4장 쓰레드
(THREADS)

강의 순서

- Part 1 : 배경 -

1주 : 01장 - 컴퓨터 시스템의 개요

2주 : 02장 - 운영체제 개요

- Part 2 : 프로세스 -

3주 : 03장 - 프로세스 기술과 제어

- Part 4 : 스케줄링 -

4주 : 09장 - 단일처리기 스케줄링 (1/2)

5주 : 09장 - 단일처리기 스케줄링 (2/2)

- Part 2 : 프로세스 -

6주 : 04장 - 쓰레드

7주 : 05장 - 병행성 - 상호배제와 동기

8주 : 중간고사

- Part 2 : 프로세스 -

9주 : 06장 - 병행성 – 교착상태와 기아

- Part 3 : 메모리 -

10주 : 07장 - 메모리 관리

11주 : 08장 - 가상메모리 (1/2)

12주 : 08장 - 가상메모리 (2/2)

- Part 5 : 입출력과 파일 -

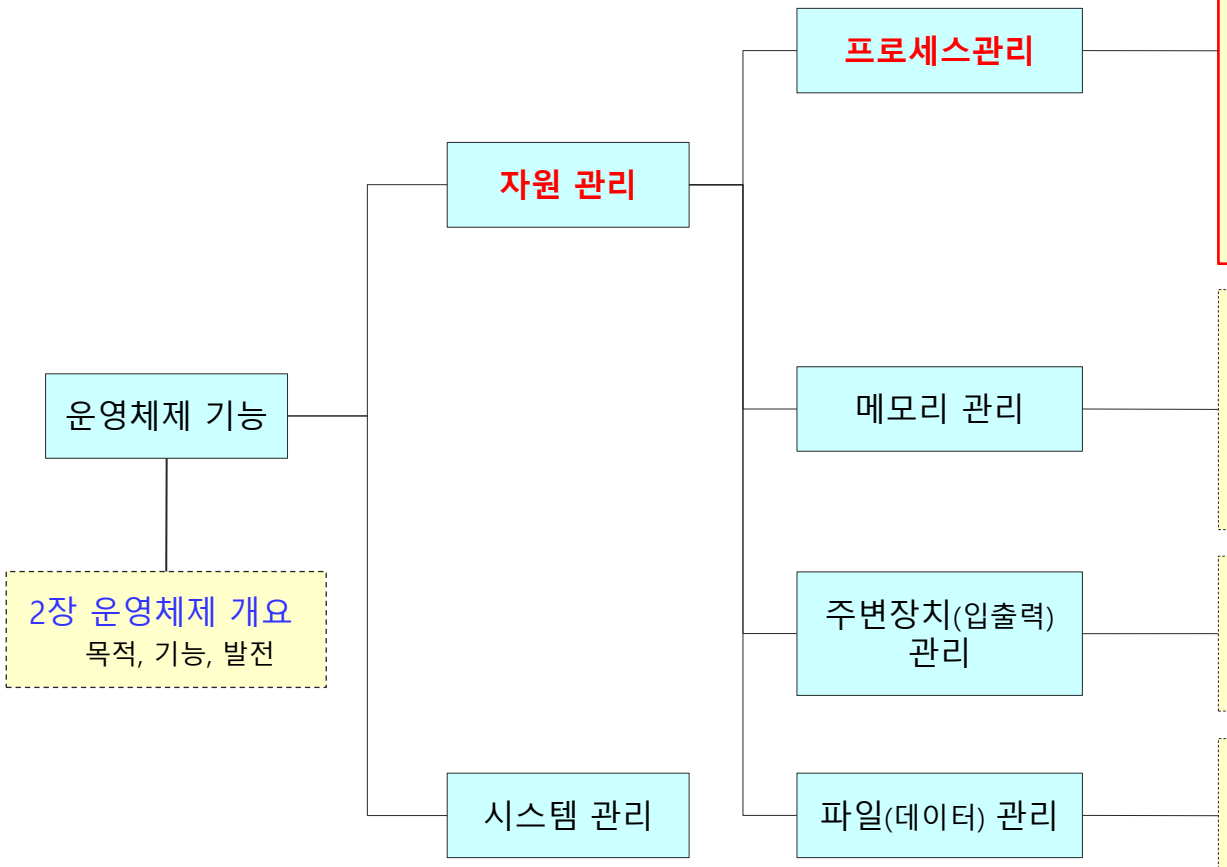
13주 : 11장 - 입출력 관리와 디스크 스케줄링

14주 : 12장 - 파일 관리

15주 : 기말고사

배울 내용

- **운영체제의 내부 기능**



3장 프로세스 : 상태, 기술, 제어
9장 단일처리기 스케줄링
단기, 중기, 장기 스케줄링
4장 쓰레드 : 멀티쓰레드, 구현 종류
5장 병행성 : 상호배제와 동기화
병행성, 상호배제, 세마포어, 모니터, 메시지, 생산자/소비자, 판독자/기록자
6장 병행성 : 교착상태와 기아
예방, 회피, 발견, 식사하는 철학자

7장 메모리관리
메모리분할, 페이징, 세그먼테이션
8장 가상메모리
반입, 배치, 교체, 적재집합관리, 클리닝, 부하제어

11장 입출력관리와 디스크 스케줄링
버퍼링, 디스크 스케줄링, RAID, 디스크 캐시

12장 파일관리
파일, 파일시스템, 디렉토리, 파일공유, 블록킹, 파일할당, 가용공관관리, 볼륨

대상 운영체제 : UNIX(SVR4, Solaris, BSD), Windows, Linux, Mac OS X, iOS, Android

목 차

4.1 프로세스와 쓰레드(thread)

• 기존 프로세스 단위의 관리에서의 문제점
① 프로세스 코드의 순차적인 실행으로 인한 오버헤드
② 시스템 호출이 포함된 프로세스의 지나친 대기시간
③ 멀티처리기(코어)를 이용해도 하나의 프로세스는 하나의 처리기(코어)에서만 동작
④ 동일한 프로그램의 복수 호출시에 동일 프로세스 이미지 복수개가 주기억장치에 생성

- **기존 프로세스 단위 관리에서의 문제점**

① **프로세스 코드의 순차적인 실행으로 인한 오버헤드**

- ❖ 세부 기능에 상관없이 프로세스 내부의 프로그램 궤적에 의해 순차적으로만 실행
  - √ 프로세스가 진행되다가 프로세스 내의 다른(입력, 출력 등) 모듈들을 수시로 확인해야 함.
- ❖ 프로세스를 쪼개어(키보드, 화면, 네트워크, 백업, 내부연산 등) 별도로 실행시키면 어떨까?

② **시스템 호출이 포함된 프로세스의 지나친 대기시간**

- ❖ 시간할당량이 남아도 블록 큐로 가서 대기하다 사건 발생시에 다시 준비 큐로 진입
- ❖ 프로세스 코드를 여러 개로 분할해 각각을 별도의 단위로 관리(수행, 블록, 준비...)하면 어떨까?

③ **멀티처리기(코어)를 이용해도 하나의 프로세스는 하나의 처리기(코어)에서만 동작**

- ❖ 중요하거나 큰 프로세스를 분할해 여러 처리기(코어)에서 나누어 동시(병렬) 수행하면 어떨까?

④ **동일한 프로그램의 복수 호출시에 동일 프로세스 이미지 복수개가 주기억장치에 생성**

- ❖ 하나만의 프로세스 이미지를 이용(공유)해서 복수개의 실행을 하면 어떨까?

이러한 다양한 문제점들을 해결하는 방법으로 탄생한 것이 **쓰레드(Thread)의 개념**
1) 프로세스의 코드를 나눌 수 있다면 나누어 별도의 실행(스케줄링) 단위로 적용 → ①, ②, ③
2) 하나의 프로세스 이미지, 특히 코드를 재활용(공유) 하자 → ④

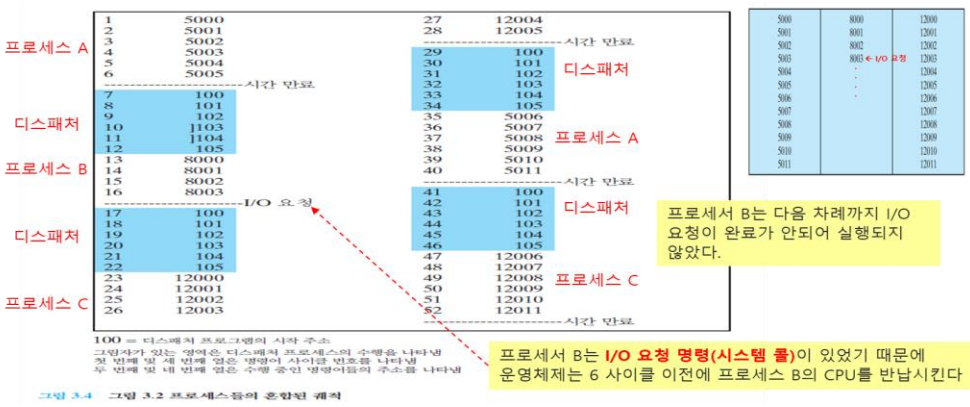
- **프로세스의 2대 특성** – 쓰레드 개념의 설명을 위한 배경

1) 자원 소유권 (resource ownership)

- ❖ 자신의 프로세스 이미지(PCB, 코드, 데이터, 스택, 속성)를 위한 주소 공간
- ❖ 경우에 따라 주기억장치, 입출력 장치, 파일 등의 자원에 대한 소유권 할당
- ❖ 운영체제는 보호기능을 수행하여 프로세스 간의 불필요한 간섭을 방지

2) 수행/스케줄링(execution/scheduling) 개체

- ❖ 프로세스의 수행은 하나 이상의 프로그램을 통과하는 수행경로(궤적)를 따른다
- ❖ 한 프로세스는 다른 프로세스들과 번갈아 가면서 수행(interleave)
- ❖ 따라서 프로세스는 수행상태(수행, 준비 등)와 디스패칭 우선순위를 가지며, 운영체제에 의해 스케줄 되고 **디스패치**되는 개체이다.



실행되는 프로세스가 변경될 때 각종 실행정보가 문맥교환 절차를 거쳐 보관되고 다시 재실행시에는 복원되어 실행된다.

