스레드 개념

5th Week

Kim, Eui-Jik

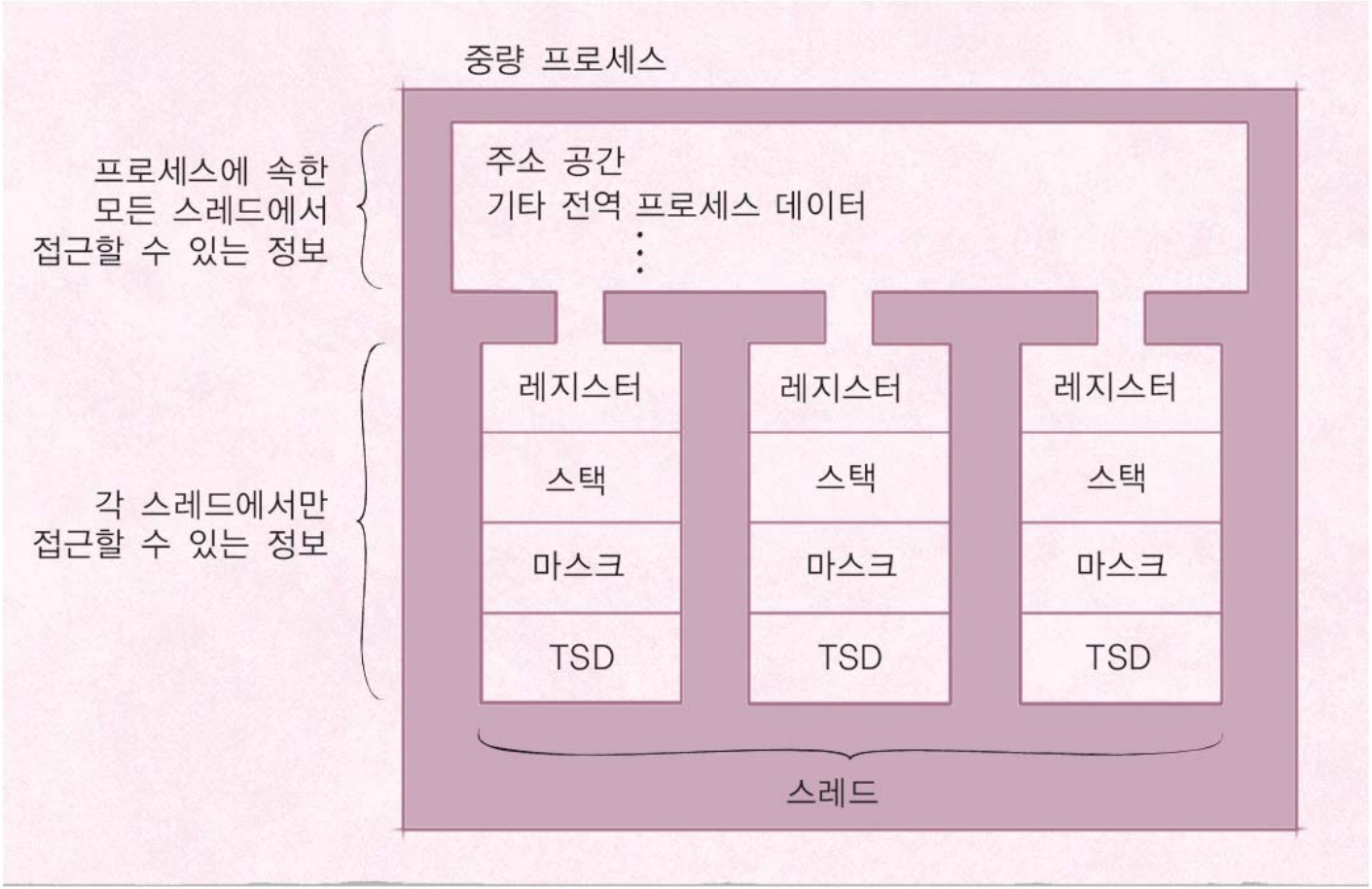
Contents

- 스레드 정의
- 스레드 상태 : 스레드 생명 주기
- 스레딩 모델

스레드 정의

- 스레드
  - 경량 프로세스(LWP: Light Weight Process)
  - 명령어들의 한 줄기 혹은 프로그램 제어 흐름
    - 프로세서를 사용하는 기본 단위이며, 명령어를 독립적으로 실행할 수 있는 하나의 제어 흐름
  - 프로세스에 있는 스레드들은 병행으로 실행되면서 공통의 목표를 이루려고 협력
  - 레지스터, 스택, 신호 마스크처럼 스레드마다 있는 특정 데이터들은 각 스레드에 국한
  - 프로세스에 속한 주소 공간은 해당 프로세스에 속한 모든 스레드에 공통
- 특정 플랫폼에서 스레드를 구현한 방식에 따라 스레드를 운영체제 혹은 스레드를 생성한 사용자 프로그램에서 관리

