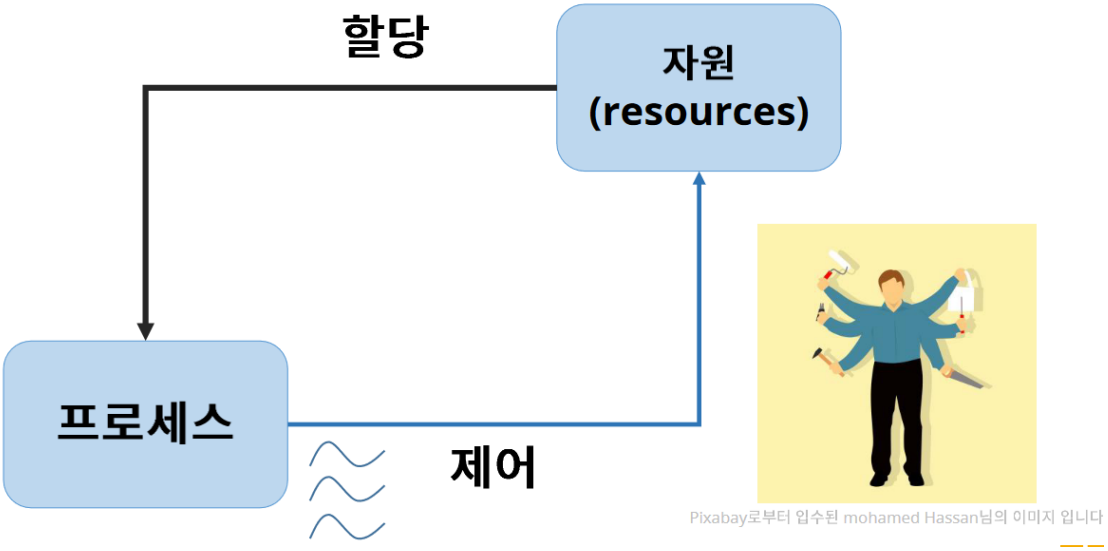
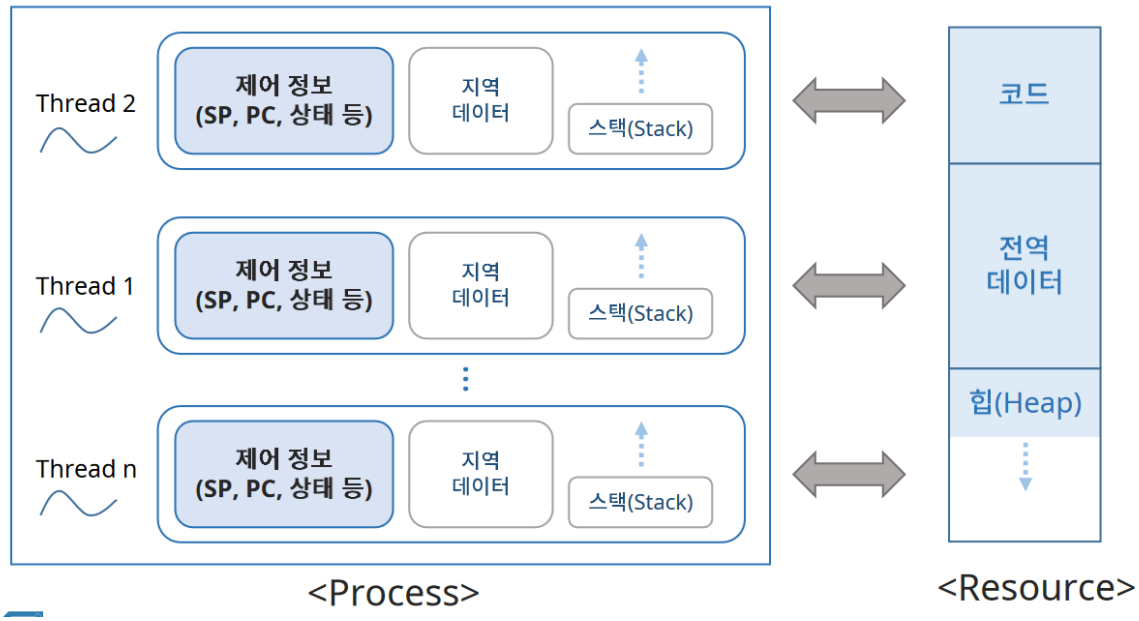
04. Thread Management