- **현대 OS에서 프로세스를 '태스크' 및 '쓰레드'라는 특성으로 분리**

• 프로세스의 주요 특성
1) 자원 소유권
2) 수행/스케줄링 개체

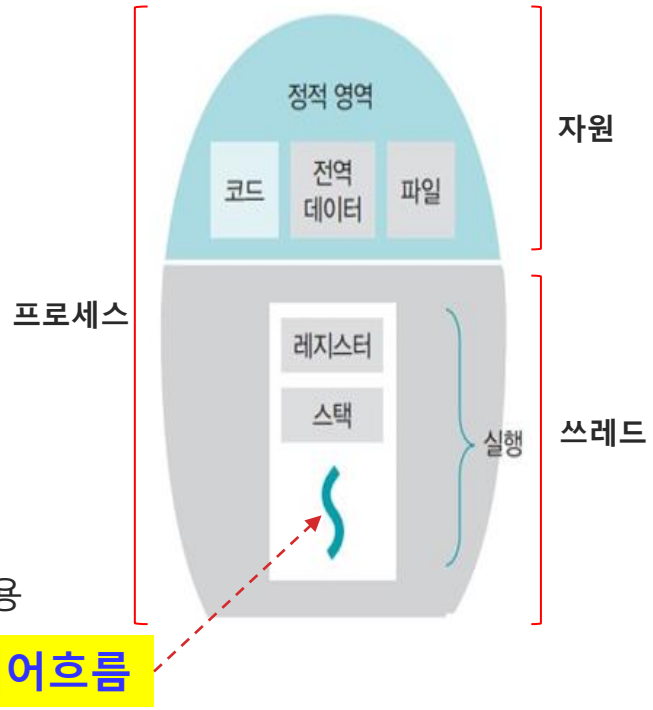
1) 태스크(Task)

- ❖ 일반적으로 **프로세스**라고도 부른다
- ❖ Resource Container (사용자 문맥, 시스템 문맥)
  - → 자원소유권 단위
- ❖ 프로세스 단위로 자원이 할당
  - → 코드, 데이터, 스택 등

쓰레드(Thread) :
- 실
- 가닥(맥락)
- 실같이 가느다란 줄기

2) 쓰레드(Thread)

- ❖ 경량 프로세스 (LWP : lightweight process)
- ❖ 수행/스케줄링 개체
- ❖ 프로세스에 속한 **제어흐름**
  - → 코드의 실행 정보, 레지스터 문맥 등
  - → 코드는 프로세스(태스크)에 있는 것을 (공용으로) 활용



- **단일 쓰레딩 대 멀티쓰레딩**
  - **단일쓰레딩**
    - ❖ 하나의 프로세스에 하나의 제어흐름만 있는 구조
  - **멀티쓰레딩** (Multithreading)
    - ❖ 하나의 프로세스 내에서 여러 개의 제어흐름(쓰레드)의 실행을 지원하는 기능
  - 사례
    - ❖ MS-DOS
      - √ 단일 사용자 프로세스와 단일 쓰레드를 지원한다
    - ❖ 초기 UNIX 계열
      - √ 다중 사용자 프로세스를 지원하지만, 프로세스 당 하나의 쓰레드를 지원한다.
    - ❖ Java 수행시간환경(run-time environment)
      - √ 하나의 프로세스가 멀티 쓰레드를 지원한다.
    - ❖ 최신 버전의 UNIX, Windows, Solaris
      - √ 멀티쓰레드를 지원하는 멀티 프로세스를 사용한다

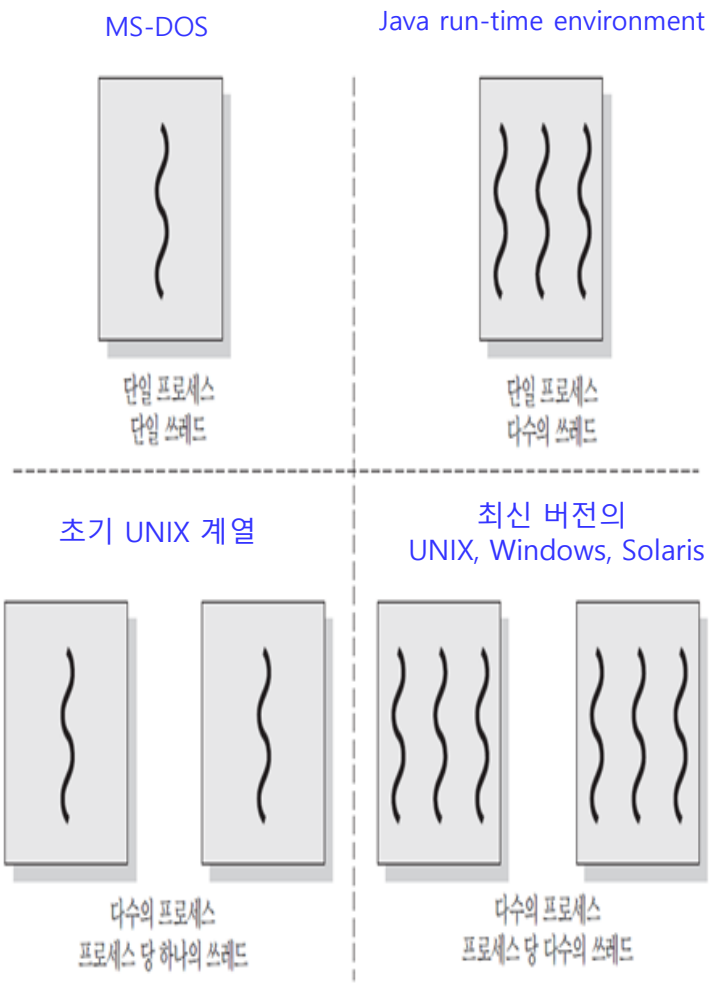


그림 4.1 쓰레드 및 프로세스

- **멀티쓰레드 환경에서의 프로세스의 정의**

  - 현대 OS에서 프로세스를 **태스크** 및 **쓰레드** 라는 두가지 특성으로 분리
    - ❖ **태스크(Task)** 또는 **프로세스**:
      - √ Resource Container (사용자 문맥, 시스템 문맥) ← **자원소유권 단위**
    - ❖ **쓰레드(Thread)** 또는 **경량 프로세스** (lightweight process) :
      - √ 제어 흐름 (실행 정보, 레지스터 문맥) ← **디스패칭 단위**

1) 태스크 (또는 **프로세스**) 관련 사항

❖ 프로세스는 보호의 단위와 자원할당의 단위로 정의

① 프로세스 이미지(코드를 포함)를 유지하는 가상 주소 공간

② 처리기, (IPC를 위한) 다른 프로세스, 파일, I/O 자원들에 대한 접근제어

2) 쓰레드 관련 사항

❖ 각각의 쓰레드들은 다음과 같은 사항을 개별적으로 갖는다

① 쓰레드 제어블록 (TCB)

✓ 실행 상태 (수행, 준비, 블록, ...), 우선순위 정보

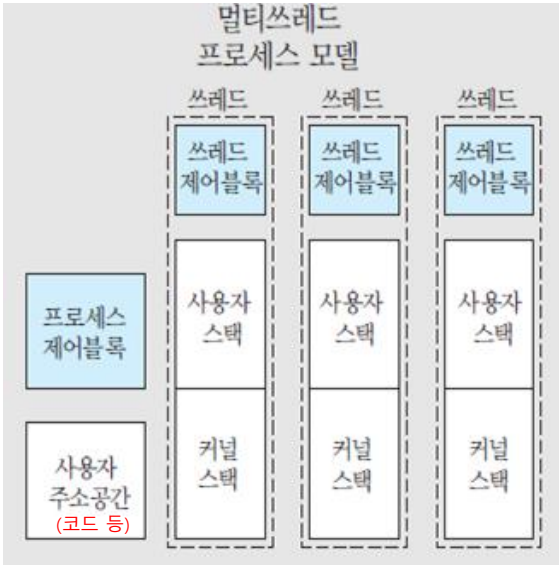
✓ 수행 중이 아닐 때 저장되는 처리기 문맥....

② 실행 스택

③ 지역 변수 저장을 위해 각 쓰레드가 사용하는 정적 저장소

❖ 프로그램 코드는 **reentrant(재진입 가능)** 구조

√ **복수 실행시 해당 실행주체(쓰레드)의 문맥들만 별도로 관리하면 공용 사용 가능**



코드 등의 자원을 공용으로 사용

한 프로세스 내에는 하나 이상의 쓰레드를 가지며
해당 프로세스 내의 모든 쓰레드들은 그 프로세스의 자원들을 공유하며 개별적으로 실행

- **쓰레드 모델 (그림 4.2)**
  - 같은 프로세스 내의 쓰레드들은 그 프로세스의 상태와 자원을 공유하며 쓰레드 단위로 실행

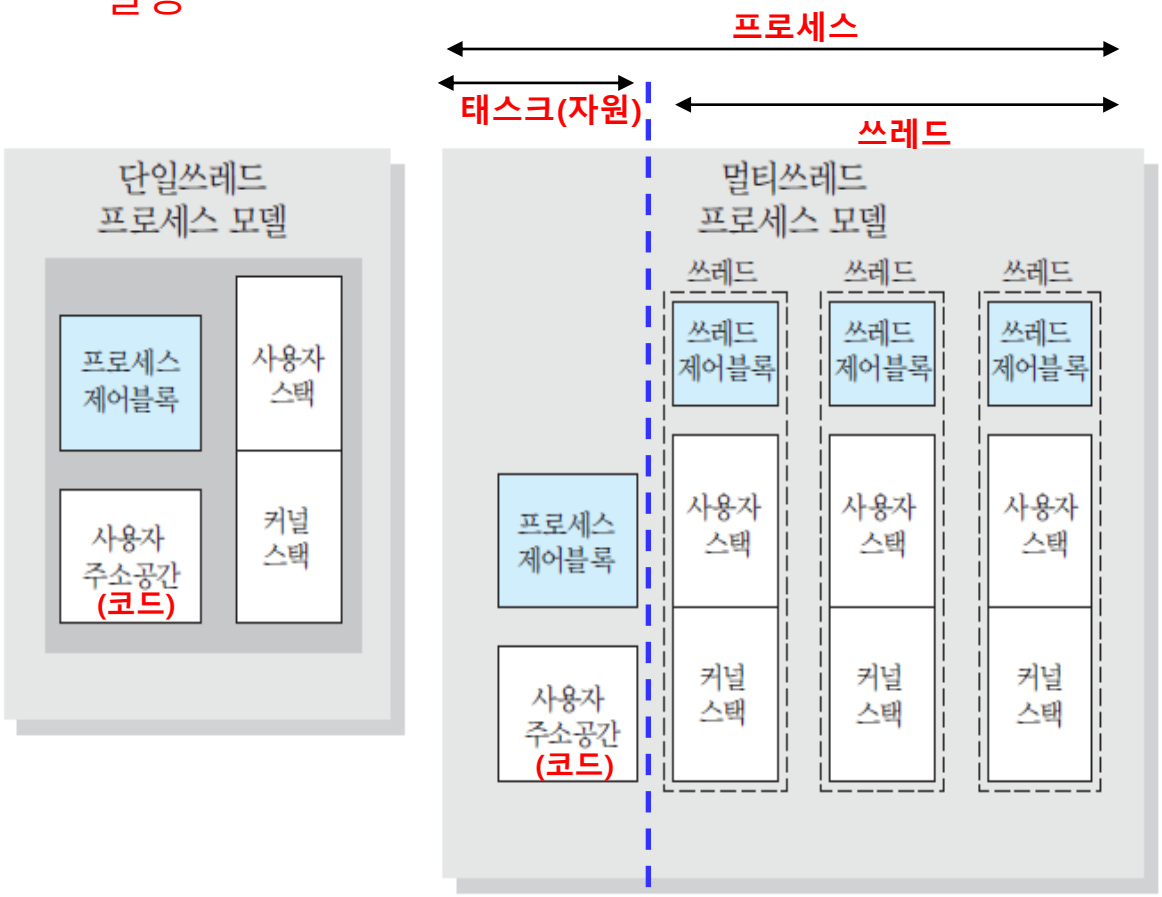
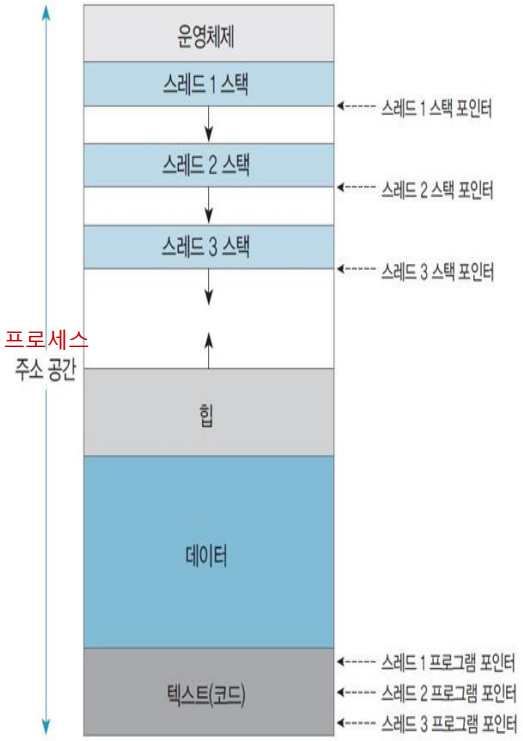


