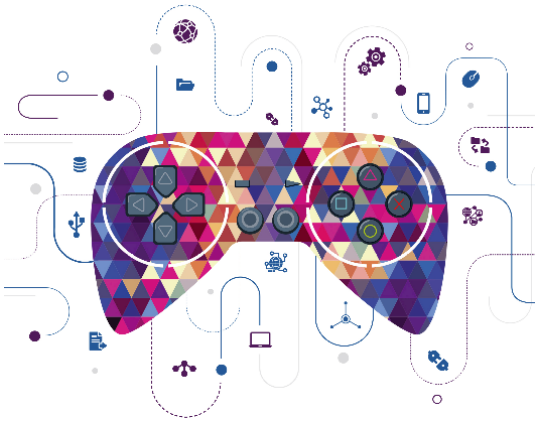


그림으로 배우는 구조와 원리

운영체제 개정 3판

Chatpter 03
프로세스와 스레드

- 프로세스와 스레드의 개념
- 프로세스의 상태 변화 과정
- 프로세스의 생성과 종료 등 프로세스 작업
- 프로세스와 스레드 차이를 이해하고, 스레드의 장점
- 사용자 수준 스레드와 커널 수준 스레드의 장단점

Section 01 프로세스의 개념과 상태변화(1. 프로세스의 개념)

프로세스process 의 정의(1960년대 멀틱스multics 운영체제에서 처음 사용)

- IBM 운영체제에서의 작업task
- 실행 중인 프로그램 가장 일반적인 정의
- 비동기적asynchronous 행위
- 실행 중인 프로시저
- 실행 중인 프로시저의 제어 추적
- 운영체제에 들어 있는 프로세스 제어 블록PCB
- 프로세서에 할당하여 실행할 수 있는 개체 디스패치dispatch가 가능한 대상