스레드 정의



[그림 4-1] 스레드와 프로세스의 관계

(TSD: Thread Specific Data)

스레드의 장점

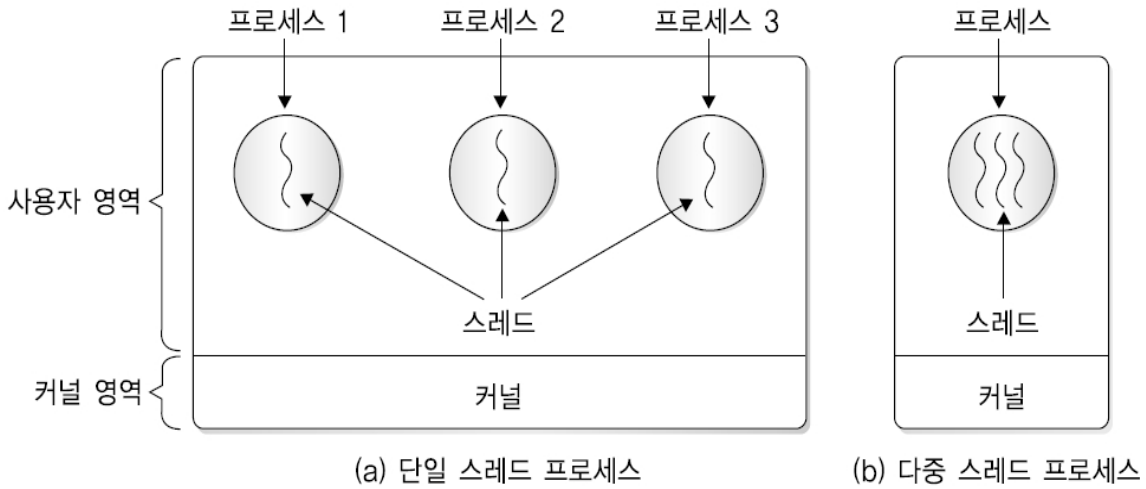
- 스레드의 장점
  - 프로세스에 포함된 스레드들은 공통의 목적 달성을 위해 병렬로 수행
    - 자원을 공유하여 한 프로세스에서 동시 작업 가능
  - 시스템 성능과 효율 향상
    - 하나의 프로세스가 서로 다른 프로세서에서 프로그램의 다른 부분을 동시에 실행 가능
    - 응용 프로그램 하나가 비슷한 작업들을 여러 개 수행
  - 스레드를 이용하여 다음과 같은 이점을 얻을 수 있음
    - 사용자에 대한 응답성 증가
      - 응용 프로그램의 일부분이 봉쇄 또는 긴 작업 수행 시에도 프로그램 실행을 계속 허용하여 사용자에 대한 응답성이 증가
    - 프로세스의 자원과 메모리 공유 가능
      - 스레드는 그들이 속한 프로세스의 자원과 메모리를 공유하므로, 응용 프로그램 하나가 같은 주소 공간에서 여러 개의 스레드를 실행, 시스템 성능 향상과 편리함 제공
    - 경제성
      - 한 프로세스의 자원을 공유하므로 프로세스를 생성하는 것보다 오버헤드를 줄일 수 있음
    - 다중 프로세서 구조 활용 가능
      - 다중 프로세서 구조에서 각 스레드는 다른 프로세서에서 병렬로 실행될 수 있음

[참고] Process vs. Thread

- Thread
  - 프로세스에 소속됨
  - CPU 스케줄링의 기본단위
  - Dynamic
  - LWP (light weight process)
  - 경량 프로세스의 문맥교환 (CPU switching, thread context switch)
    - 레지스터 세트 교환만
- Process
  - 다중 스레드를 갖을 수 있음
  - 스레드의 콘테이너
  - Static
  - HWP (heavy weight process)
  - 중량 프로세스의 문맥교환 (process switching, context switching)
    - 레지스터 세트 교환과 메모리 관련 작업도
- (예1) web browser
  - 구성요소: Image와 text 를 display하는 thread
  - Network에서 데이터를 가져오는 thread
- (예2) word processor
  - Graphics를 display하는 thread
  - Keystroke를 읽어오는 thread
  - Spelling과 grammar를 검사하는 thread