그림 4.2 단일쓰레드 프로세스 모델과 멀티쓰레드 프로세스 모델



같은 프로세스의 쓰레드 정보들은 동일한 프로세스내에 위치

- **다중 쓰레드의 적용 예 - 1**
  - 하나의 프로그램(프로세스)을 동시에 복수개 실행하는 경우 하나의 프로세스 이미지만을 사용 가능
  - 방법
    1. 복수개의 동일한 프로세스 이미지를 주기억장치에 만들지 않고 하나의 프로세스 이미지만을 주기억장치에 생성
    2. 프로세스 내에 추가적인 새로운 쓰레드를 생성
    3. 프로세스가 아닌 쓰레드 단위의 단기 스케줄링
    4. 각 쓰레드들은 하나의(공통의) 프로세스 이미지의 정보(코드)를 이용해 자신의 프로그램을 개별적으로 실행 → 쓰레드에게 필요한 최소 정보만을 분리해서 개별 사용
    5. 각 쓰레드들은 자신만의 쓰레드 제어블록에 자신만의 정보를 저장(상태, 처리기 문맥, 스택, PC 등등)해 다음번에 스케줄링이 되어도 하던 일을 계속 수행할 수 있게 한다.

- 기존 프로세스 단위의 관리에서의 문제점
  1. 프로세스 코드의 순차적인 실행으로 인한 오버헤드
  2. 시스템 호출이 포함된 프로세스의 지나친 대기시간
  3. 멀티처리기(코어)를 이용해도 하나의 프로세스는 하나의 처리기(코어)에서만 동작
  4. 동일한 프로그램의 복수 호출시에 동일 프로세스 이미지 복수개가 주기억장치에 생성

- **다중 쓰레드의 적용 예 - 1 - 계속**
  - ➢ 하나의 프로그램(프로세스)을 동시에 복수개 실행하는 경우 하나의 프로세스 이미지만을 이용하는 것이 가능

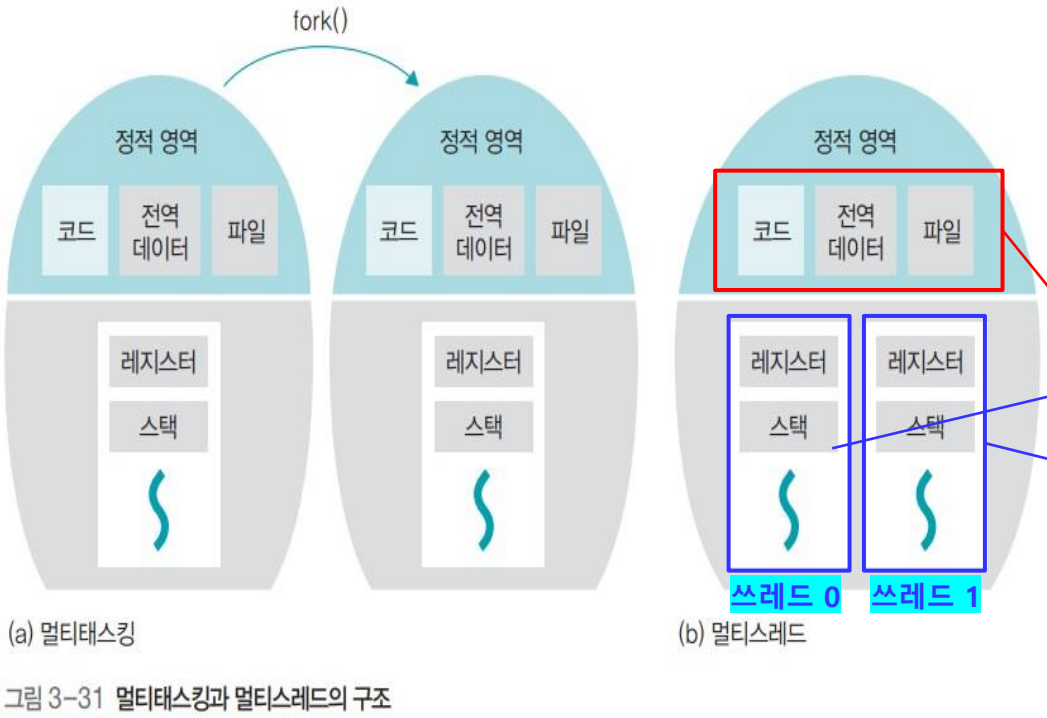
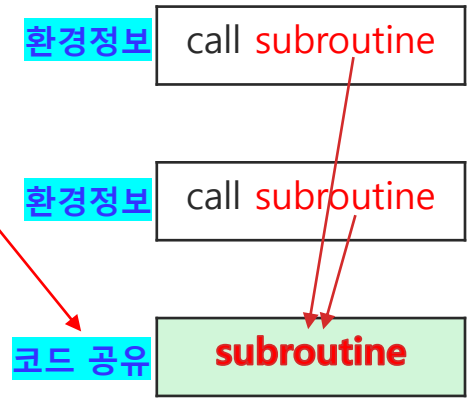


그림 3-31 멀티태스킹과 멀티스레드의 구조

개별 쓰레드들이 실행될 때에
정적영역(코드 및 전역변수 등)은
공통으로 사용하며
각 쓰레드들의 실행에 따라
변경되는 부분만을 별도로
쓰레드마다 별도로 정의해 사용



< 자원 공유 유사 개념 >

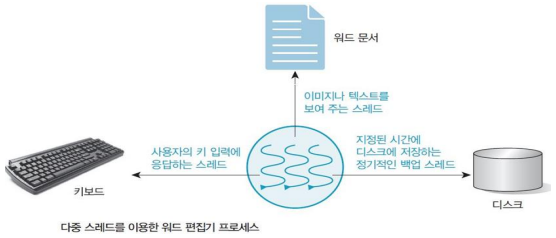
동일한 프로그램을 실행하는데 동일한 프로세스 이미지가 복수개가 생기므로 비효율적

동일한 프로그램을 복수개 실행해도 하나의 프로세스 이미지만을 사용하므로 효율적

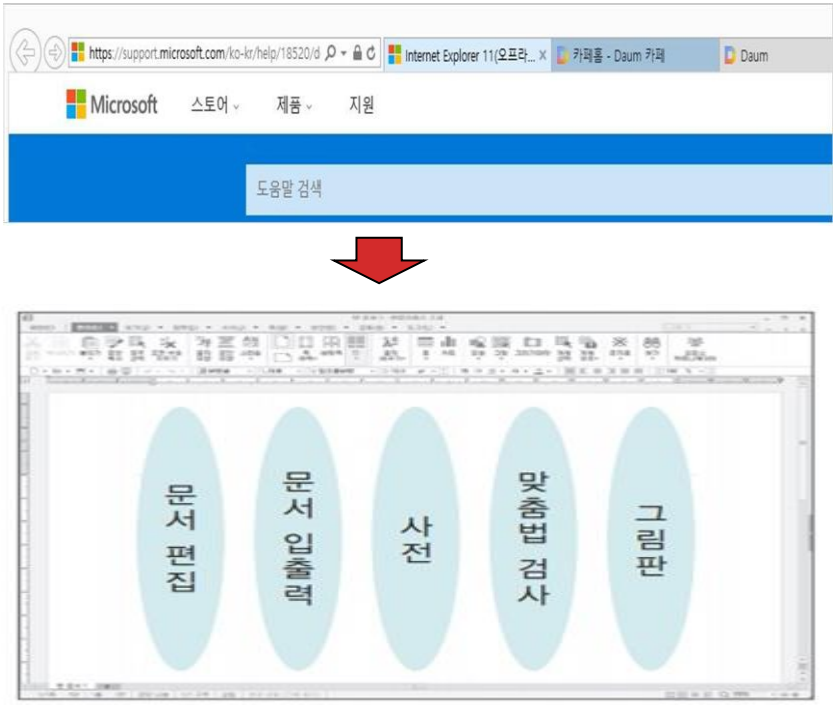
4.1 프로세스와 쓰레드(thread)

- **다중 쓰레드의 적용 예 - 2**
  - 한 프로세스에서 코드를 기능별로 쓰레드로 구분
    - 가능하면 기능적으로 코드를 분리(예: 입력, 출력, 계산..)
  - 프로세스 생성시에 모든 쓰레드들이 준비 큐로 진입
  - 프로세스가 아닌 쓰레드를 기본 단위로 단기 스케줄링
  - 쓰레드들은 자신만의 쓰레드 제어블록에 자신의 정보(상태, 처리기 문맥, 스택, PC 등)를 저장해 다음 스케줄링시에 하던 일을 계속 수행할 수 있게 함
  - 장점 (쓰레드 단위로 스케줄링)
    - 하나의 기능을 실행하면서 반복적으로 다른 기능들의 상황을 계속 조사할 필요가 없이 자기 일만 하면 된다 → ①
    - 동일 프로세스 내의 한 쓰레드가 블록되어도 나머지 쓰레드들은 수행이 가능 → ②
    - 동일 프로세스 내의 쓰레드들을 각기 다른 처리기에서 실행 가능(프로세스 이미지는 공용 주기억장치에 있다) → ③