프로그램

- 저장장치에 저장되어 있는 정적인 상태

프로세스

- 실행을 위해 메모리에 올라온 동적인 상태

1. 프로세스의 개념

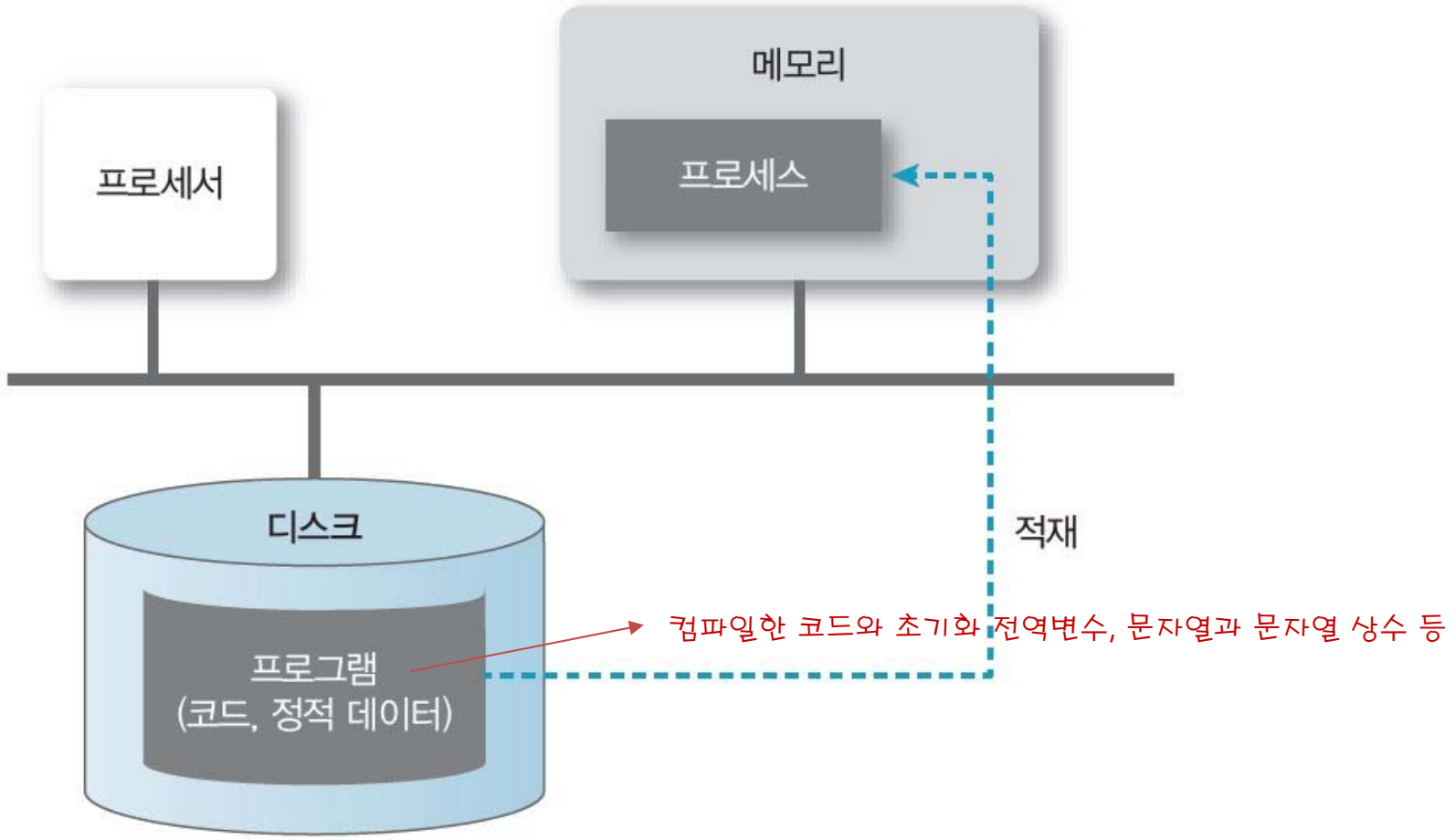


그림 3-1 프로그램과 프로세스 : 프로그램이 메모리로 적재되면 프로세스가 됨

1. 프로세스의 개념

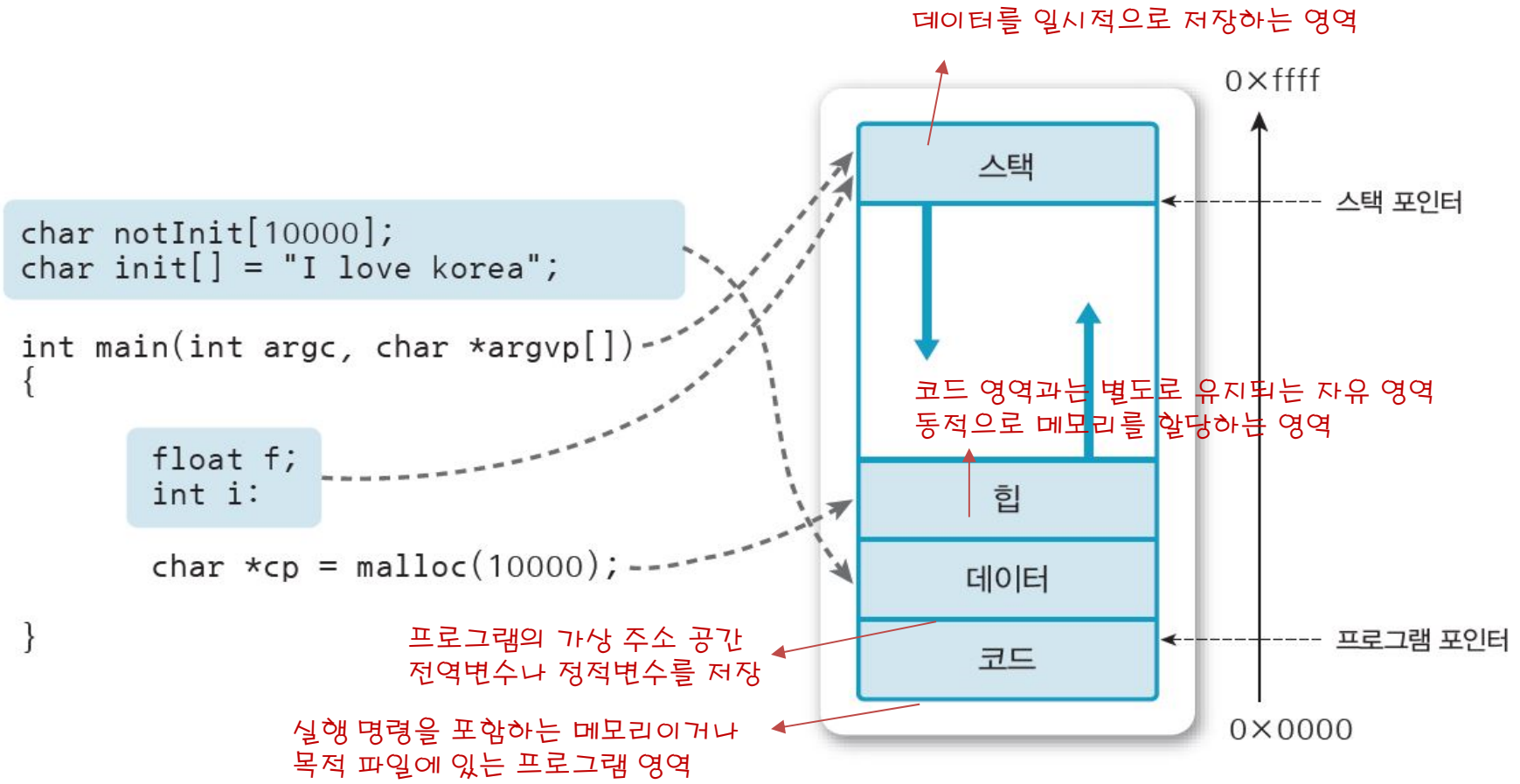


그림 3-2 프로세스의 일반적인 메모리 구조(사용자 관점의 프로세스)

1. 프로세스의 개념

■ 시스템 관점에서의 프로세스

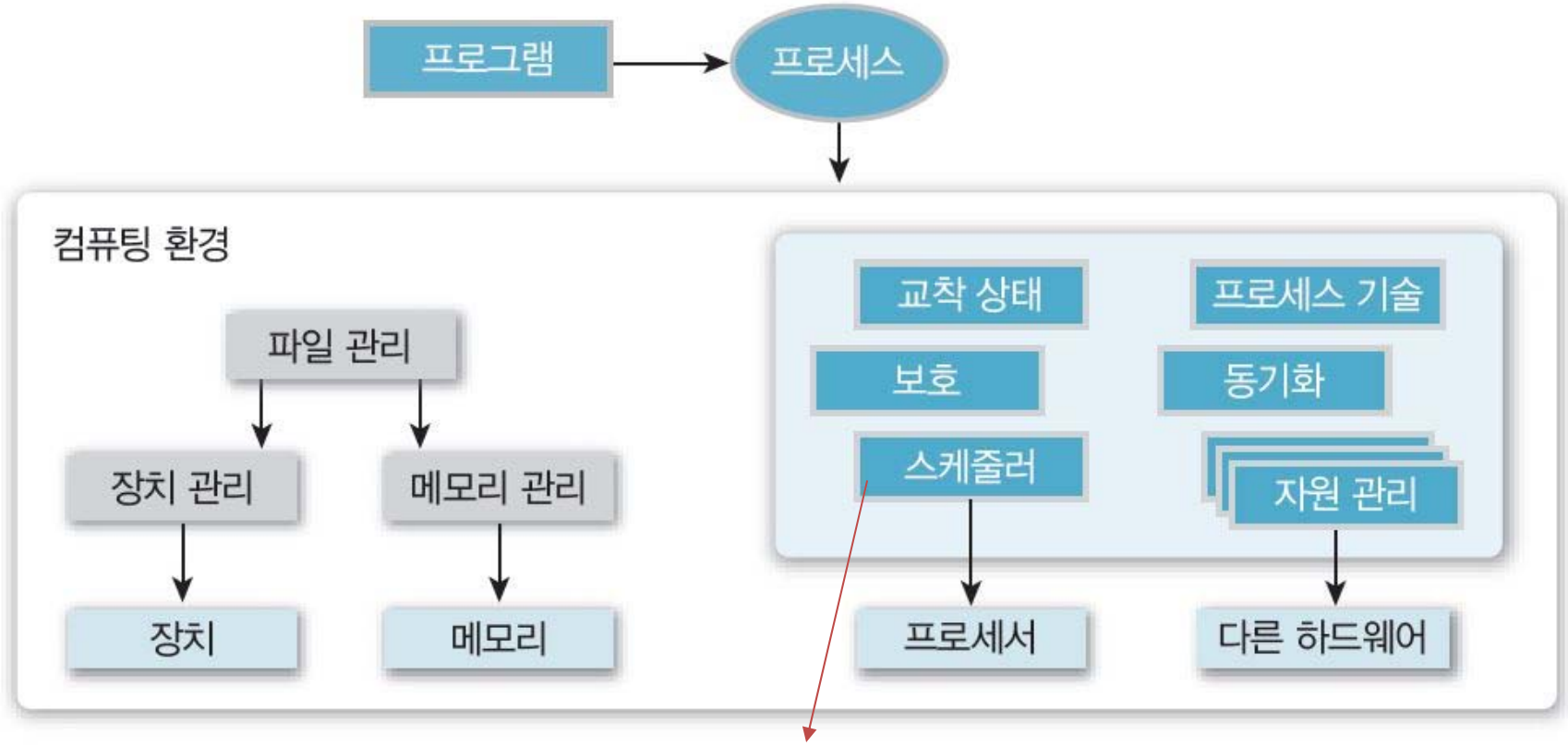


그림 3-3 시스템 관점에서 바라본 프로세스

· 실행 순서 결정
· 디스크에 저장된 프로그램에 프로세서 할당해 장치나 메모리 같은 파일 자원 참조
· 프로세스 지원, 협력하여 교착 상태, 보호, 동기화 등 정보 교 환

1. 프로세스의 개념

■ 프로세스의 종류

표 3-1 프로세스의 종류

구분	종류	설명
역할	시스템(커널) 프로세스	모든 시스템 메모리와 프로세서의 명령에 액세스할 수 있는 프로세스이다. 프로세스 실행 순서를 제어하거나 다른 사용자 및 커널(운영체제) 영역을 침범하지 못하게 감시하고, 사용자 프로세스를 생성하는 기능을 한다.
	사용자 프로세스	사용자 코드를 수행하는 프로세스이다.
병행 수행 방법	독립 프로세스	다른 프로세스에 영향을 주지 않거나 다른 프로세스의 영향을 받지 않으면서 수행하는 병행 프로세스이다.
	협력 프로세스	다른 프로세스에 영향을 주거나 다른 프로세스에서 영향을 받는 병행 프로세스이다.