[참고1] 단일 스레드와 다중 스레드 모델

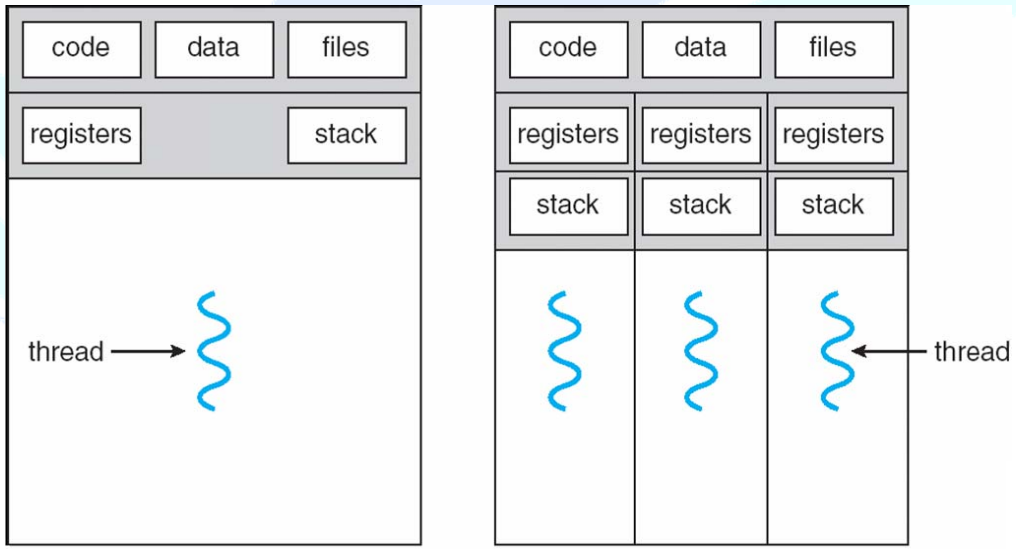
- 운영체제는 단일 프로세스에서 단일 스레드 실행과 다중 스레드 실행을 지원
  - 단일 스레드 프로세스
    - 프로세스 하나에 스레드 하나가 실행되는 전통적인 방식으로 스레드의 개념이 불확실함
    - MS-DOS
  - 다중 스레드 프로세스
    - 프로세스 하나에 여러 스레드를 실행하는 것을 지원
    - 윈도우 NT/XP, 솔라리스 등



(a) 단일 스레드 프로세스 (b) 다중 스레드 프로세스

[참고2] 단일 스레드와 다중 스레드 모델

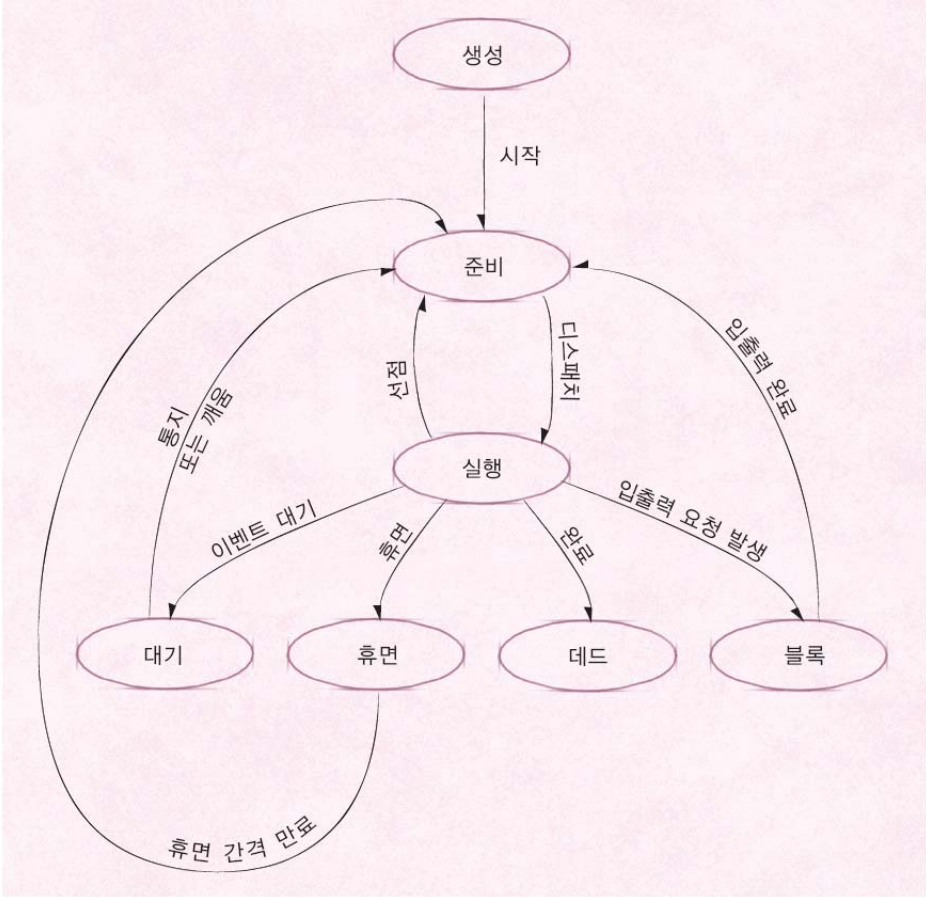
- 단일 스레드 프로세스 모델
  - 프로세스를 하나의 스레드, 스레드가 가진 레지스터와 스택으로 표현
- 다중 스레드 프로세스 모델
  - 프로세스를 각각의 스레드와 고유의 레지스터, 스택으로 표현, 프로세스 주소 영역을 모든 스레드가 공유
  - 프로세스의 모든 스레드는 해당 프로세스의 자원과 상태를 공유, 같은 주소 공간에 존재하며 동일한 데이터에 접근