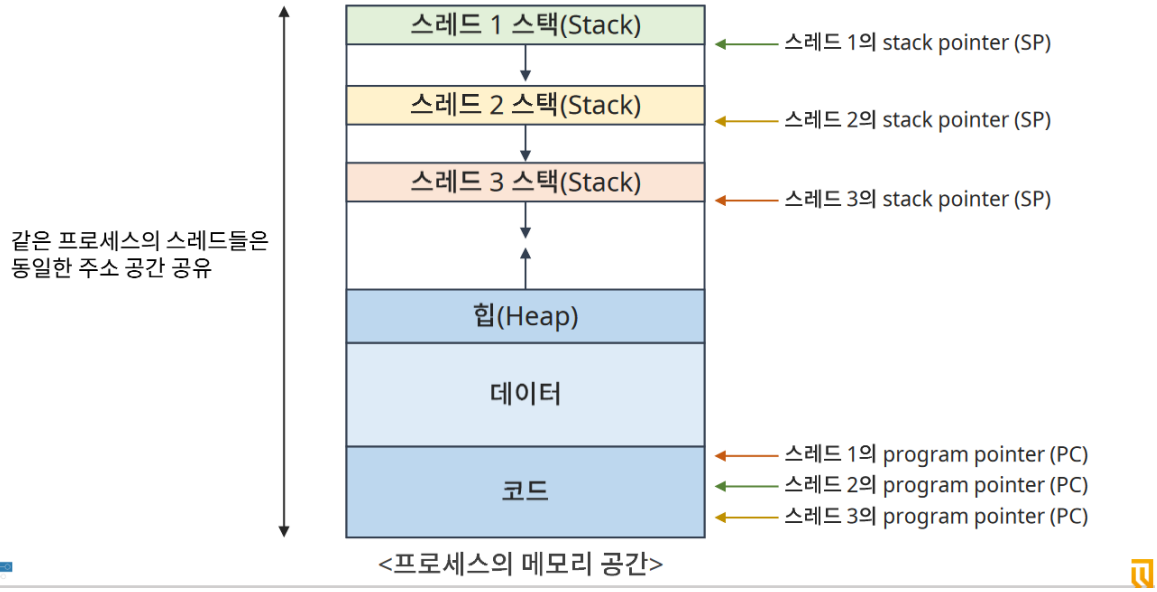
프로세스와 스레드



- 프로세스: 작업을 위한 자원을 할당 받고, 해당 자원을 제어해서 작업 진행
  - 자원: 작업을 위해 필요한 리소스 ⇒ 코드, 데이터, 힙
  - 제어: 자원을 이용해 실제 작업만 실행하는 것 ⇒ 제어 정보, 지역 데이터, 스택
- 스레드: 프로세스의 제어 부분만 분리해둔 것
  - 하나의 프로세스 안에 여러 스레드가 존재할 수 있음



- 프로세스 내부 스레드는 제어 역할만 담당, 리소스 영역은 공유



- 같은 프로세스에 속한 스레드는 동일한 주소 공간 공유
  - 공유 자원: 코드, 데이터, 힙
  - 고유 영역: 스택에 할당되는 스레드 고유의 작업 영역

스레드란

- Light Weight Process (LWP)

    - 프로세스는 자원과 제어 둘 다 갖고 있지만, 스레드는 자원을 공유하고 제어만 갖고 있음
- CPU를 활용하는 기본 단위
    - 스레드가 여러 개면 동시에 CPU 여러 개 사용 가능
- 스레드의 구성 요소
    - Thread ID
    - Register Set
    - Stack
- 제어 요소 외 코드, 데이터 및 자원들은 프로세스 내 다른 스레드와 공유
- 전통적인 프로세스는 단일 스레드 프로세스
    - 멀티 스레드 프로세스도 있음