2. 프로세스의 상태 변화와 상태 정보

■ 프로세스의 상태 변화

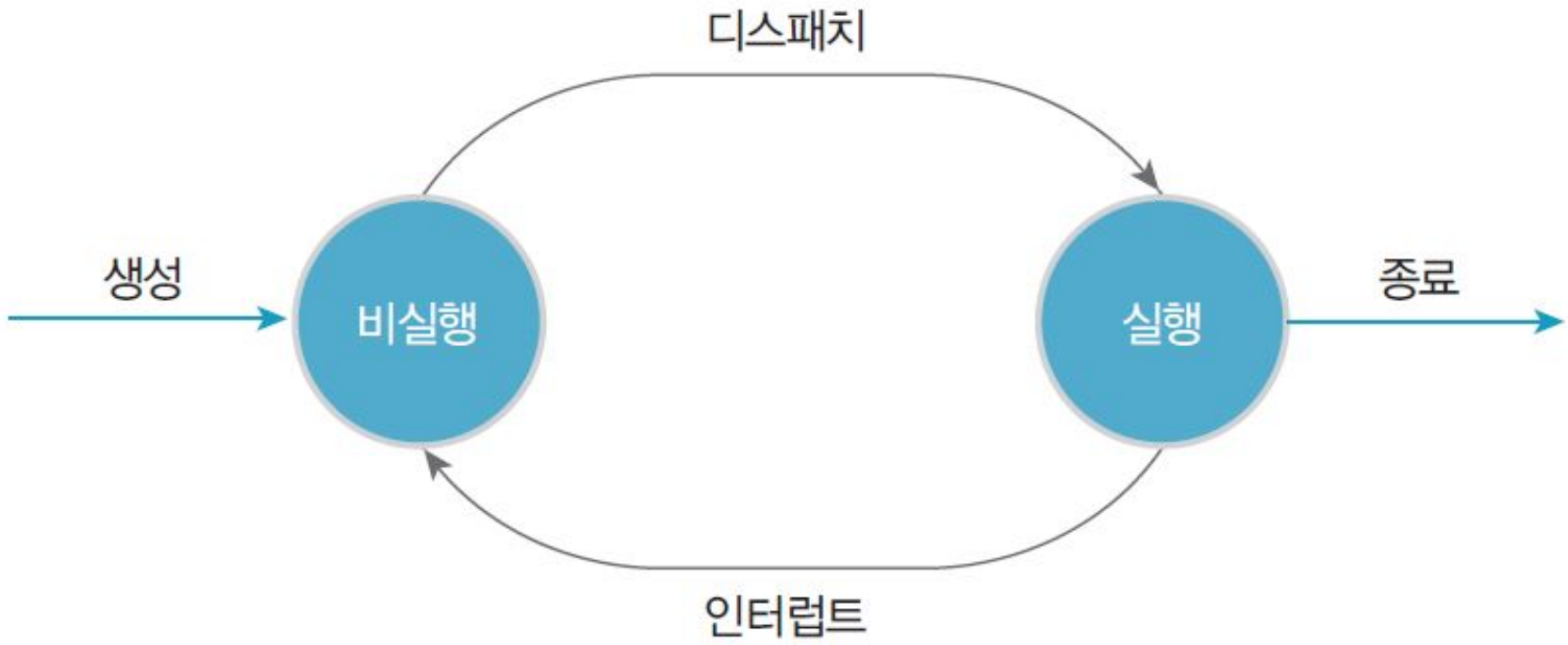


그림 3-4 프로세스의 상태

2. 프로세스의 상태 변화와 상태 정보

- 프로세스의 상태 변화는 운영체제가 프로세서 스케줄러 이용하여 관리
- 작업 스케줄러는 스풀러가 디스크에 저장한 작업 중 실행할 작업 선정하고 준비 리스트에 삽입하여 다중 프로그래밍의 정도 결정
- 프로세스 스케줄러는 선정한 작업의 상태를 변화시키며 프로세스의 생성에서 종료까지 과정 수행

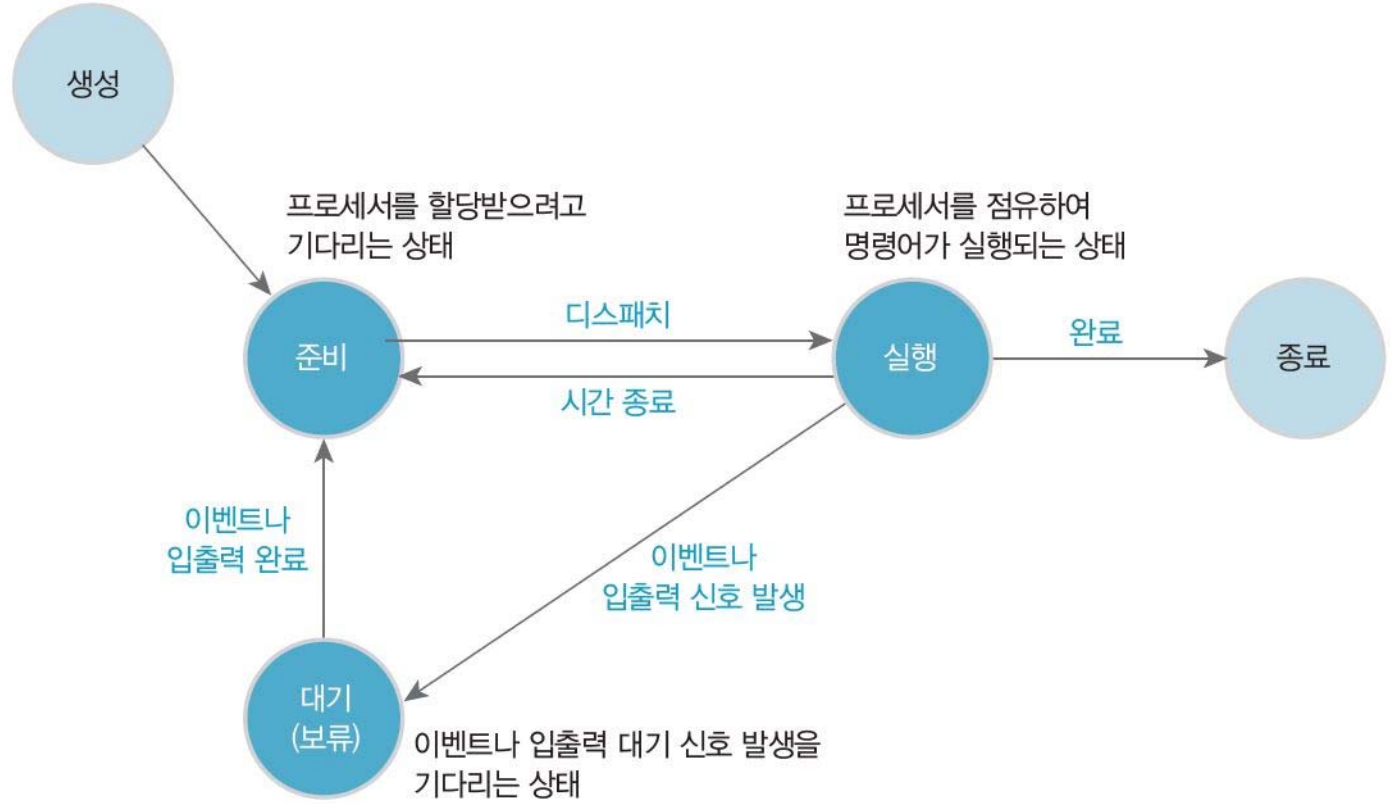


그림 3-5 프로세스의 상태 변화

2. 프로세스의 상태 변화와 상태 정보

■ 프로세스 제어 블록PCB, Process Control Block

- 운영체제가 프로세스 제어 시 필요한 프로세스 상태 정보 저장
- 특정 프로세스 정보 저장하는 데이터 블록이나 레코드(작업 제어 블록TCB, Task Control Block)
- 프로세스가 생성되면 메모리에 프로세스 제어 블록 생성, 프로세스가 실행 종료하면 해당 프로세스 제어 블록도 삭제