스레드 상태 : 스레드 생명 주기

- 스레드 상태
  - 스레드들은 프로세서를 함께 사용하고 항상 스레드 하나만 실행된다
  - 탄생 상태(생성 상태)
  - 준비 상태(실행가능 상태)
    - 스레드가 프로세서에 의해 실행될 수 있는 상태
    - 스레드를 대기시킨 이벤트 발생시 해당 스레드는 준비 리스트에 삽입됨
  - 실행 상태
    - 스레드가 프로세서를 점유하여 실행 중인 활성화 상태
  - 데드 상태(종료 상태)
    - 스레드가 작업을 완료하거나 다른 스레드에 의해 종료될 때
    - 스레드가 작업을 종료하면 자원을 해제, 레지스터 문맥과 스택 할당 제거
  - 블록 상태
    - 입출력 요청이 완료되기를 대기해야 할 때
  - 대기 상태
    - 특정 이벤트 발생(ex. 마우스 움직임, 다른 스레드가 보낸 신호 등)을 대기해야 할 때
  - 휴면 상태
    - 일정 시간 동안 휴면 상태일지를 휴면 간격이라 명함
    - ex. 워드프로세서의 문서 백업
    - 휴면시간이 만료되면 준비상태가 됨

스레드 상태 : 스레드 생명 주기



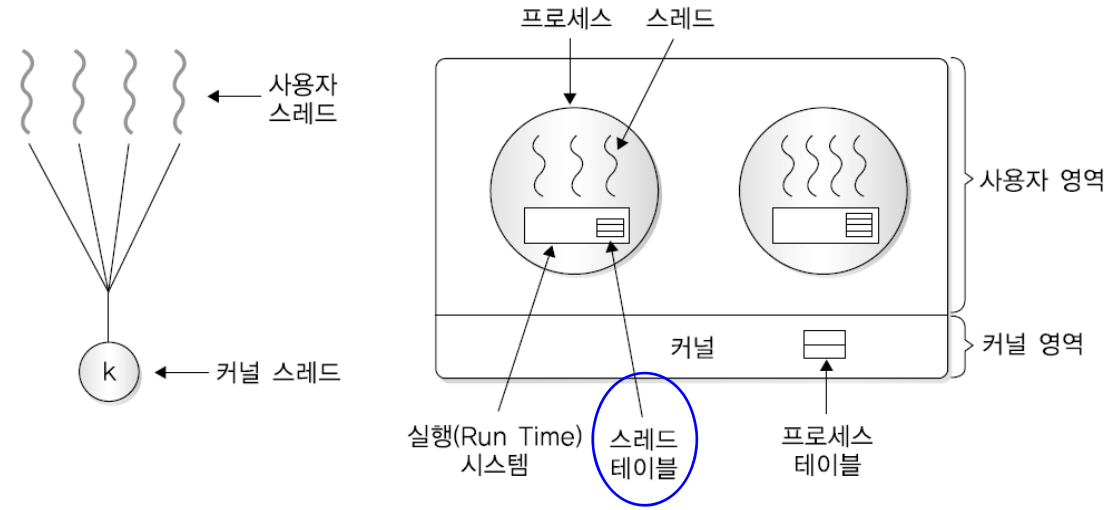
[그림 4-2] 스레드 생명 주기

스레딩 모델

- 세가지의 대중적인 스레딩 모델
  - 사용자 수준 스레드
  - 커널 수준 스레드
  - 사용자 수준과 커널 수준 스레드의 조합

[참고] 사용자 수준 스레드

- 커널 스레드를 지원하지 않는 운영체제에서 사용
- 사용자 수준 스레드 여러 개가 커널 스레드(프로세스) 하나로 매핑되는 방식
- 다중 스레드 프로세스에 대해 프로세서(실행 문맥) 하나를 할당하므로 다대일 스레드 매핑(Many-to-One)이라 칭함
- 사용자 수준(공간)에서 스레드 관리가 효율적으로 이루어지므로 스레드와 관련된 모든 과정을 응용 프로그램이 수행
- 응용 프로그램은 사용자 수준 스레드 관리를 위한 루틴으로 구성된 스레드 라이브러리(스레드 패키지)를 이용, 다중 스레드로 프로그래밍 가능