스레드 장점

1. 사용자 응답성
    - 일부 스레드 처리가 지연돼도 다른 스레드는 작업 계속 처리 가능
2. 자원 공유
    - 자원을 공유해서 효율성 증가 (커널 개입 최소화)
        - ex) 동일 주소 공간에서 스레드 여러 개
    - 2개 프로세스가 같은 자원을 사용하면, 커널이 개입해서 Context Switching 발생, 오버헤드 증가. 하나의 프로세스 내 2개 스레드가 같은 자원을 사용하면 커널 개입 없이 프로세스 내 자원 공유 가능, Context Switching 발생 X ⇒ 효율성 ⬆️
3. 경제성
    - 프로세스 생성, Context Switching 비해 효율적
4. 멀티 프로세스 활용
    - 병렬 처리를 통해 성능 향상
        - 스레드는 프로세서(CPU)를 활용하는 기본 요소. 스레드 여러 개를 통해 동시에 CPU 코어 여러 개를 사용 가능, 성능 향상

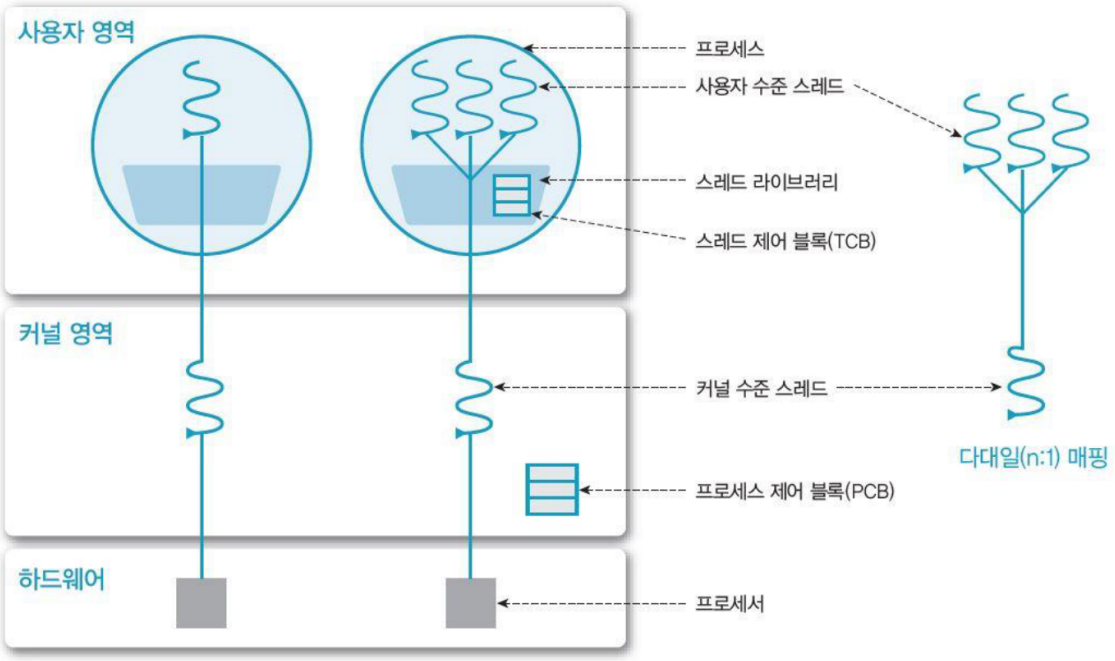
스레드 활용 예시

- 게임할 때, 계속 I/O 요청이 들어옴 (사용자 입력, 스피커/마이크, 화면 출력 등)
- 단일 스레드 경우, I/O 요청이 들어올 때마다 프로세스가 RUN ⇒ BLOCK ⇒ READY ⇒ RUN 상태 변이 과정을 거침. 작업 하나할 때마다 다른 작업을 할 수 없는 상황 발생
- 멀티 스레드로 해결 가능. 3개 스레드가 각각 사용자 입력, 화면 출력, 스피커/마이크를 제어하는 역할을 맡으면 멈춤 없이 게임 가능.
    - 스레드 하나가 지연돼도 다른 스레드 작업 지속 가능