- 프로세스와 프로그램의 관계
  - 프로그램이 프로세스가 된다는 것은 운영체제로부터 프로세스 제어 블록을 얻는다는 뜻
  - 프로세스가 종료된다는 것은 해당 프로세스 제어 블록이 폐기된다는 뜻

정의 3-1 프로세스와 프로그램의 관계

프로세스 = 프로그램 + 프로세스 제어 블록

프로그램 = 프로세스 − 프로세스 제어 블록

2. 프로세스의 상태 변화와 상태 정보

프로세스 식별자	각 프로세스의 고유 식별자(숫자, 색인 항목)
프로세스 상태	생성, 준비, 실행, 대기, 중단 등 상태 표시
프로그램 카운터	프로세스를 실행하는 다음 명령의 주소 표시
레지스터 저장 영역	누산기, 인덱스 레지스터, 스택 포인터, 범용 레지스터, 조건 코드 등 정보로, 컴퓨터 구조에 따라 수나 형태가 다르다. 인터럽트가 발생하면 프로그램 카운터와 함께 저장하여 재실행할 때 원래대로 복귀할 수 있게 한다.
프로세서 스케줄링 정보	프로세스의 우선순위, 스케줄링 큐의 포인터, 기타 스케줄 매개변수
계정 정보	프로세서 사용 시간, 실제 사용 시간, 사용 상한 시간, 계정 번호, 작업이나 프로세스 번호 등
입출력 상태 정보 메모리 관리 정보 ⋮	• 특별한 입출력 요구 프로세스에 할당된 입출력장치, 열린 파일 리스트 등 • 운영체제가 사용하는 메모리 시스템에 따른 상한 · 하한 레지스터 (경계 레지스터), 페이지 테이블이나 세그먼트 테이블 값 등

그림 3-7 **프로세스 제어 블록**

2. 프로세스의 상태 변화와 상태 정보

■ 프로세스 테이블(process table)

- 운영체제는 각 프로세스의 PCB를 가리키는 포인터를 시스템 전체 혹은 사용자별 프로세스 테이블에 유지
- PCB에 빠르게 접근 할 수 있도록 함
- 프로세스가 종료하면, 운영체제는 프로세스의 메모리와 기타 자원을 해제해 다른 프로세스가 사용할 수 있게 하고, 프로세스 테이블에서 해당 프로세스를 제거

[그림 3-2] 프로세스 테이블과 프로세스 제어 블록

2. 프로세스의 상태 변화와 상태 정보

프로세스의 네 가지 상태

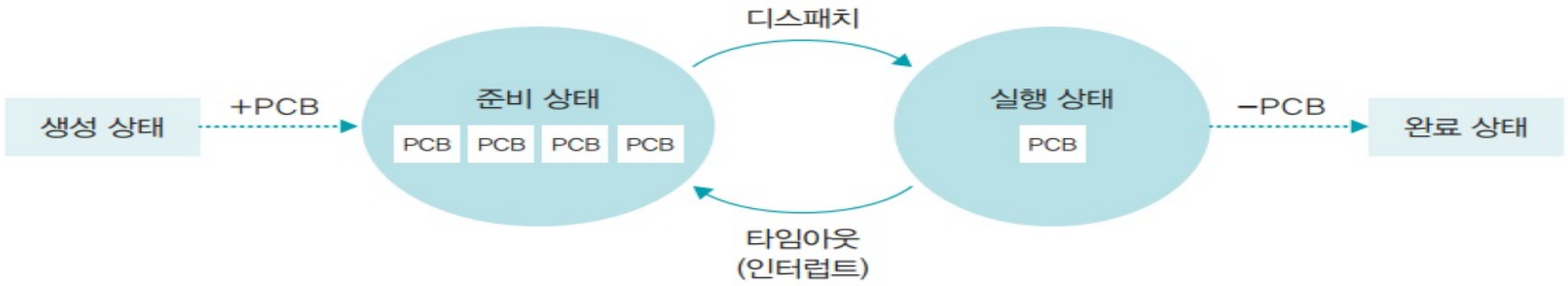


그림 3-8 간단한 프로세스의 상태

- **생성 상태** : 프로세스가 메모리에 올라와 실행 준비를 완료한 상태
- **준비 상태** : 생성된 프로세스가 CPU를 얻을 때까지 기다리는 상태
- **실행 상태** : 준비 상태에 있는 프로세스 중 하나가 CPU를 얻어 실제 작업을 수행하는 상태
- **완료 상태** : 실행 상태의 프로세스가 주어진 시간 동안 작업을 마치면 진입하는 상태(프로세스 제어 블록이 사라진 상태)
- **디스패치** : 준비 상태의 프로세스 중 하나를 골라 실행 상태로 바꾸는 CPU 스케줄러의 작업
- **타임아웃** : 프로세스가 자신에게 주어진 하나의 타임 슬라이스 동안 작업을 끝내지 못하면 다시 준비 상태로 돌아가는 것

2. 프로세스의 상태 변화와 상태 정보

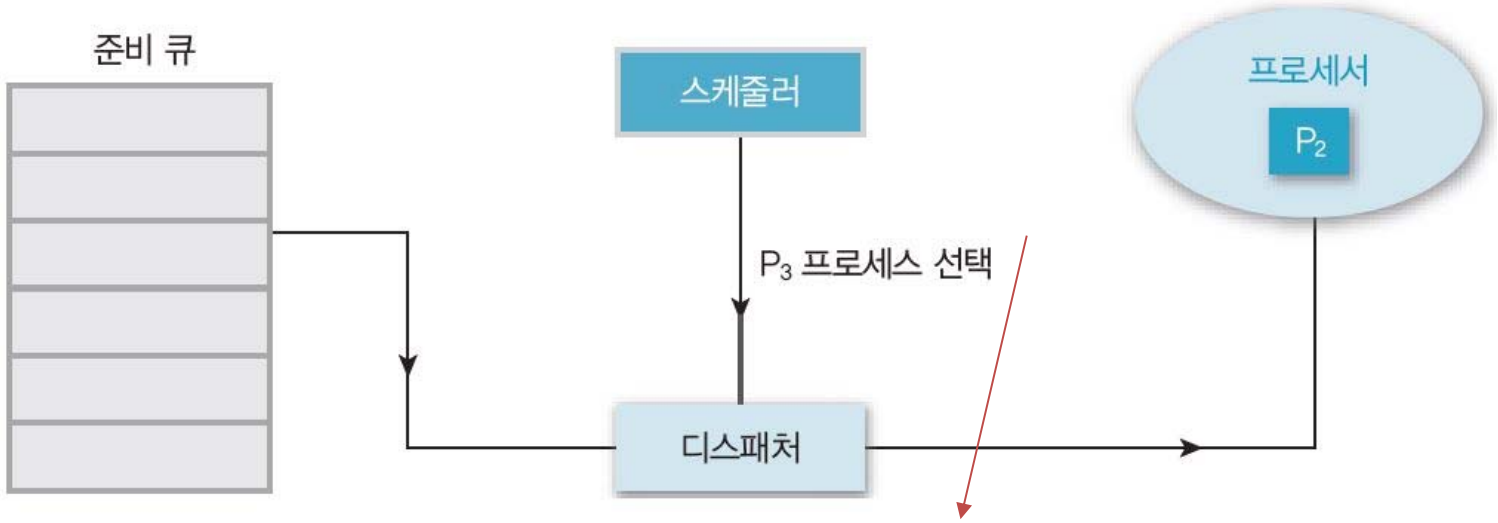


그림 3-6 프로세스와 디스패처 : 디스패처가 P_3 프로세스에 프로세서를 할당하는 예

표 3-2 프로세스의 상태 변화

프로세스 스스로 하는 것은 대기뿐, 나 머지는 외부 조건으로 발생

상태 변화	표기 방법
❶ 준비 → 실행	dispatch(프로세스 이름)
❷ 실행 → 준비	timeout(프로세스 이름)
❸ 실행 → 대기(보류)	block(프로세스 이름)
❹ 대기(보류) → 준비	wakeup(프로세스 이름)