- 기존 프로세스 단위의 관리에서의 문제점
  - ① 프로세스 코드의 순차적인 실행으로 인한 오버헤드
  - ② 시스템 호출이 포함된 프로세스의 지나친 대기시간
  - ③ 멀티처리기(코어)를 이용해도 하나의 프로세스는 하나의 처리기(코어)에서만 동작
  - ④ 동일한 프로그램의 복수 호출시에 동일 프로세스 이미지 복수개가 주기억장치에 생성



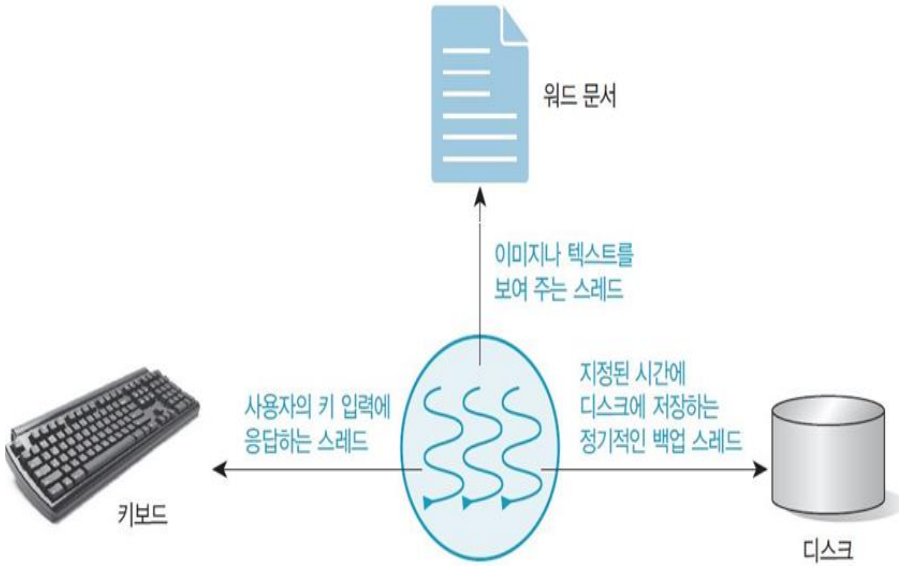
- **다중 쓰레드의 적용 예 - 2 - 계속**
  - 하나의 프로세스에서 코드를 기능별로 나누어 쓰레드로 구분함으로써
  - 동일 프로세스 내에 있는 각각의 쓰레드(기능 모듈)들이 별도로 단기 스케줄링(실행) 된다
    - 다른 기능 모듈(쓰레드)을 일일이 확인하지 않아된다
    - 하나의 쓰레드가 블록되어도 해당 프로세스의 나머지 쓰레드들은 정상동작이 가능



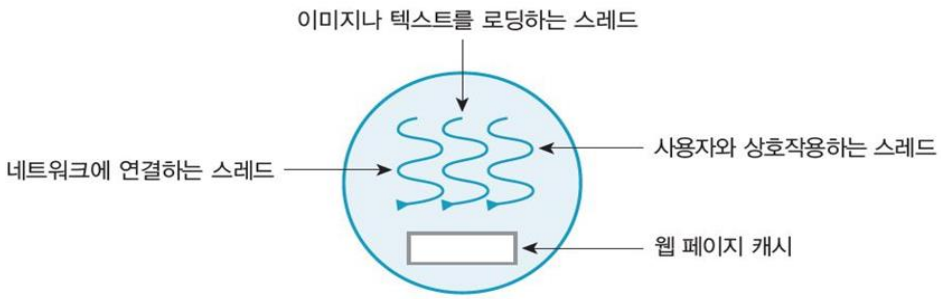
(a) 단일 스레드 (b) 멀티스레드

그림 3-34 단일 스레드와 멀티스레드의 구조

- **참고 : 다중쓰레드의 사용 예**

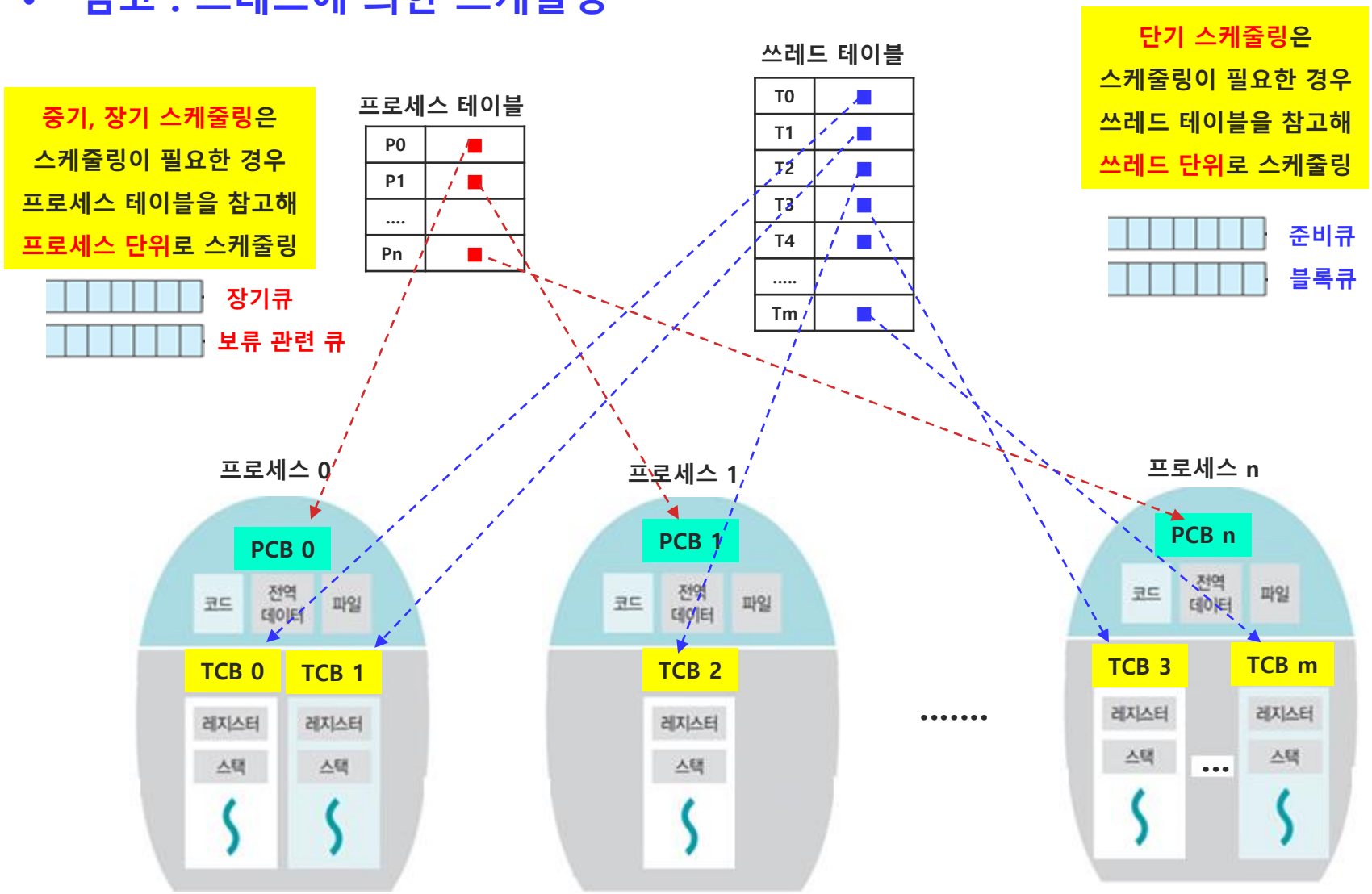


다중 스레드를 이용한 워드 편집기 프로세스



다중 스레드를 이용한 웹 브라우저 프로세스

- **참고 : 쓰레드에 의한 스케줄링**



- **쓰레드의 장점**

1) 새로운 프로세스의 생성보다 기존 프로세스 내에서의 새로운 쓰레드 생성 시간이 짧다
   - ❖ Mach의 쓰레드 생성시간이 UNIX의 프로세스 생성시간 보다 10배 빠르다
   - ❖ 생성된 프로세스의 자원을 활용 → 프로세스의 기존 주소공간, PCB 등을 공용화
2) 프로세스 종료 시간보다 쓰레드 종료 시간이 더 짧다
3) 프로세스들 간의 교환보다 한 프로세스 내의 쓰레드들 사이의 교환/교체 시간이 짧다
   - ❖ 동일 프로세스 내의 쓰레드들은 메모리 및 파일을 공유하기 때문에, 이들 쓰레드들은 커널의 개입없이 서로 통신 가능 (ULT)
4) 연관된 수행단위의 집합으로 구현되어야 하는 응용이나 기능이 있다면 독립적인 프로세스보다는 쓰레드의 모음으로 구성하는 것이 훨씬 효율적
   - ❖ 논리적으로 여러가지 다른 기능을 수행하는 프로그램의 구조(structure)를 단순화
   - ❖ 스프레드시트 프로그램 : 메뉴표시/입력 쓰레드 + 사용자명령수행/갱신 쓰레드
5) **멀티처리기**(multiprocessor)의 효율적 사용

- **쓰레드의 장점 (다른 교재들의 분류)** - 계속

➢ 프로세스에 포함된 쓰레드들은 공통의 목적 달성을 위해 **병렬**로 수행 가능

- ❖ 자원을 공유하여 하나의 제어기에서 동시(병행) 작업 가능
- ❖ 하나의 프로세스가 다른 제어기에서 프로그램의 다른 부분을 동시(병렬)에 실행 가능
- ❖ 응용 프로그램 하나가 비슷한 작업들을 여러 개 수행 → 예) 서버단의 프로세스

1) 사용자에 대한 응답성 증가

- ❖ 응용 프로그램의 일부분이 봉쇄되어도 프로그램 실행을 계속 허용하여 사용자 응답성이 증가 → 예) 웹 브라우저 : 파일적재 쓰레드 + 사용자 상호작용 쓰레드

2) 프로세스의 자원과 메모리 공유로 성능 향상

- ❖ 쓰레드는 그들이 속한 프로세스의 자원과 메모리를 공유하므로, 응용 프로그램 하나가 같은 주소 공간에서 여러 개의 쓰레드를 실행, 시스템 성능 향상 → 쓰레드간 통신, 자원 공유

3) 경제성

- ❖ 한 프로세스의 자원을 공유하므로 프로세스를 생성하는 것 보다 오버헤드가 적다(속도/공간)

