지고 프로세스가 실행이 되다가 I/O 요청을 하면 I/O큐에서 기다리고 수행 끝나면 다시 CPU를 기다리는 ready큐에 기다림
- 마지막 줄도 사실 같음
- cpu를 가지고 instruction을 실행하다가 오래걸리는 작업을 하면 보통 끝날 때 interrupt를 통해 알려준다. I/O작업을 하러가면 interrupt를 기다리게 되고 I/O가 끝나서 interrupt가 발생하게되면 다시 cpu를 기다리는 ready 큐에서 기다린다.

Thread(쓰레드)

- **쓰레드는 프로세스 내부에 CPU 수행 단위가 여러개 있는 경우 쓰레드라고 부름**





- 코드, 데이터, 스택으로 구성된 주소공간이 프로세스마다 구성된다.
- 이 프로세스 하나를 관리하기 위해 운영체제 내부에 PCB를 둔다. PCB는 프로세스의 상태, ID, 프로그램카운터(메모리의 어느 부분을 실행하고 있는지를 알려주는 레지스터,) 등을 나타낸다.
- 즉 프로세스가 하나 주어지면 그 프로세스만의 코드 데이터 스택이 주어지고 PCB에서 프로그램카운터로 어느 메모리의 어느부분이 실행하고 있는지 알려준다.
- 어떤 동일한 일을 하는 프로세스가 여러개 있다고 하면 별도의 프로세스로 만드면 메모리 주소공간이 여러개 만들어질것이다. 그럼 낭비가 된다. 그래서 같은일을 하는 프로세스를 여러개 하고 싶다면 메모리 공간을 하나만 하고 프로세스마다 다른 부분의 코드를 실행하게 하면 된다. 이것이 쓰레드의 개념이다.
- 즉 프로그램 카운터만 여러개 둔다. 프로세스 하나에 CPU 수행 단위만 여러개 두는 것을 쓰레드라고 한다.
- 코드 데이터 부분은 다른 쓰레드와 공유하지만 CPU수행과 관련된 프로그램 카운터, 레지스터, 스택은 별도로 가지고 있다.

쓰레드의 장점

- 다중 스레드로 구성된 태스크 구조에서는 하나의 서버 스레드가 **blocked(waiting)** 상태인 동안에도 동일한 태스크 내의 다른 스레드가 실행(runnging)되어 빠른 처리를 할 수 있다.
- 동일한 일을 수행하는 다중 스레드가 협력하여 높은 처리율(throughput)과 성능 향상을 얻을 수 있다
- 스레드를 사용하면 병렬성을 높일 수 있다