큐 맨 앞에 있던 프로세스가 프로세서를 점유하는 것

2. 프로세스의 상태 변화와 상태 정보

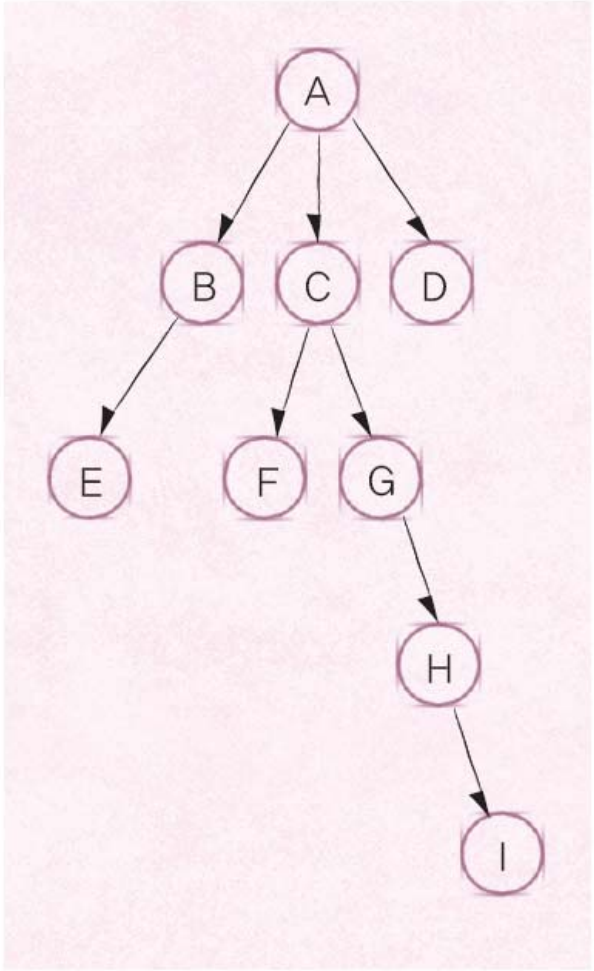
프로세스 연산

- 프로세스 생성
- 프로세스 소멸
- 프로세스 일시정지
- 프로세스 재시작
- 프로세스의 우선순위 변경
- 프로세스 블록킹
- 프로세스 깨우기
- 프로세스 디스패치
- 프로세스가 다른 프로세스와 통신(프로세스간 통신)

2. 프로세스의 상태 변화와 상태 정보

■ 프로세스 연산

- 프로세스는 새로운 프로세스를 생성(spawn)
  - 부모 프로세스(parent process)
    - 프로세스를 생성하는 프로세스
  - 자식 프로세스(child process)
    - 생성된 프로세스
  - 각 자식 프로세서는 정확히 한 부모 프로세스를 통해 생성
  - 프로세스가 소멸 될 경우 (2가지 경우 중 1가지)
    - 부모 프로세스가 소멸될 때 자동으로 자식 프로세스들도 소멸
    - 자식 프로세스가 부모에 독립적으로 계속 실행되어, 부모 프로세스의 소멸이 자식에 영향을 미치지 않음



[그림 3-3] 프로세스 생성 계층

Section 02 프로세스의 관리(1. 프로세스의 구조)

■ 프로세스의 구조

- 프로세스 실행 중 프로세스 생성 시스템 호출 이용 새로운 프로세스 생성
- 프로세스 생성 순서를 저장, 부모-자식 관계 유지하여 계층적 생성
- 생성하는 프로세스는 부모 프로세스parent process, 생성되는 프로세스는 자식 프로세스child process 또는 서브 프로세스subprocess
- 부모 프로세스는 자식 프로세스를 생성 과정 반복하면서 계층 구조 형성

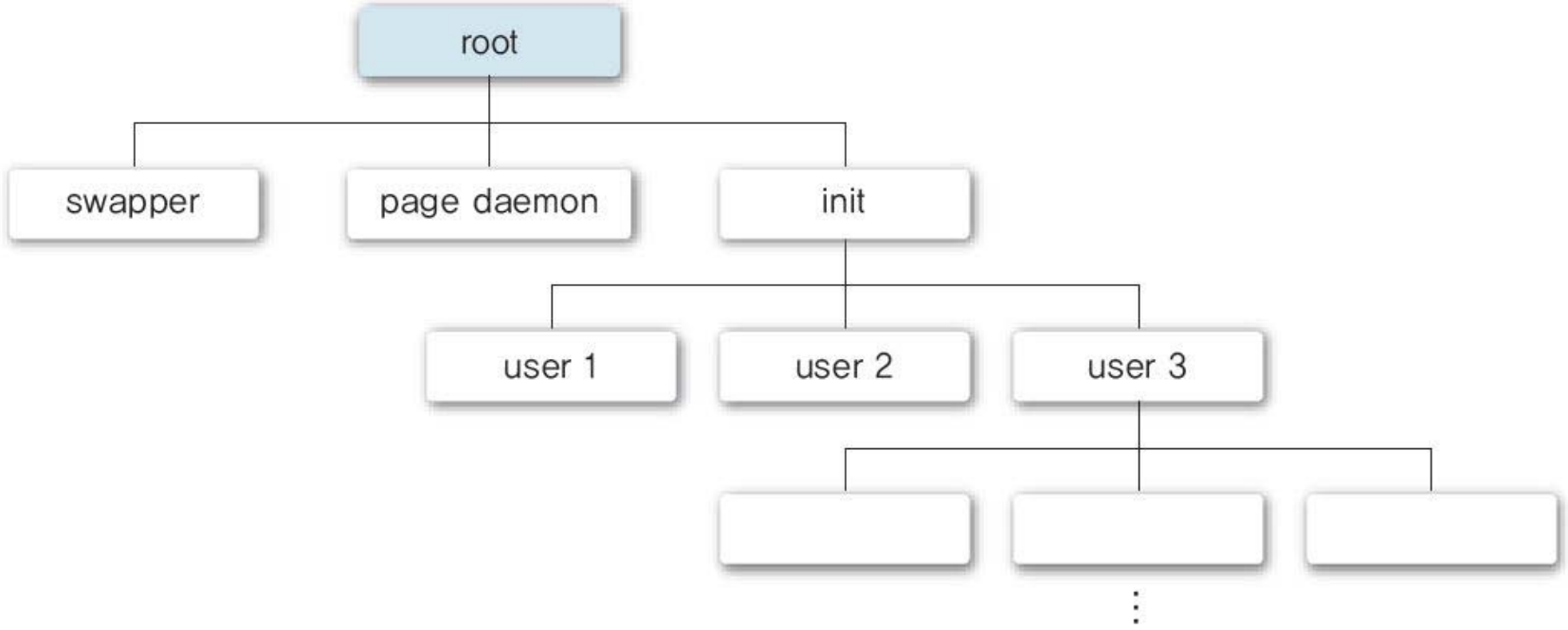


그림 3-9 유닉스 시스템의 프로세스 계층 구조 예

2. 프로세스의 생성

프로세스의 생성 시기

- 일괄처리 환경 : 작업이 도착할 때 프로세스 생성
- 대화형 환경 : 새로운 사용자가 로그온 log-on할 때 프로세스 생성

프로세스 생성시 필요한 세부 작업 순서

❶ 새로운 프로세스에 프로세스 식별자 할당

❷ 프로세스의 모든 구성 요소를 포함할 수 있는 주소 공간과 프로세스 제어 블록 공간 할당

❸ 프로세스 제어 블록 초기화(프로세스 상태, 프로그램 카운터 등 초기화, 자원 요청, 프로세스 제어 정보(우선순위) 등을 포함)