4) 다중 프로세서 구조 활용 가능

- ❖ 다중 프로세서 구조에서 각 쓰레드는 다른 프로세서에서 병렬로 실행 가능

- **단일사용자 멀티프로세싱 시스템에서의 쓰레드의 사용 4가지 예** (뒷장 예제)

1. 전면(foreground) 작업과 후면(background) 작업 → 응용의 속도 향상
   - ❖ 예를 들어 스프레드시트 프로그램에서 하나의 쓰레드가 메뉴를 나타내고 사용자 입력을 읽는 중에, 다른 쓰레드는 사용자 명령을 수행하고 스프레드시트를 갱신할 수 있다.
2. 비동기(asynchronous) 처리 → 프로그램의 비동기적 요소들을 쓰레드를 통해 구현
   - ❖ 예를 들어 정전으로부터 보호하기 위해 1분마다 메모리(RAM) 버퍼의 내용을 디스크로 기록하는 워드 프로세서를 설계할 수 있다. 이를 위한 쓰레드가 생성될 수 있는데, 유일한 업무는 주기적인 백업이며 운영체제를 통해 직접 자신을 스케줄 한다. 이때 시간을 검사하거나 또는 입력 및 출력을 조정하기 위해 주 프로그램 내에 복잡한 코드를 작성할 필요는 없다.
3. 빠른 수행 → 한 프로세스내의 여러 쓰레드들은 실제적으로 동시에 수행될 수 있다.
   - ❖ 멀티프로세서 시스템에서는 한 프로세스 내의 여러 쓰레드들을 실제적으로 동시에 다른 작업을 할 수 있다. 따라서 한 쓰레드가 입출력 작업 완료를 기다리면서 블록(block)될지라도, 또 다른 쓰레드가 수행될 수 있다.
4. 모듈 프로그램 구조
   - ❖ 다양한 활동 혹은 다양한 입출력 연산을 포함하고 있는 프로그램의 경우, 쓰레드들을 사용하여 설계하고 구현하는 것이 편리하다.

- **다중쓰레드와 운영체제**
  - 쓰레드를 지원하는 운영체제에서는 단기 스케줄링(디스패칭)이 **쓰레드를 기초**로 이루어진다. 따라서 수행에 관련된 대부분의 정보가 쓰레드 수준의 자료구조에 의해 유지
    → 쓰레드 제어 블록(TCB: Thread Control Block)
  - 몇몇 작업들은 프로세스 내의 모든 쓰레드들에게 영향을 미치므로 운영체제는 이를 프로세스 수준에서 관리해야 함
    - 보류(suspend)상태는 프로세스 단위로 처리되므로 쓰레드 관리에서는 없다
      - √ 보류된 프로세스 내의 모든 쓰레드들은 수행이 불가능 하다
      - √ 프로세스 = 태스크 + 쓰레드들
    - 하나의 프로세스가 종료되면 그 프로세스 내의 모든 쓰레드들도 함께 종료된다
      - √ 프로세스 = 태스크 + 쓰레드들

- **쓰레드의 기능**
  - 프로세스처럼 쓰레드도 수행 상태를 가지며 서로 동기화될 수 있다
  - 쓰레드 주요 상태
    1) 수행
    2) 준비
    3) 블록
    ※ 보류(Suspend) :
      - √ 프로세스 수준(자원: 주기억장치)의 개념이라 쓰레드에는 보류상태가 없다
      - √ 보류는 프로세스 단위의 행위이므로 모든 소속된 모든 쓰레드들도 스왑된다

- **쓰레드의 상태 전이와 관련된 쓰레드 연산**

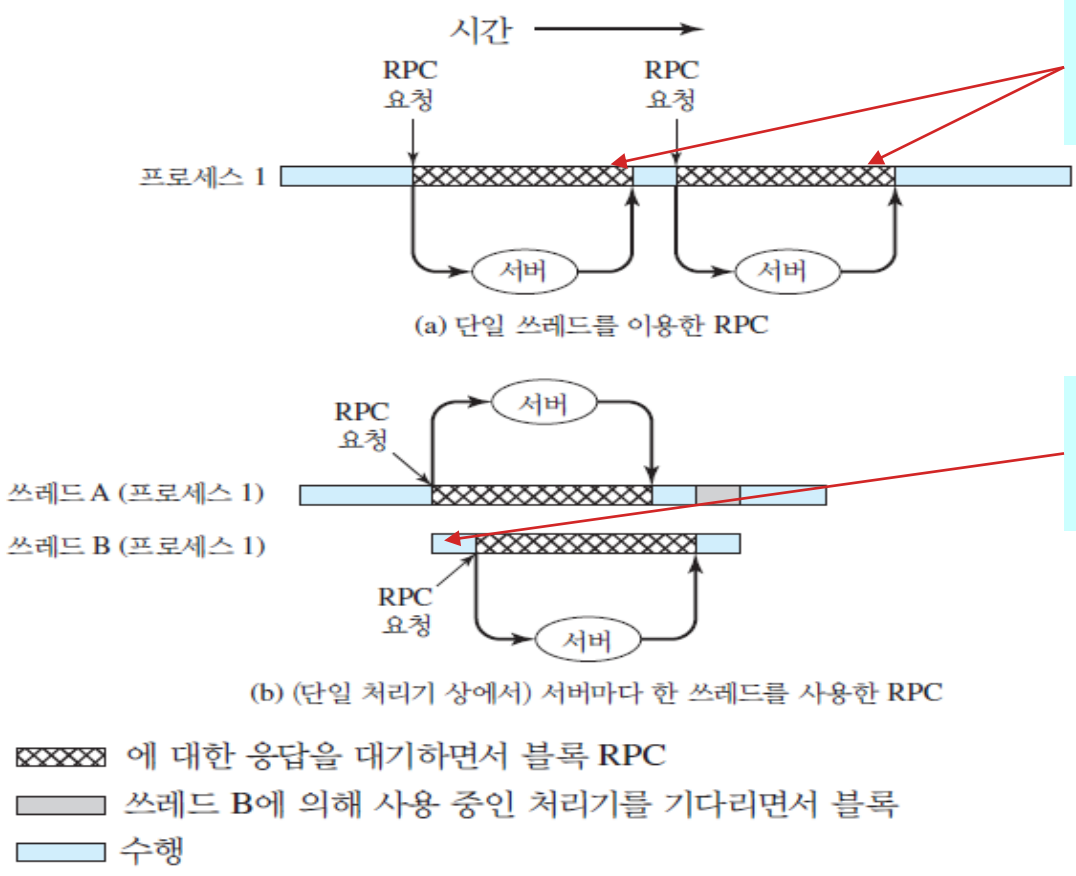
1) 생성(Spawn)
   - ❖ 새로운 프로세스가 생성될 때 프로세스 내부에 정의된 모든 쓰레드들도 함께 생성
   - ❖ 기존 쓰레드가 동일 프로세스 내에서 새로운 쓰레드 생성 가능
     - √ 새로운 쓰레드는 자신의 레지스터 문맥과 스택공간을 가지며 **준비 큐**에 위치

2) 블록(Block)
   - ❖ 사건을 기다릴 때 블록된다
   - ❖ 이때 처리기는 동일 프로세스 내의 쓰레드 또는 다른 프로세스 내의 준비된 쓰레드를 수행

3) 비블록(Unblock)
   - ❖ 사건이 발생되면 준비 큐로 이동

4) 종료(Finish)
   - ❖ 쓰레드의 작업이 완료되면 레지스터 문맥과 스택을 해제

- **다중쓰레드의 장점의 예**
  - 그림 4.3: RPC : Remote Procedure Call



단일쓰레드로 작성하면 하나의 RPC 요청을 한 후 그 결과가 올 때까지 계속 기다려야 한다.

다중쓰레드로 작성하면 하나의 RPC 요청을 한 후 그 결과가 올 때까지 계속 기다리지 않아도 된다.

그림 4.3 쓰레드를 사용한 원격 프로시저 호출(RPC)

- **다중쓰레드의 장점의 예**
  - 그림 4.4: 단일 처리기 상에서 다중쓰레딩의 예
    - 2개의 프로세스내에서 3개의 쓰레드가 한 처리기 상에서 번갈아 수행된다

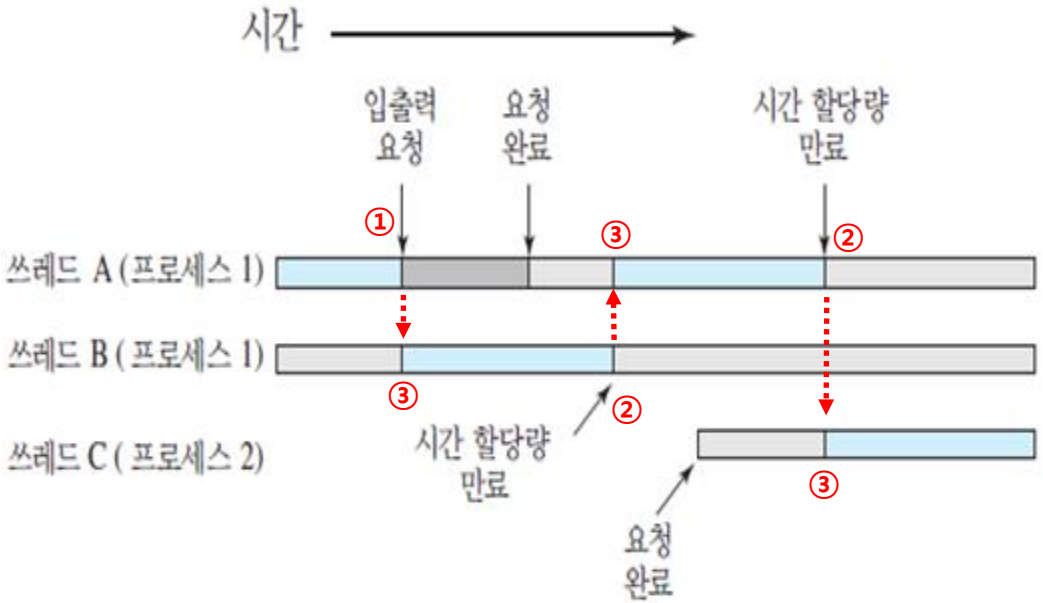


그림 4.4 단일 처리기 상에서 멀티쓰레딩 예

수행중인 쓰레드가
①블록(→블록상태)되거나
②정해진 시간 할당을 모두 소비하면
(→ 준비상태)
③수행흐름이 다른 쓰레드로 넘어간다.