스레드의 구현

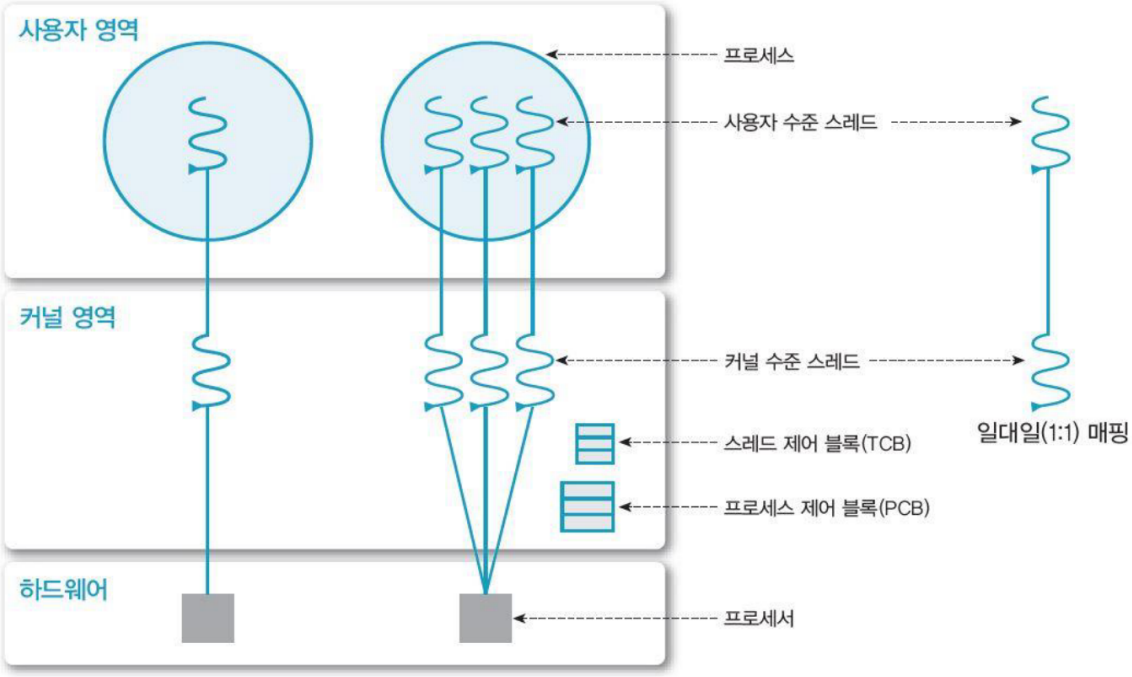
- 사용자 수준 스레드
- 커널 수준 스레드

사용자 수준 스레드



- 사용자 영역의 스레드 라이브러리로 구현
  - 스레드의 생성, 스케줄링 등
  - POSIX threads, Win32 threads, Java thread API…
- 장점
  - 커널은 스레드의 존재를 모름
    - 스레드를 라이브러리 레벨에서 관리해서 커널의 관리를 받지 않음
    - 생성 및 관리의 부하가 적음, 유연한 관리 가능
  - 이식성 (portability) 높음
    - 해당 라이브러리가 있는 시스템이라면 생성해둔 멀티스레드 프로그램 사용 가능
    - ex) JVM 있으면 다른 곳에서도 컴파일 안해도 사용 가능
- 단점
  - 커널은 프로세스 단위로 자원 할당
    - 하나의 스레드가 block 상태가 되면, 모든 스레드가 대기 (single threaded kernel)