❹ 링크(해당 큐에 삽입)

프로세스가 새로운 프로세스 생성 시 다음 두 가지 실행 발생

- 부모 프로세스와 자식 프로세스 동시 실행
- 부모 프로세스는 자식 프로세스 모두 종료할 때까지 대기

3. 프로세스의 종료

프로세스의 종료

- 프로세스가 마지막 명령 실행, 종료하여 운영체제에 프로세스의 삭제 요청
- 일괄 처리 환경 : 작업 종료 의미의 신호로 인터럽트 발생 또는 시스템 호출로 중단 명령
- 대화형 환경 : 사용자가 로그오프log-off하거나 터미널 닫음
- abort 명령어로 프로세스 종료
- 부모 프로세스의 자식 프로세스 종료
  - 보통 부모 프로세스 종료하면 운영체제가 자식 프로세스도 종료(연속 종료)
  - 자식 프로세스가 할당된 자원을 초과하여 자원을 사용할 때
  - 자식 프로세스에 할당한 작업이 더는 없을 때
- exit 명령어 : 유닉스에서 프로세스 종료
- wait 명령어 : 부모 프로세스가 자식 프로세스의 종료 기다림
- 프로세스 종료 이유
  - 정상 종료 : 프로세스가 운영체제의 서비스 호출
  - 시간 초과
  - 실패 : 파일 검색 실패, 입출력이 명시된 횟수 초과하여 실패할 때
  - 산술 오류, 보호 오류, 데이터 오류 등, 메모리 부족, 액세스 위반 등

4. 프로세스의 제거

프로세스 제거

- 프로세스 파괴
- 사용하던 자원 시스템에 돌려주고, 해당 프로세스는 시스템 리스트나 테이블에서 사라져 프로세스 제어 블록 회수
- 프로그램은 여전히 디스크에 저장
- 자식 프로세스는 부모 프로세스를 제거하면 자동 제거

5. 프로세스의 중단과 재시작

■ 프로세스의 중단

- 시스템의 유휴시간 문제를 프로세스 중단(일시정지) 상태를 이용 해결
- 운영체제는 새로운 프로세스를 생성하여 실행하거나 실행 중인 프로세스를 중단했다가 다시 실행 하여 사용 가능. 후자의 방법 이용하면 시스템 전체의 부하를 증가시키지 않고 프로세스에 서비스 제공
- 실행에서 대기가 아닌 중단 상태 추가하면 특정 이벤트의 발생을 기다리면서 대기 상태가 되어 해당 이벤트가 발생할 때 즉시 실행 상태로 바꿀 수 있는 이점
- 다중 프로그래밍에서 중단
  - 프로세스 입출력 요구 외에 다른 원인으로 프로세스가 실행을 중단한 상태(자원 부족(대기) 상태)
- 단일 처리 시스템 : 해당 프로세스 스스로 중단
- 다중 처리 시스템 : 다른 프로세서가 실행 중인 프로세스 중단
- 중단된 프로세스는 다른 프로세서가 재시작하기 전에는 실행 불가
- 장시간 중단 시 해당 프로세스에 할당된 자원 반환, 자원의 성질에 따라 반환 자원 결정
  - 메인 메모리 : 프로세서 중단 즉시 반환
  - 보조 메모리 : 중단 시간 예측할 수 없거나 너무 길 때 반환
- 중단한 프로세스는 중단한 지점부터 다시 시작

5. 프로세스의 중단과 재시작

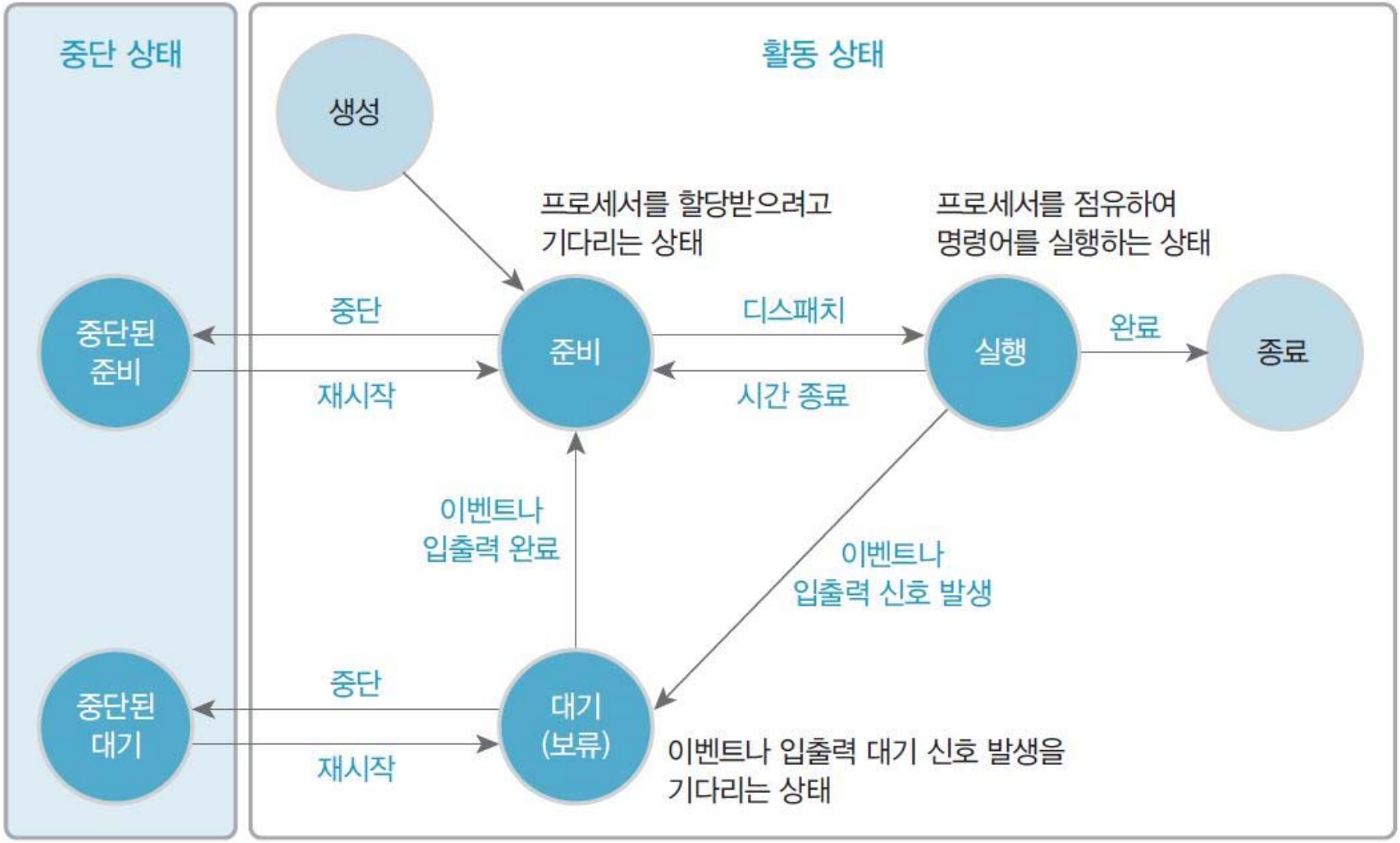


그림 3-10 중단과 재시작을 추가한 프로세스의 상태 변화

6. 프로세스의 우선 순위

프로세스 스케쥴러

- 프로세스 제어 블록의 우선순위 이용하여 준비 리스트의 프로세스 처리
- 준비 리스트의 프로세스는 프로세서 중심 프로세스와 입출력 중심 프로세스로 구분
- 입출력 중심 프로세스는 속도가 느리면서 빠른 응답 요구하는 단말기 입출력 프로세스에 높은 우선순위 부여, 속도가 빠른 디스크 입출력 프로세스에는 낮은 우선순위 부여
- 우선순위가 낮은 프로세스에는 시간을 많이, 우선순위가 높은 프로세스에는 적게 할당
- 입출력 중심 프로세스는 프로세서를 짧게 자주 사용하도록 하고, 프로세서 중심 프로세스 프로세서를 길게 사용하되 사용 횟수를 줄여 균형 유지