(a) 다대일 스레드 매핑

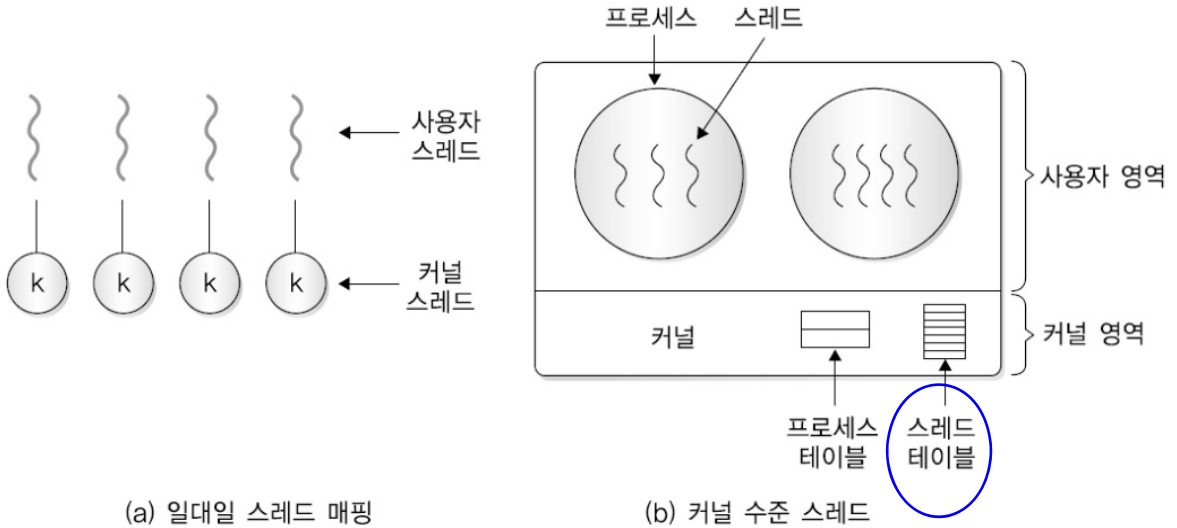
(b) 사용자 수준 스레드

[참고] 사용자 수준 스레드

- 사용자 수준 스레드 이용 시 시스템의 장점
  - 높은 이식성
    - 기본 커널 변경 없이 모든 운영체제에 적용 가능
  - 오버헤드 감소
    - 스레드 관리를 위한 모든 데이터 구조가 프로세스의 사용자 주소 공간에 있어 커널의 도움 없이 스레드 교환 가능
  - 스케줄링의 유연성
    - 스레드 라이브러리에서 스레드 스케줄링을 제어하므로 스케줄링이 응용 프로그램에 맞게 적절히 구성됨
- 사용자 수준 스레드 이용 시 시스템의 단점
  - 시스템의 동시성 지원 불가
    - 한 번에 하나의 스레드만 커널에 접근 가능하므로 여러 스레드가 시스템 호출을 동시에 사용할 수 없음
  - 시스템 규모 확장 제약
    - 커널이 프로세서 내부의 다중 스레드를 프로세스 하나로 관리, 다중 처리 환경에서 여러 프로세스를 이용한 분산 처리를 할 수 없음
    - 한 Thread가 blocking system call하면 전체 프로세스 block

[참고] 커널 수준 스레드

- 사용자 수준 스레드의 한계를 해결하기 위해 사용자 스레드마다 프로세서(실행 문맥)를 매핑하는 일대일 스레드 매핑을 지원
- 커널에 의해 생성, 삭제되고 커널의 텍스트와 전역 데이터를 공유, 자신만의 커널 스택을 가짐
- 사용자 영역에는 스레드 관리를 위한 코드가 없음
  - 모든 응용 프로그램은 다중 스레드로 지원, 응용 프로그램의 스레드는 하나의 프로세스에서 지원
  - 스레드 관련 모든 작업은 커널(운영체제)이 지원, 프로세스와 스레드에 대한 실행 문맥 정보 유지
- 일대일 모델(One-to-One) 구현 운영체제 : 윈도우 NT/XP/2000, 리눅스, 솔라리스 9 이상 버전, OS/2