'10장 멀티프로세서, 멀티코어 및 실시간 스케줄링'에서 쓰레드 관련 스케줄링을 다루나 우리는 학습하지 않는다

- **쓰레드 동기화**(synchronization)
  - 프로세스처럼 쓰레드도 수행 상태를 가지며 서로 동기화될 수 있다
  - 한 프로세스 내의 모든 쓰레드들은 동일 주소 공간 및 자원들을 공유
  - 공유 자원 :
    - 변수, 파일, 이중 연결 리스트(double linked lists)
  - 공유 자원에 대해 동시 접근시(특히, 갱신 시) **일관성 유지 기법 필요**
    - 예로, 한 공유 변수에 대해 읽기 연산은 동시 접근이 가능하나, 쓰기 연산은 한 순간에 하나만 가능
    - 동기화 관련 내용은 제 5장 및 제 6장에서 다룸

이중 연결 리스트(double linked lists) :
연결 목록의 하나로, 목록의 각 노드에 목록 중 직후(直後)의 노드와 직전의 노드를 가리키는 2개의 지시자(pointer)가 있는 것. 데이터 요소를 전후 양방향으로 삽입하거나 삭제, 탐색할 수 있으므로 단방향 연결 목록보다 기억 장소는 더 많이 필요하지만 조작 효율이 높다.

- **운영체제에서 쓰레드의 구현은 여러수준에서 구현이 가능하다**

1) 사용자 수준 쓰레드(User-level thread, **ULT**)

❖ 커널은 쓰레드의 존재를 모르며, 쓰레드 라이브러리가 선택된 프로세스내의 쓰레드를 스케줄링

2) 커널 수준 쓰레드(Kernel-level thread, **KLT**)

❖ 커널 지원 쓰레드(kernel-supported thread) 또는 경량 프로세스(lightweight process)라고도 부르기도 한다(다른 교재). 커널은 프로세스가 아닌 모든 쓰레드를 기준으로 단기스케줄링을 실시

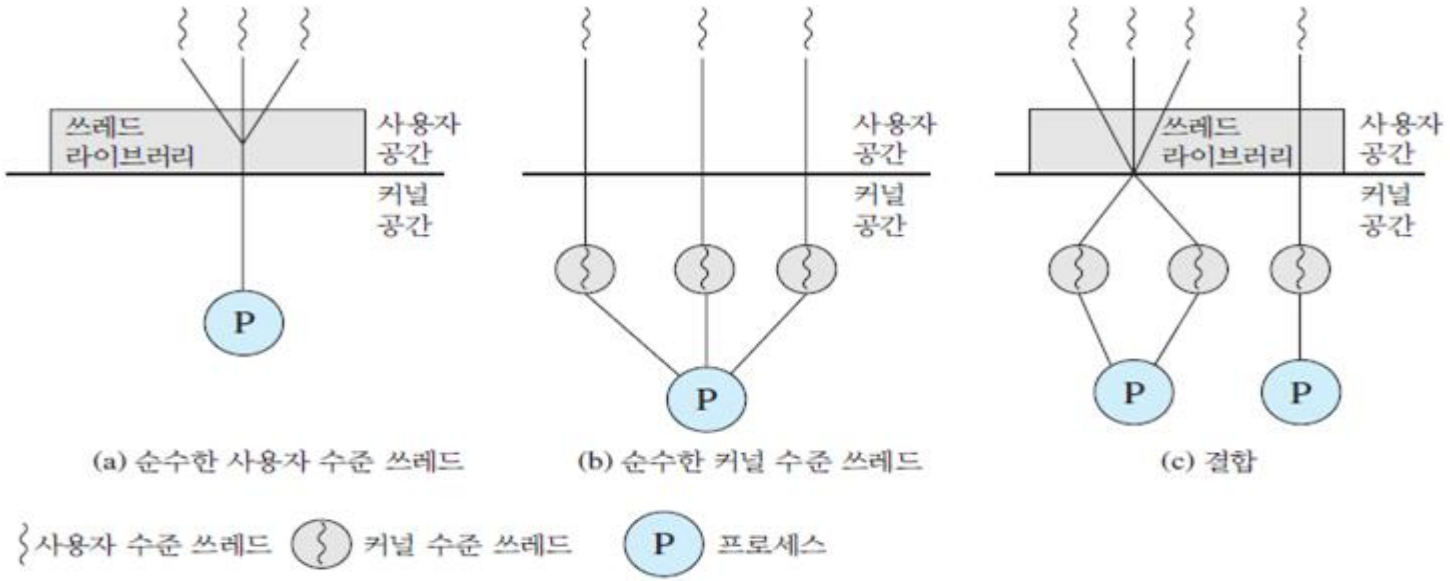
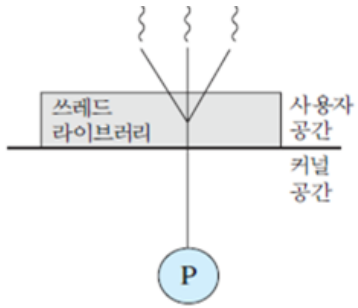
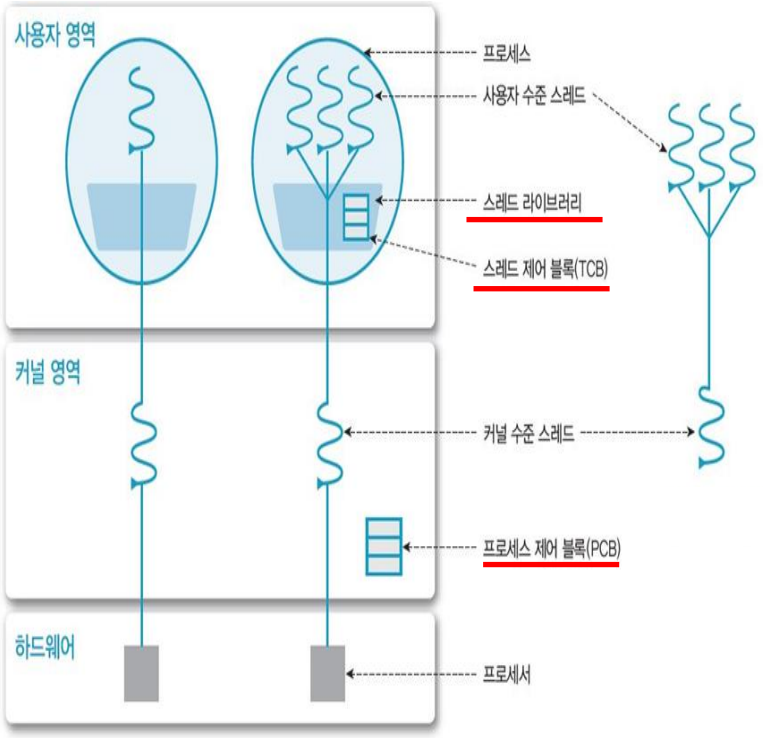


그림 4.5 사용자 수준 쓰레드와 커널 수준 쓰레드

- **사용자 수준 쓰레드(ULT)**
  - 커널은 쓰레드의 존재를 모름
    - 커널은 프로세스 단위로만 스케줄링(준비큐)
  - 응용(쓰레드 라이브러리)이 모든 쓰레드를 관리
  - 쓰레드 라이브러리
    - ULT 관리 루틴으로 구성된 패키지이며
    - 쓰레드의 생성, 제거, 데이터 전송, 동기화, 스케줄, 문맥교환을 수행
    - 단기 스케줄러에 의해 프로세스가 선택되면 해당 프로세스에게 주어진 시간 내에서 쓰레드 라이브러리가 프로세스 내부의 쓰레드들을 스케줄링(각 프로세스 내에 별도의 쓰레드용 준비 큐)
    - 쓰레드 라이브러리를 이용해 모든 응용을 멀티쓰레드 기반으로 프로그래밍할 수 있다
    - POSIX Pthreads, Windows, JAVA

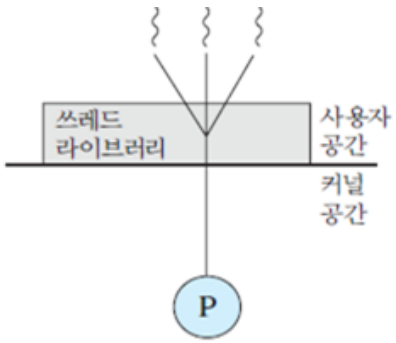


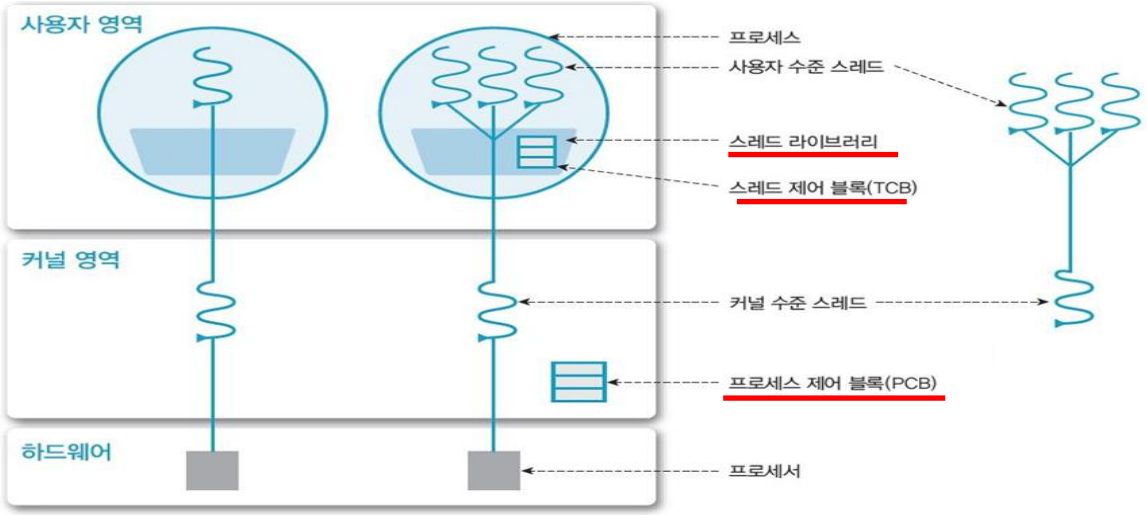


사용자 수준 스레드 : 다대일(n :1) 매핑

- **사용자 수준 쓰레드(ULT)의 장점**

1) 동일 프로세스 내의 쓰레드 교환/교체 시에 커널 모드 권한이 불필요
   - ❖ 2번의 모드 전이 오버헤드 절약 (기존 단일 쓰레드 : 시스템호출 → ①커널 →② 다른 프로세스)
2) 운영체제 스케줄러와 무관하게 특정 응용에 적합한 쓰레드 스케줄링의 적용 가능
3) 커널과 무관한 쓰레드 라이브러리를 사용하므로 모든 OS에서 수행 가능
   - ❖ 표준화된 쓰레드 라이브러리(POSIX Pthreads, 약어: pthread)를 사용하면 컴파일만 다시 하면 동작