7. 프로세스의 문맥 교환

■ 프로세스의 문맥 교환

- 실행 중인 프로세스의 제어를 다른 프로세스에 넘겨 실행 상태가 되도록 하는 것
- 프로세스 문맥 교환이 일어나면 프로세서의 레지스터에 있던 내용 저장

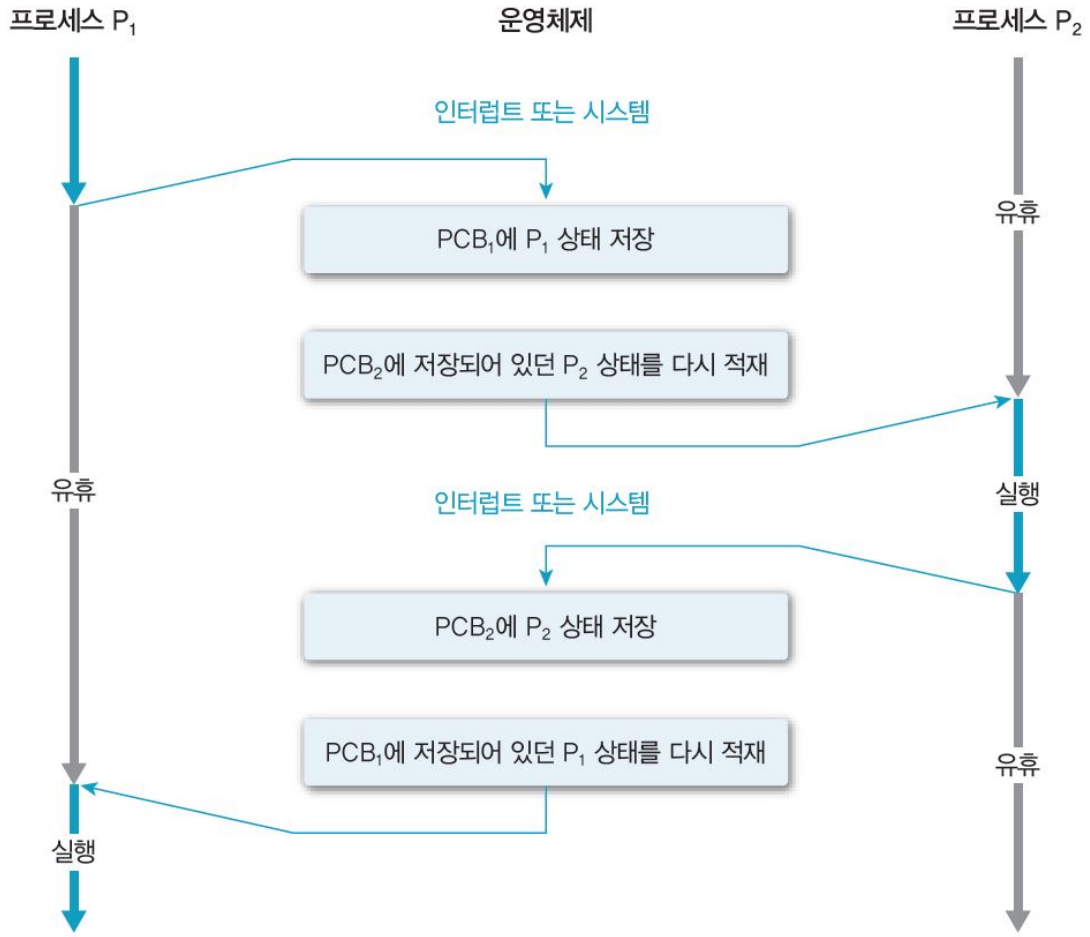


그림 3-8 프로세스 문맥 교환 예

7. 프로세스의 문맥 교환

프로세스의 문맥 교환 발생

- 실행 중인 프로세스에 인터럽트가 발생하면 운영체제가 다른 프로세스를 실행 상태로 바꾸고 제어를 넘겨주어 프로세스 문맥 교환 발생
- 인터럽트 발생 : 현재 실행하는 프로세스와 별도로 외부에서 이벤트(예 : 입출력 동작의 종료) 발생시
- 인터럽트 유형에 따른 루틴 분기
  - 입출력 인터럽트 : 입출력 동작이 발생 확인, 이벤트 기다리는 프로세스를 준비 상태로 바꾼 후 실행할 프로세스 결정
  - 클록 인터럽트 : 실행 중인 프로세스 할당 시간 조사, 준비 상태로 바꾸고, 다른 프로세스를 실행 상태로 전환
- 인터럽트는 인터럽트 처리 루틴을 실행한 후 현재 실행 중인 프로세스를 재실행할 수 있으므로 인터럽트가 곧 프로세스 문맥교환으로 발전하지는 않음

7. 프로세스의 문맥 교환

문맥 교환context switching

- 이전 프로세스의 상태 레지스터 내용 보관하고 다른 프로세스의 레지스터 적재하여 프로세스를 교환하는 일련의 과정
- 오버헤드 발생, 이는 메모리 속도, 레지스터 수, 특수 명령어의 유무에 따라 다름
- 오버헤드는 시간 비용 소요되어 운영체제 설계 시 불필요한 문맥 교환 감소가 주요 목표
- 레지스터 문맥 교환, 작업 문맥 교환, 스레드 문맥 교환, 프로세스 문맥 교환 가능

