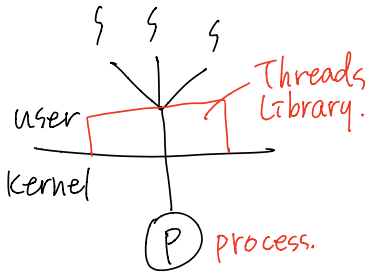
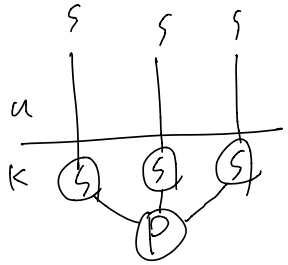
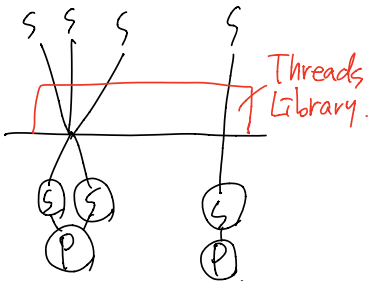


①pure user-level.



②pure kernel-level



③Combined.

* 사용자 레벨 스레드는 말그대로 #include <thread> 또는 import해서 스레드를 이용하는 것을 의미.

① Pure user-level : 커널 스레드 1개 당 유저 스레드 n개 (1:n)
커널은 유저스레드 100개여도 전혀 모른다. → 유저 스레드 중 하나라도 I/O 발생 시 해당 process는 I/O 끝날 때까지 block된다.

② Pure kernel-level : (1:1 방식) n개의 커널 스레드 → n개의 유저 스레드 담당.
1:1 이기에 병렬성은 좋으나 유저모드 ↔ 커널모드 전환이 빈번히 일어나서 효율↓일 수 있다.

③ Combined : 두 장점을 혼합한 방식.

1. 커널 레벨 스레드
- 커널 스레드는 가장 가벼운 커널 스케줄링 단위.
- 하나의 프로세스는 최소 1개의 커널 스레드 가짐.
- 커널 영역에서 스레드 연산.
- 커널이 스레드를 관리하므로 커널에 종속적. (스케줄링 주체가 커널).

장점)
- 프로세스의 스레드들을 여러 프로세서에 한번에 디스패치 가능 (멀티 프로세서에서 매우 빠름)
- 한 스레드의 입출력 작업 끝나기 전에 다른 스레드를 통해 다른작업이 가능.

- 커널이 각 스레드를 개별적으로 관리할 수 있다. (안정적이라는 느낌).
- 커널이 직접 스레드를 제공하므로 안정성 + 다양한 기능 제공.

단점)

- 스케줄링, 동기화를 위해 커널을 호출 → 무겁고 오래걸림 (save & reload필요)
- 사용자 모드 ↔ 커널모드 전환이 빈번히 이루어져 성능저하.
- 구현이 까다롭고 자원 소모 ↑ 경향.

2. 사용자 레벨 스레드

- 사용자 영역에서 스레드 연산.
- 커널에 의존적이지 않은 라이브러리 형태로 활용된다.

장점)

- OS에서 스레드 지원할 필요없다. (라이브러리로 생성 가능하므로).
- 스케줄링, 동기화에 커널 호출X
  → 인터럽트 발생해도 커널레벨 스레드보다 오버헤드↓
  (OS스케줄러에 의한 문맥 교환이 없다. 유저 레벨 스레드 스케줄러 이용).
- 커널은 유저레벨 스레드의 존재조차 모르기 때문에 모드간의 전환이 없고 성능↑.

단점)

- 스케줄링 우선순위 지원 X → 어떤 스레드가 먼저 실행될지 모름.
- 프로세스에 속한 스레드 중 I/O 작업에 의해 하나라도 block되면 전체 스레드가 블록된다.

3. 유저-커널 스레드 사상 방법.

① 다대일 모델 (Many-to-One)

장점) • 사용자는 필요한 만큼의 유저 스레드 생성 가능.

- 스레드 관리가 유저 공간의 라이브러리에 의해 이루어진다. (모드 전환 X 효율적).

단점) • 한 번에 하나의 스레드만 커널에 접근. 실제 병렬 실행이 아님.

• 한 스레드 block → 프로세스의 전체 스레드 block.

② 일대일 모델 (One-to-One) : 가장 이상적인 방법.

장점) • 병렬성 제공 (하나 스레드 block되어도 다른 스레드로 작업 가능)

단점) • 유저 스레드 생성 시 → 커널 스레드도 생성 (오버헤드↑)
(모드 전환도 ↑ → 성능↓일수 있으나 현재 H/W성능↑라서 큰 무리가능).
병렬성을 위해

③ 다대다 모델 (Many-to-Many) : 유저 > 커널

⇒ 필요한 만큼 유저 스레드 생성하지만, 커널 스레드 수 만큼 스케줄되기 때문에 진정한 의미의 병렬성이라고 보기 어렵다.
우선순위 지정되지 않아 유저스레드 간의 순차적 커널 스레드 점유.