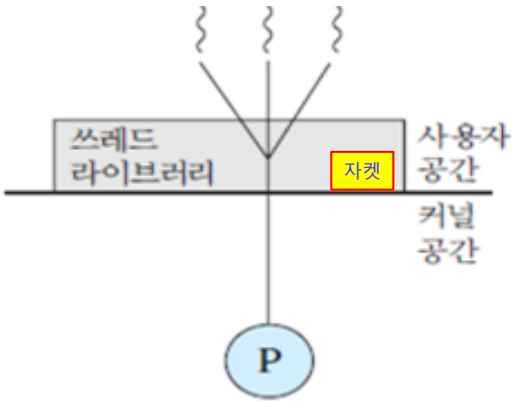




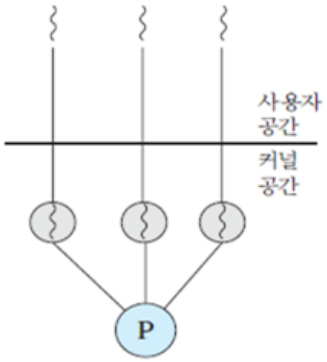
사용자 수준 스레드 : 다대일(n :1) 매핑

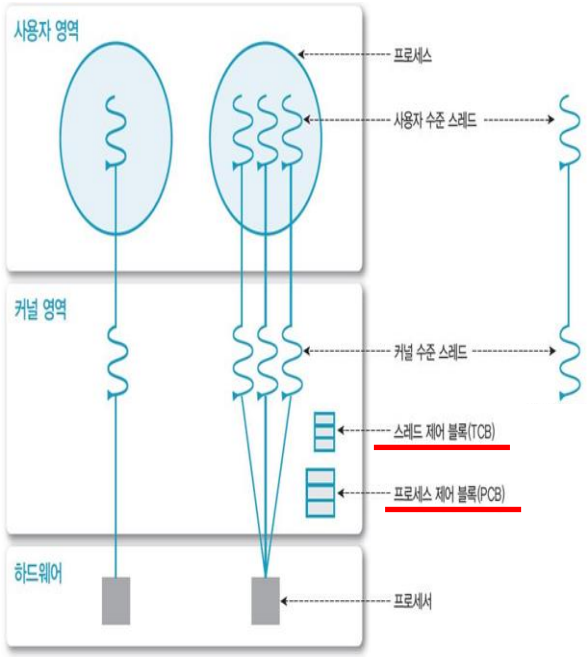
- **사용자 수준 쓰레드(ULT)의 단점**
  - 한 쓰레드가 블록 상태를 유발하는 시스템 호출을 수행할 경우, ~~자신뿐만 아니라 그 프로세스 내의 모든 다른 쓰레드들도 블록됨~~
    - 쓰레드가 블록되는 것은 아니고 해당 프로세스 자체가 블록됨 → 결국 쓰레드들도 동작 불가
  - 자켓팅(Jacketing) 기술
    - 프로세스의 블록을 방지하는 기술로 쓰레드 라이브러리에 구현(프로그램)
    - blocking system call → non-blocking system call
      1. 쓰레드에서 시스템 호출시 커널이 아닌 응용수준의 입출력용 자켓 루틴을 호출
      2. 자켓 루틴은 입출력장치의 사용여부를 검사해 사용중이면 해당 쓰레드는 블록상태로 바뀌고 다른 쓰레드가 수행(쓰레드 라이브러리에 의해) → **프로세스는 블록이 되지 않는다**
      3. 자켓 루틴이 입출력을 수행하고 결과를 쓰레드에게 넘겨줌
  - 다중처리기의 장점을 살리지 못함.
    - 커널 루틴 자체는 다중쓰레딩 될 수 없다.
    - 응용을 멀티쓰레드가 아닌 멀티프로세스로 작성

      → 쓰레드 교환이 아닌 프로세스 교환



- **커널 수준 쓰레드 (KLT)**
  - **커널**이 프로세스 및 쓰레드에 대한 문맥 정보를 관리
    - 응용내에서는 단순히 커널 쓰레드 기능용 API만 존재
    - 커널 내부에 모든 프로세스 및 쓰레드 정보
  - 쓰레드에 대한 스케줄링이 커널 수준에서 수행됨
  - 장점
    1) 한 프로세스내의 여러 쓰레드들을 각기 다른 처리기에서 동시 수행 가능하게 할당
    2) 동일 프로세스 내의 쓰레드가 블록되어도 다른 쓰레드들은 수행 가능
    3) 커널 루틴 자체도 멀티 쓰레딩이 가능하다는 장점
  - 단점
    - 쓰레드 전환시 커널 모드로의 전환에 의한 오버헤드





커널 수준 스레드 : 일대일(1:1) 스레드 매핑

- **결합된 접근 방법 (Combined approach)**
  - ULT와 KLT의 결합
  - 대부분의 쓰레드 생성, 동기화 및 스케줄링을 사용자 공간에서 실행
  - ULT들은 같거나 적은 수의 KLT와 결합
    - √ 조건에 따라 KLT의 개수 조정 가능
  - 한 응용의 쓰레드들이 다수의 처리기에서 수행이 가능하며 블록형 시스템 호출이 전체 프로세스를 블록시키지 않는다.(KLT의 장점)
  - 윈도우 계열, 리눅스 계열, Solaris가 있음
    - √ ULT : KLT = 1:1 구조로 사용

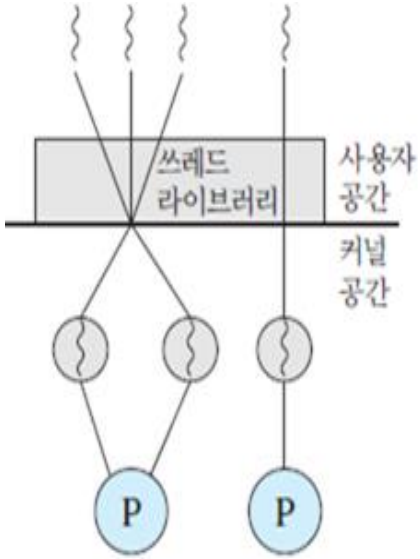


표 4.1 쓰레드와 프로세스 연산지연(μs)

연산	사용자 수준 쓰레드	커널 수준 쓰레드	프로세스
Null Fork	34	948	11,300
Signal–Wait	37	441	1,840

- **쓰레드와 프로세스 간의 관계 (개수)**

표 4.2 쓰레드와 프로세스간의 관계

쓰레드:프로세스	설 명	시스템 예
1:1	수행 중인 각 쓰레드는 자신의 주소 공간과 자원을 갖는 유일한 프로세스이다.	대부분의 전통적인 UNIX 시스템
M:1	프로세스는 주소 공간과 동적인 자원 소유권을 정의하며, 여러 쓰레드가 이 프로세스 내에서 생성 및 수행될 수 있다.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	쓰레드는 한 프로세스 환경에서 다른 프로세스 환경으로 이동할 수 있다. 이것은 쓰레드가 다른 시스템 간을 쉽게 이동할 수 있도록 해준다.	Ra(Clouds), Emerald (생략)
M:N	1:M과 M:1의 특성을 혼합한 것이다.	TRIX (생략)

- **쓰레드 프로그래밍 기법**
  - thread Library를 이용하는 방법 → 프로그래머가 API를 이용해 직접 작성
    - 3대 API: ①Pthreads, ②JAVA threads, ③Windows threads
    - ULT는 물론 KLT도 지원
  - 암묵적 쓰레딩 → 컴파일러한테 쓰레드 관련한 일을 전담 시킴
    1) OpenMP
       - √ 프로그램 작성시 쓰레드를 병렬적으로 수행하고자 하는 부분을 지정
       - √ #progma omp pararrel { } ← { } 부분의 코드는 컴파일러가 저절로 코어 개수만큼 병렬적으로 수행
    2) Grand Central Dispatch
       - √ 주로 애플에서 이용하는 기술로 OpenMP와 마찬가지로 병렬적으로 수행하고자 하는 부분을 지정하면 자동으로 병렬적 수행

쓰레드 풀 (Thread Pool) 기술
서버와 같은 경우는 많은 쓰레드들을 만들어야 하는데 이는 시간도 많이 걸리고 처리기 자원도 조절해야 되는 등의 불편함을 없애기 위해서 풀(pool)이라는 곳에 미리 여러 개의 쓰레드를 제한된 개수만큼 만들어 놓고 요청이 들어오면 하나씩 할당해 주는 방식

4.2 프로세스와 쓰레드

- **쓰레드 프로그래밍 기법**

< pthread의 사용 예 >

```
001: #include <stdio.h>
002: #include <stdlib.h>
003: #include <pthread.h>
004:
005: #define TH_NUM 4
006: #define N 100
007:
008: static void *thread_increment(void *array)
009: {
010:     int i;
011:     int *iptr;
012:
013:     iptr = (int *)array;
014:     for(i = 0; i < N/TH_NUM; i++) {
015:         iptr[i] += 1 ;
016:     }
017:     return NULL;
018: }
019:
020: int main(void)
021: {
022:     int i;
023:     pthread_t thread[TH_NUM];
024:
025:     int array[N];
026:
027:     /* 배열 초기화 */
028:     for(i = 0; i < N; i++) {
029:         array[i] = i;
030:     }
031:
032:     /* 병렬처리 시작 */
033:     for(i = 0; i < TH_NUM; i++) {
034:         if(pthread_create(&thread[i], NULL, thread_increment, array + i * N/TH_NUM) != 0)
035:         {
036:             return 1;
037:         }
038:     }
039:
040:     /* 스레드 종료 대기 */
```

쓰레드들을 직접 생성

쓰레드들의 실행

< OpenMP의 사용 예 >

```
001 : #include<stdio.h>
002 : #include<stdlib.h>
003 : #in[lude<omp.h>
004 : #define N 100
005 : #define TH_NUM 4
006 :
007 : int main ()
008: {
009 :     int i;
010 :     int rootBuf[N];
011:
012 :     omp_set_num_threads(TH_NUM);
013 :
014:     /* 배열 초기화 */
015:     for(i = 0; i < N; i++) {
016:         rootBuf[i] = i;
017:     }
018:
019:     /* 병렬처리 시작 */
020: #pragma omp parallel for
021:     for(i = 0; i < N; i++) {
022:         rootBuf[i] = rootBuf[i] + 1;
023:     }
024:
025:     return(0);
026: }
```

쓰레드들의 생성

선택여부에 따라 컴파일러가 쓰레드 수를 자동으로 조절

- **멀티코어 상에서의 소프트웨어 성능**

➢ **Amdahl(암달)의 법칙**

❖ 목적

√ 멀티코어(멀티프로세서)가 얼마만큼의 최대 성능 향상이 있는지 계산하는 데 사용

❖ 가정

√ 프로그램의 실행 시간이 본래적으로 순차적으로 동작하는(병렬화 할 수 없는) 코드를 포함하는 (1-*f*)의 실행시간과 스케줄링 부하없이 병렬로 처리가능한 코드를 포함한 f의 실행시간으로 구성