[참고] 커널 수준 스레드

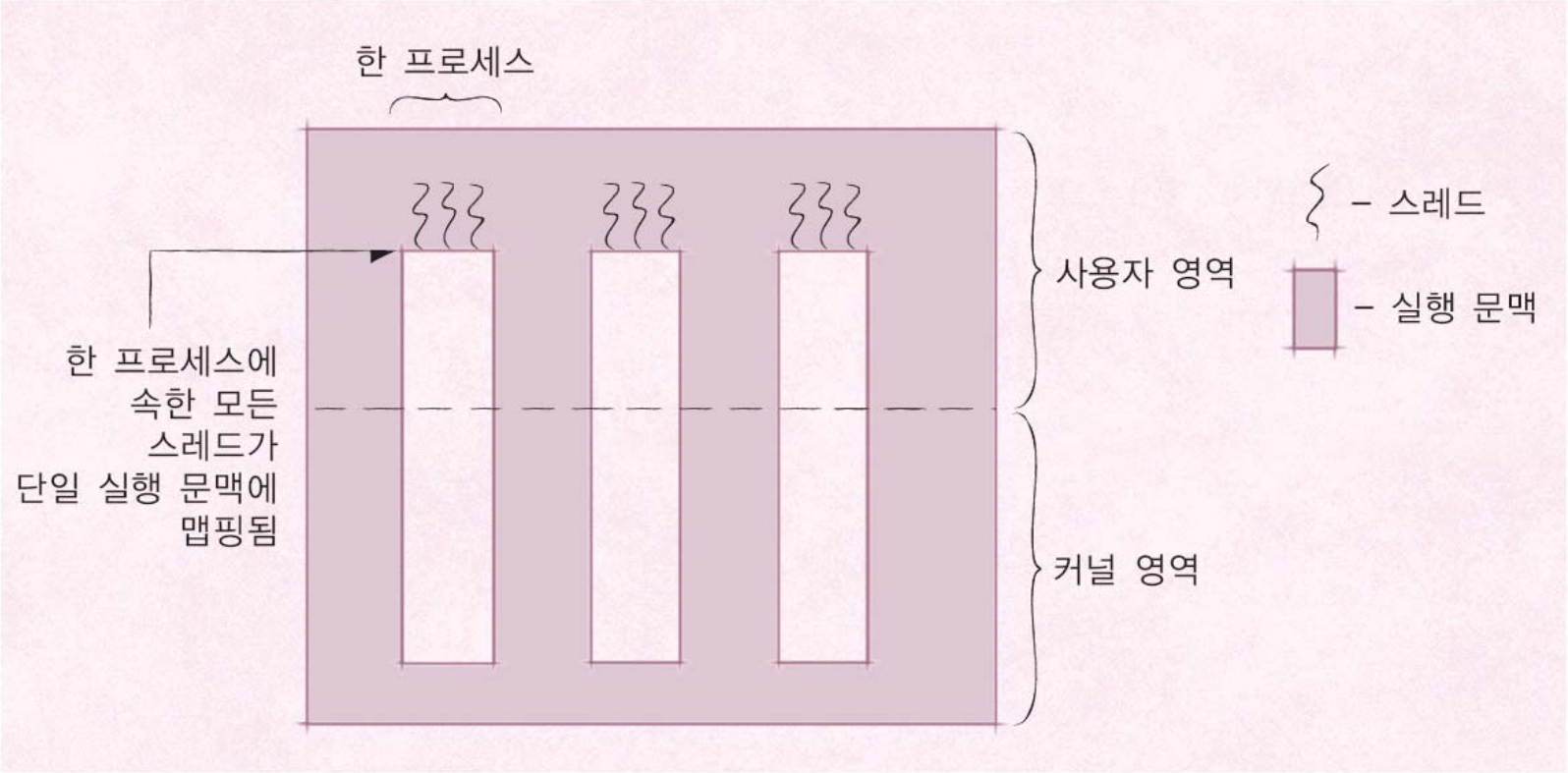
- 커널 지원 부족 문제 해결
  - 커널에 의한 직접적인 스케줄링과 실행으로 사용자 수준 스레드의 문제를 해결하여 시스템 성능 향상
- 병렬 실행 가능
  - 하나가 시스템 호출 시 다른 스레드가 중단되는 다대일 방식의 문제를 해결할 수 있어 다중 프로세서에서 다중 스레드를 병렬로 실행 가능
  - 시스템 규모 확장이 쉬우며 처리량을 늘릴 수 있음
- 커널 모드 전환 오버헤드
  - 커널 스레드 생성으로 인한 오버헤드 증가로 응용 프로그램의 성능 저하를 막기 위해 시스템이 지원 스레드 수를 제한해야 함
  - 사용자 수준 스레드보다 스케줄링과 동기화를 위해 자원이 더 필요함
  - 시스템이 모든 스레드를 관리하므로 오버헤드 증가
- 이식성이 떨어짐
  - 시스템 변경 시 제공된 스레드 API(Application Program Interface)를 사용하여 프로그램을 수정
  - 제한적인 자원으로 인해 사용자 수준 스레드 생성에 따라 커널 스레드를 무한정 생성할 수 없음