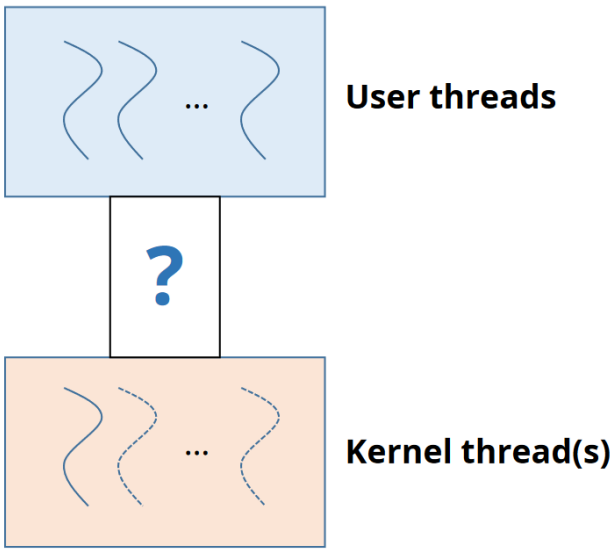
커널 수준 스레드



- OS(Kernel)이 직접 관리
- 장점
  - 커널이 각 스레드를 개별적으로 관리
    - 프로세스 내 스레드들이 병행 수행 가능
- 단점
  - 커널 영역에서 스레드의 생성, 관리 수행
    - Context Switching 등 오버헤드가 큼

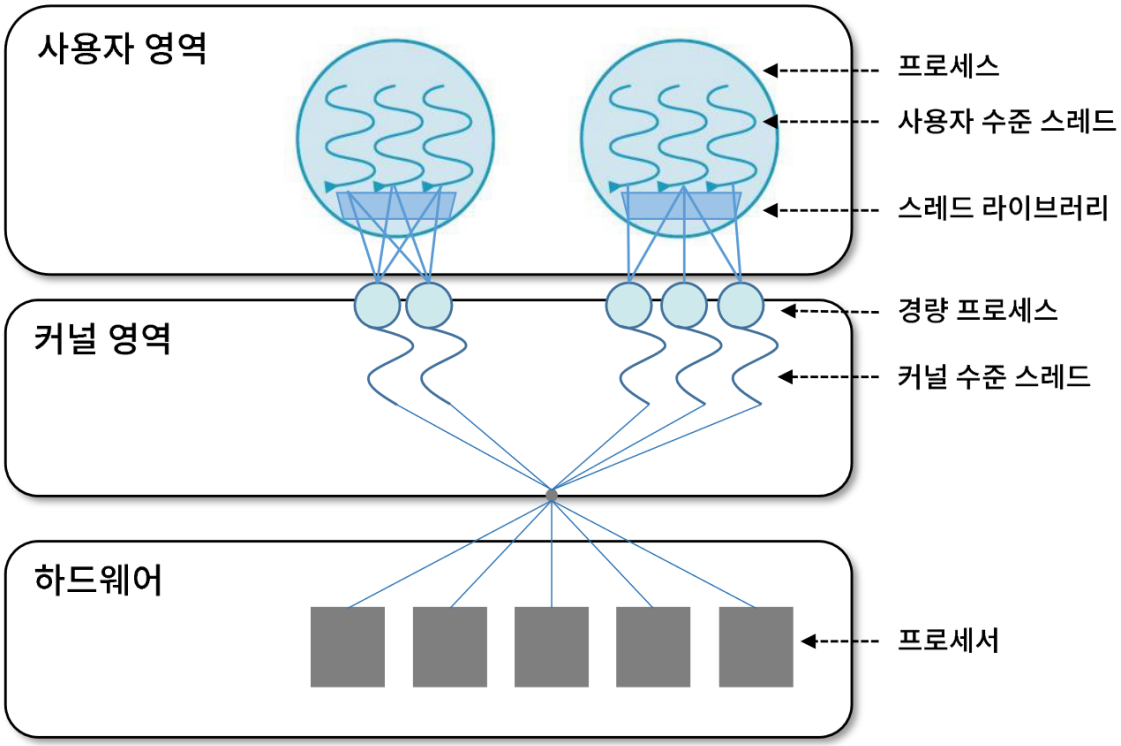
멀티 스레딩 모델

- **다대일(n:1) 모델**
  - 사용자 수준 스레드
- **일대일(1:1) 모델**
  - 커널 수준 스레드
- **다대다(n:m) 모델**
  - n > m
  - 혼합형 스레드



- 다대다 모델: 사용자 수준 스레드와 커널 수준 스레드의 성질을 혼합

혼합형 스레드



- n개 사용자 수준 스레드 - m개의 커널 스레드 (n > m): 사용자
  - 사용자는 원하는 수만큼 스레드 사용
  - 커널 스레드는 자신에게 할당된 하나의 사용자 스레드가 block 상태가 되어도, 다른 스레드 수행 가능 ⇒ 병행 처리 가능

- 효율적이면서도 유연함