❖ 수식

$$\text{속도향상} = \frac{\text{단일 처리기 상에서 프로그램을 실행한 시간}}{N\text{개의 병렬 처리기 상에서 프로그램을 실행한 시간}} = \frac{1}{(1-f) + \frac{f}{N}}$$

❖ 결론

√ 프로그램 성능을 개선할 때 가장 많은 시간이 소요되는 곳에 집중하라 → 병렬로 처리하는 부분의 코드를 최대화 해라

- **멀티코어 상에서의 소프트웨어 성능**

➢ **Amdahl의 법칙 (원래의 일반적인 의미)**

❖ 의미

√ 컴퓨터 시스템의 일부를 개선할 때 전체적으로 얼마만큼의 최대 성능 향상이 있는지 계산하는데 사용

❖ 수식

√ 어떤 시스템을 개선하여 전체 작업 중 P%의 부분에서 S배의 성능이 향상되었을 때 전체 시스템에서 최대 성능 향상은 다음과 같다

$$\frac{1}{(1-P)+\frac{P}{S}}$$

❖ 예

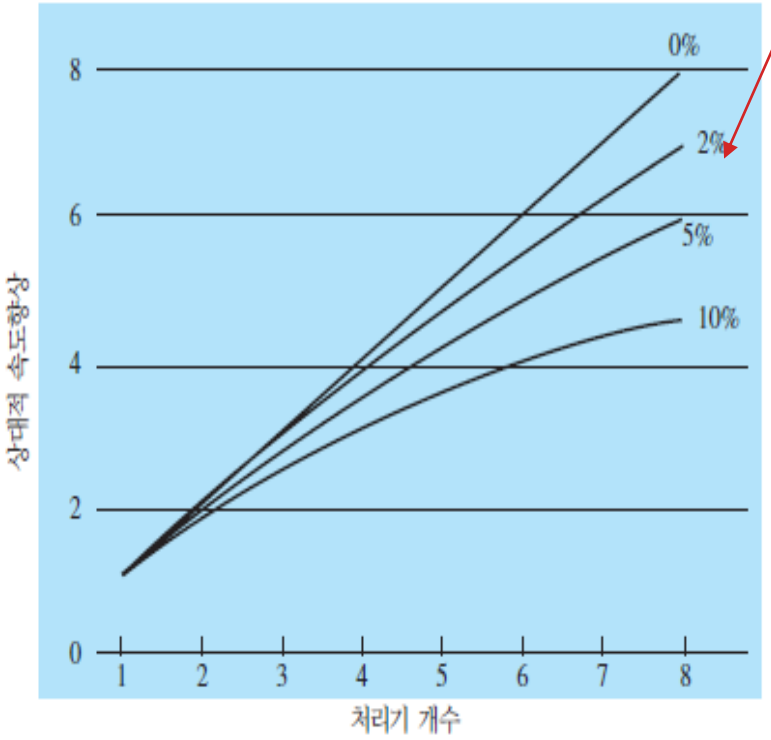
√ 어떤 작업의 40%에 해당하는 부분의 속도를 2배로 늘릴 수 있다면 P는 0.4, S는 2이고 최대 성능 개선은 다음과 같다

$$\frac{1}{(1-0.4)+\frac{0.4}{2}} = 1.25$$

- **멀티코어가 성능에 미치는 영향**

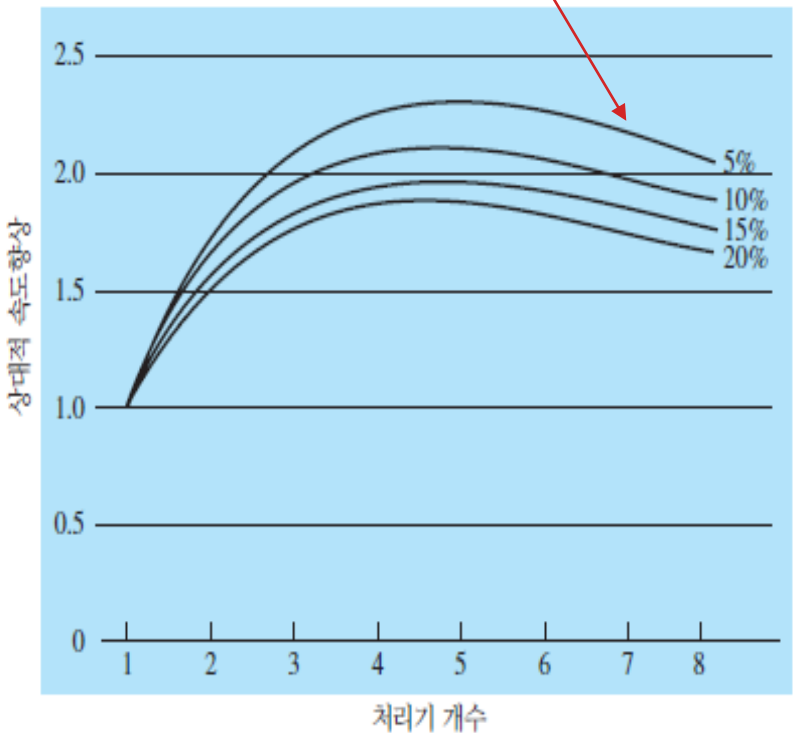
순차코드의 영향

멀티프로세서 오버헤드 고려



(a) 순차 수행 비율이 0%, 2%, 5%, 10% 일 때 속도 향상

Amdahl의 법칙의 결과
(이론적인 접근)



(b) 오버헤드를 고려한 속도향상

오버헤드를 고려한 결과
(통신, 작업분배, 캐시 일관성 작업 등)

- **멀티코어가 성능에 미치는 영향 – 개선된 성능**
  - 앞의 문제를 개선하기 위해 하드웨어 구조, 운영체제, 미들웨어, 데이터베이스 응용 소프트웨어의 순차적인 영향을 줄이기 위한 노력을 한 데이터베이스 응용의 예

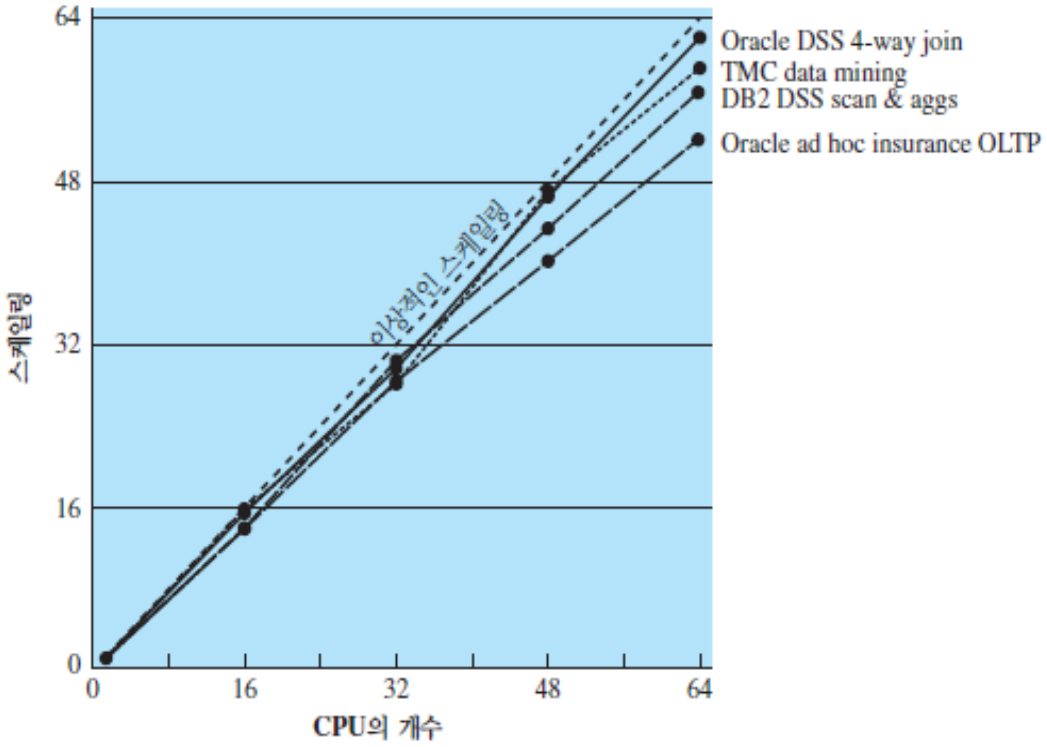


그림 4.8 멀티프로세서 하드웨어 상에서 데이터베이스 작업량의 스케일링

- **멀티코어 성능 향상이 많은 응용**
  - 멀티쓰레드화된 네이티브 응용:
    - 멀티쓰레드화된 응용은 소수의 고도로 쓰레드화된 프로세스를 갖는 특징을 갖는다. 쓰레드화된 응용의 예는 IBM사의 Lotus Domino와 Oracle사의 Siebel 고객 관계 관리(CRM)가 있다.
  - 멀티프로세스 응용:
    - 멀티프로세스 응용은 다수의 단일 쓰레드화된 프로세스들이 존재하는 특징을 갖는다. 멀티프로세스 응용의 예는 Oracle 데이터베이스와 SAP, PeapleSoft가 있다.
  - 자바 응용:
    - 자바 응용은 근본적인 방식으로 쓰레딩을 포함하고 있다. 자바 언어가 멀티쓰레드화된 응용을 개발하는데 매우 용이할 뿐만 아니라 자바 가상머신도 자바 응용에 대한 스케줄링과 메모리 관리를 지원하는 멀티쓰레드화된 프로세스이다. 멀티코어 자원으로부터 직접적인 혜택을 받는 자바 응용은 Sun사의 Java Application Server, BEA사의 Weblogic, IBM사의 Websphere, 오픈소스인 Tomcapt 응용 서버와 같은 응용 서버들을 포함한다.
  - 멀티(다중)인스턴스 응용:
    - 개별적인 응용이 많은 수의 쓰레드를 사용해서 속도를 향상시키지 못할지라도 다수의 응용 인스턴스를 멀티코어 구조상에서 병렬적으로 실행함으로써 속도를 향상 시킬 수 있다. 만약 다수의 응용 인스턴스가 어느 정도의 격리(isolation)를 요구한다면 가상 기술(운영체제의 하드웨어에 대한)을 사용하여 각각의 인스턴스마다 독립되고 안전한 환경을 제공할 수 있다.

요약

- **프로세스를 세분화하여 하나 이상의 실행단위로 쪼갠 쓰레드**

  1) 프로세스와 쓰레드의 차이

  2) 쓰레드와 관련된 기본 설계 이슈

  3) 사용자 수준의 쓰레드와 커널 수준의 쓰레드의 차이

수고 하셨습니다.

주1 : IT CookBook, 운영체제(개정판) 그림으로 배우는 원리와 구조 – 구현회
주2 : 생능출판사-운영체제[개정판]-박규석
주3 : 자체 정리
주4 : IT CookBook, 쉽게 배우는 운영체제-조성호