스레딩 모델

- 사용자 수준 스레드는 사용자 영역에서 스레드 연산 수행
  - 특권 명령을 실행할 수 없거나 커널 프리미티브에 직접 접근할 수 없는 런타임 라이브러리가 스레드를 생성

- 사용자 수준 스레드 운용
  - 다대일 스레드 맵핑
    - 운영체제가 멀티스레드 프로세스 하나에 있는 모든 스레드에 실행 문맥 하나를 맵핑
    - 사용자 수준 라이브러리로 프로세스의 스레드를 스케줄링하고 디스패치 함

스레딩 모델

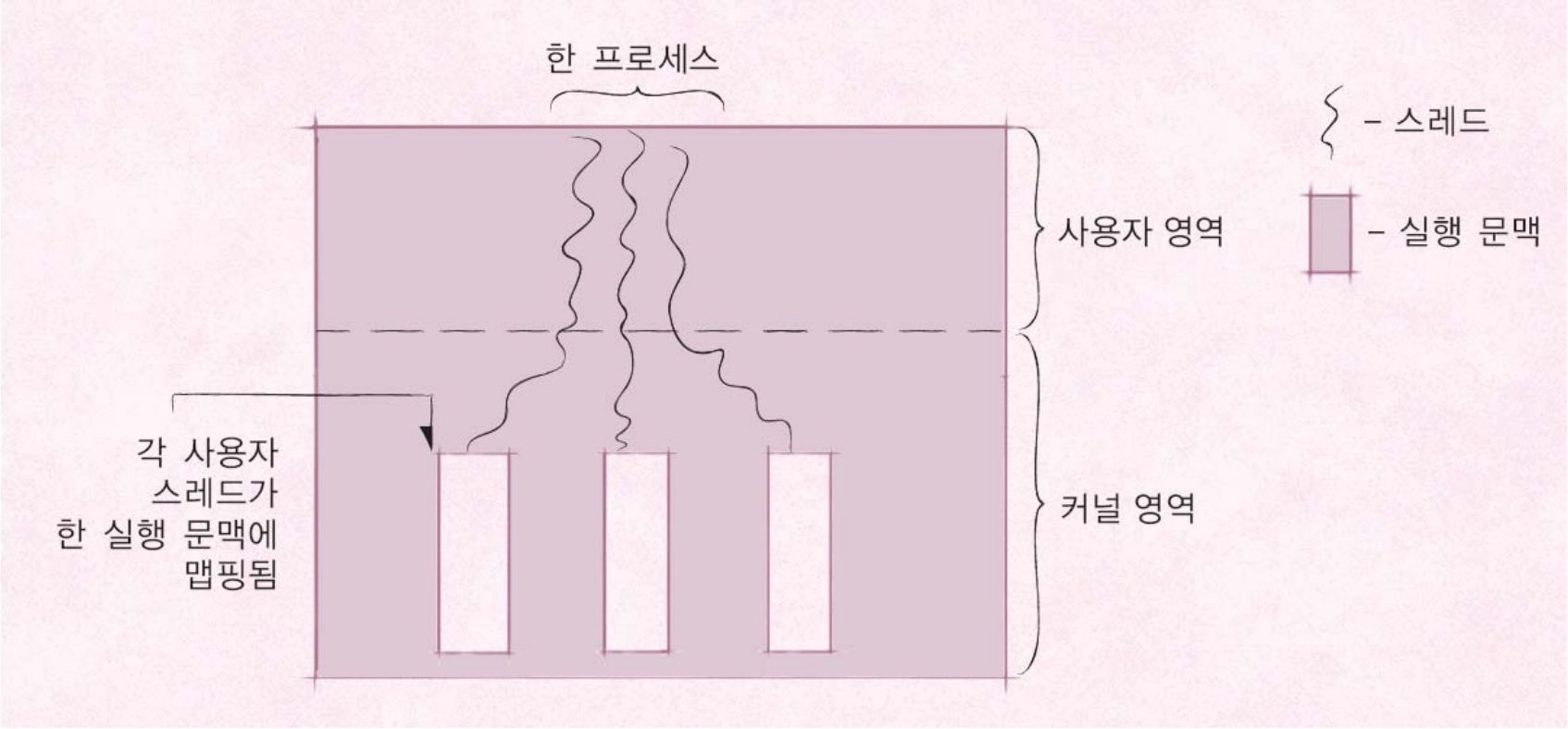


[그림 4-3] 사용자 수준 스레드

스레딩 모델

- 커널 수준 스레드
  - 각 스레드 마다 고유한 실행 문맥을 맵핑하는 방법으로 사용자 수준 스레드의 한계 해결
    - 일대일 스레드 맵핑 제공
      - 장점 : 상호작용성 증가
      - 단점 : 실행될 시스템이 달라지면 해당 운영체제에서 제공하는 스레드 API를 사용해 프로그램을 수정해야 함

스레딩 모델

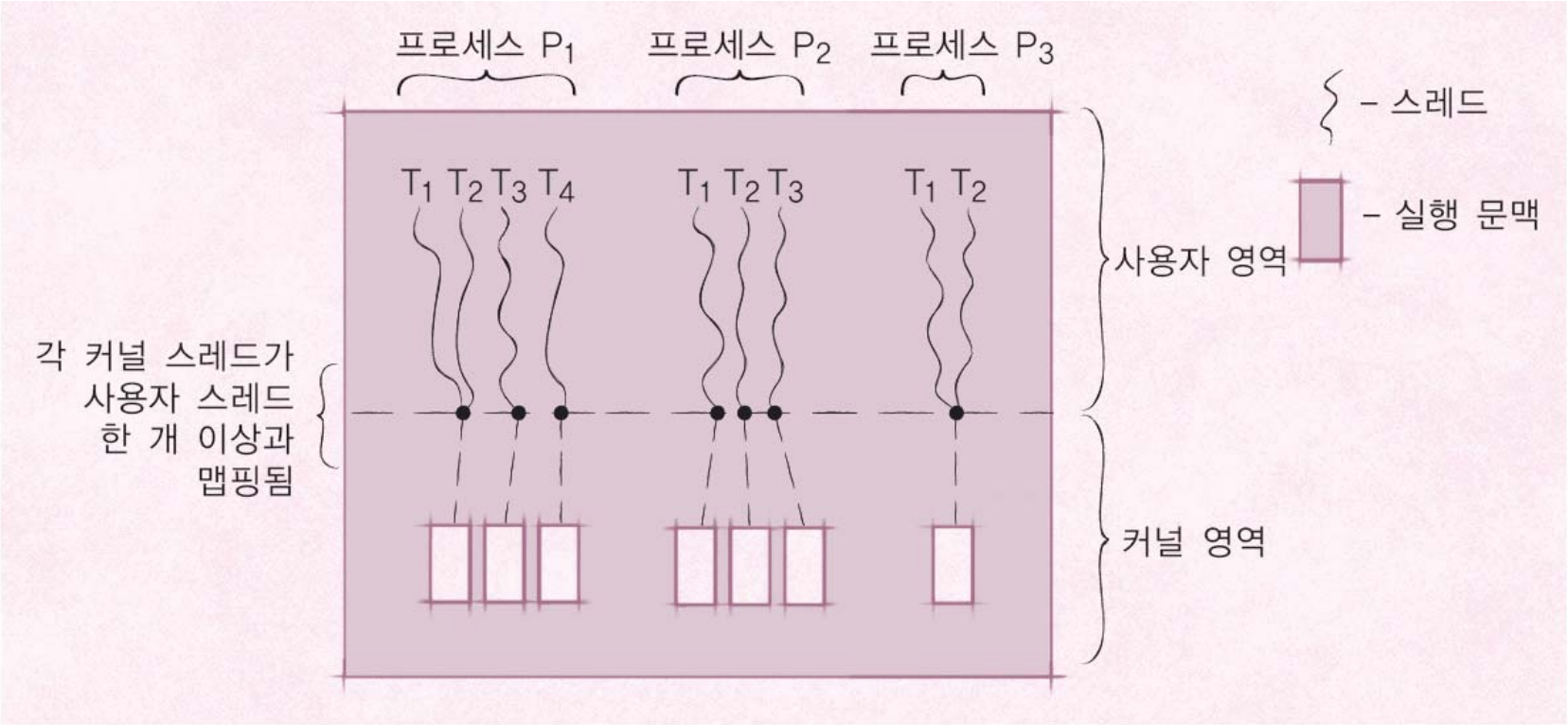


[그림 4-4] 커널 수준 스레드

스레딩 모델

- 사용자 수준 스레드와 커널 수준 스레드의 조합
  - 사용자와 커널 수준 스레드 조합의 수행
    - 다대다 스레드 맵핑 (m-to-n)
      - 많은 사용자 수준 스레드를 한 그룹의 커널 스레드에 맵핑
      - 스레드 풀링을 통해 오버헤드 문제를 해결
        - 시스템이 관리하는 스레드의 풀(pool)을 응용프로그램에 제공, 스레드를 효율적으로 사용
        - 미리 생성한 스레드를 재사용, 스레드를 생성하는데 소비되는 시간을 줄여 시스템의 부담을 덜어줌
  - 작업자 스레드
    - 풀에서 지속되는 커널 스레드는 할당된 스레드에 따라 다른 기능을 수행

스레딩 모델



[그림 4-5] 하이브리드 스레딩 모델