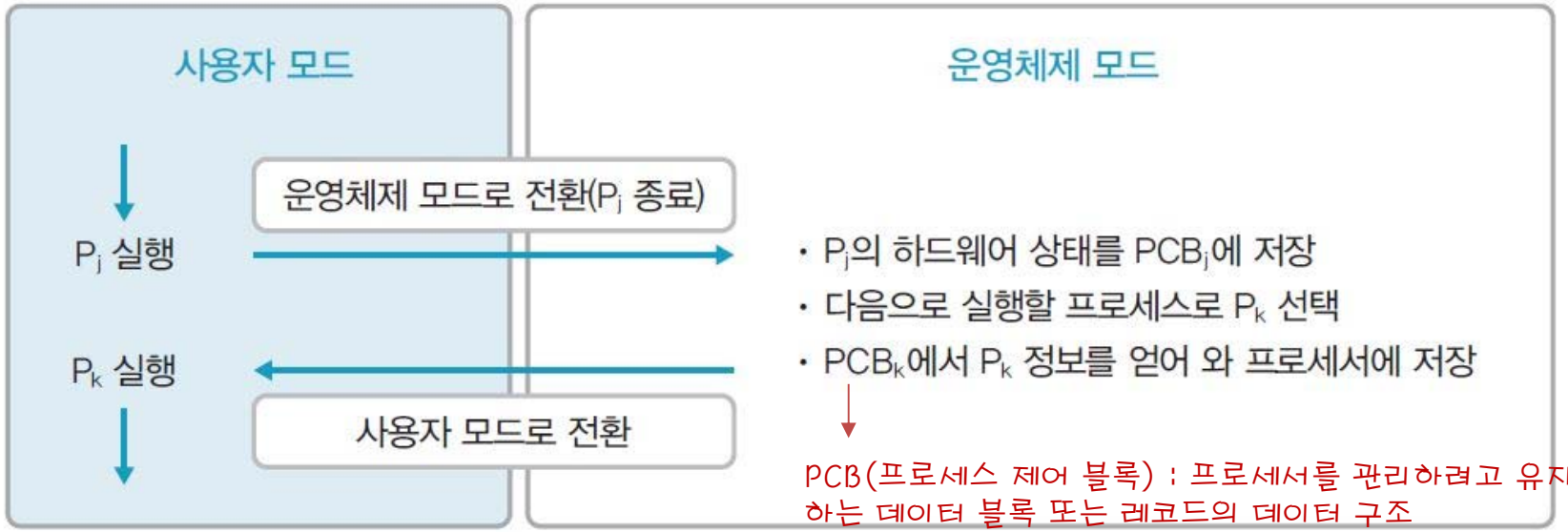


그림 3-11 문맥 교환 예

8. 프로세스간 통신

프로세스 간 통신의 종류

- 프로세스 내부 데이터 통신
  - 하나의 프로세스 내에 2개 이상의 스레드가 존재하는 경우의 통신
  - 프로세스 내부의 스레드는 전역 변수나 파일을 이용하여 데이터를 주고받음

- 프로세스 간 데이터 통신
  - 같은 컴퓨터에 있는 여러 프로세스끼리 통신하는 경우
  - 공용 파일 또는 운영체제가 제공하는 파이프를 사용하여 통신

- 네트워크를 이용한 데이터 통신
  - 여러 컴퓨터가 네트워크로 연결되어 있을 때 통신
  - 소켓을 이용하여 데이터를 주고받음

8. 프로세스간 통신

프로세스 간 통신의 종류

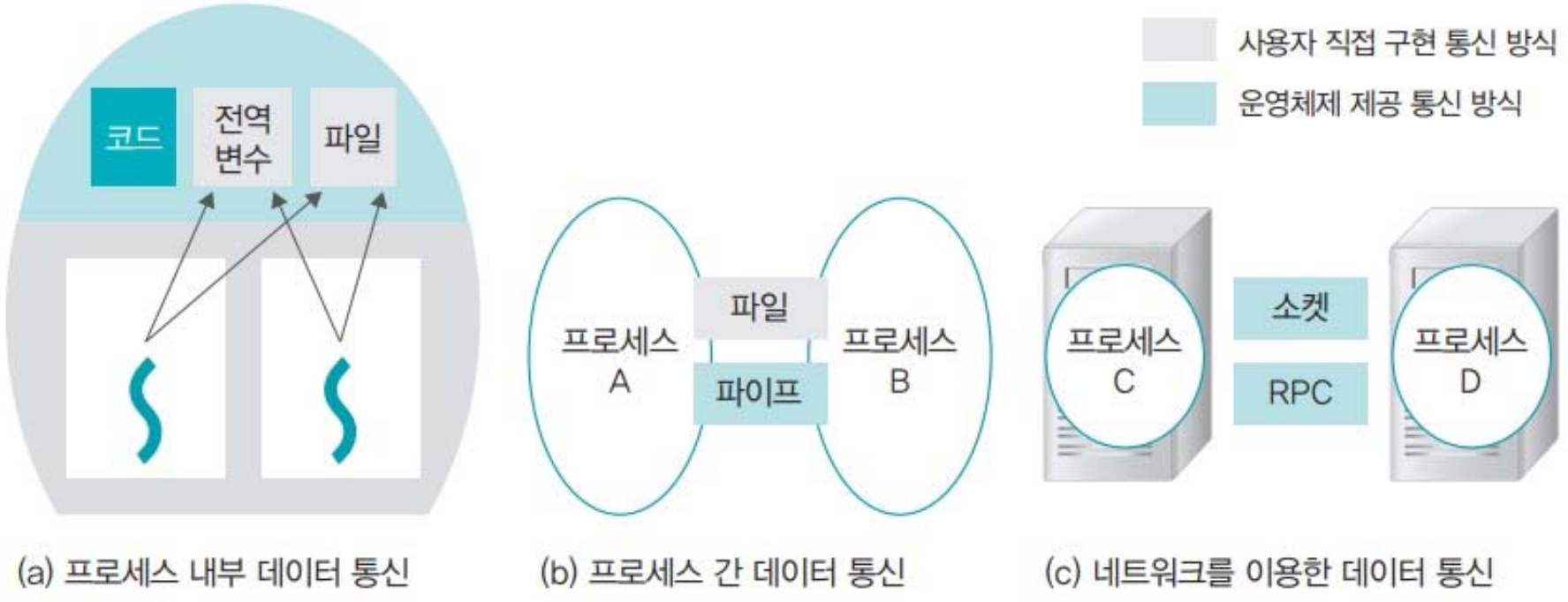


그림 5-1 **프로세스 간 통신의 종류**

8. 프로세스간 통신

- **많은 운영체제에서 프로세스 간 통신(IPC: InterProcess Communication) 메커니즘 제공**
  - 멀티프로그래밍 네트워크 환경
    - 예를 들어, 원격 서버에서 데이터 조회
  - 공통의 목적을 달성하기 위해 서로 협력

- **신호(signal)**
  - 프로세스에 이벤트가 발생했음을 알리는 소프트웨어 인터럽트
    - 프로세스들이 다른 프로세스와 교환할 데이터를 명시하지 않음
  - 잡음(catch), 무시(ignore), 마스킹(masking)
    - 프로세스는 신호를 전달할 때 운영체제가 호출하는 루틴을 정함으로써 신호를 잡음
    - 프로세스는 신호 처리를 위한 운영체제의 기본 동작에 의존함으로써 신호 무시
    - 프로세스가 특정 유형의 신호를 마스크하면, 해당 유형의 신호를 전달 하지 않음

8. 프로세스간 통신

메시지 전달(message passing)

- 메시지 기반 통신
  - 메시지는 한 번에 한 방향으로 전달
    - 송신자, 수신자
  - 메시지 전달이 양방향으로 일어날수도 있음
    - 전송자인 동시에 수신자
  - 블록킹 송신, 넌블록킹 송신
    - 블록킹 송신은 수신자가 메시지를 수신할 때까지 대기
    - 넌블록킹 송신은 송신자가 수신자로부터 메시지를 받지 않았을 때도 다른 작업을 계속 할 수 있는 전송 방식
  - 파이프(pipe)
    - 잘 알려진 메시지 전달 구현
    - 운영체제에서 보호하는 메모리 영역으로, 버퍼 역할을 해 프로세스 둘 이상이 데이터를 교환

8. 프로세스간 통신

메시지 전달(message passing)

- 분산 시스템에서 IPC
  - 전송에 오류가 생기거나 심지어 데이터를 잃어 버릴 수 있음
    - 승인 프로토콜 사용
      - » 송신자와 수신자가 각각의 데이터 전송이 적절히 이루어지는지 확인할 수 있게 사용
    - 타이아웃 메커니즘
      - » 승인 메시지를 기다리는 송신자가 타임아웃 동안 승인이 도착되지 않으면 메시지 재전송 요구
  - 명확하지 않은 프로세스 이름은 메시지 전달을 어렵게 함
    - 번호가 부여된 포트들을 통해 컴퓨터를 사이에서 메시지 전송
  - 보안 문제
    - 인증 문제

9. 스레드의 개념과 상태 변화(스레드의 개념)

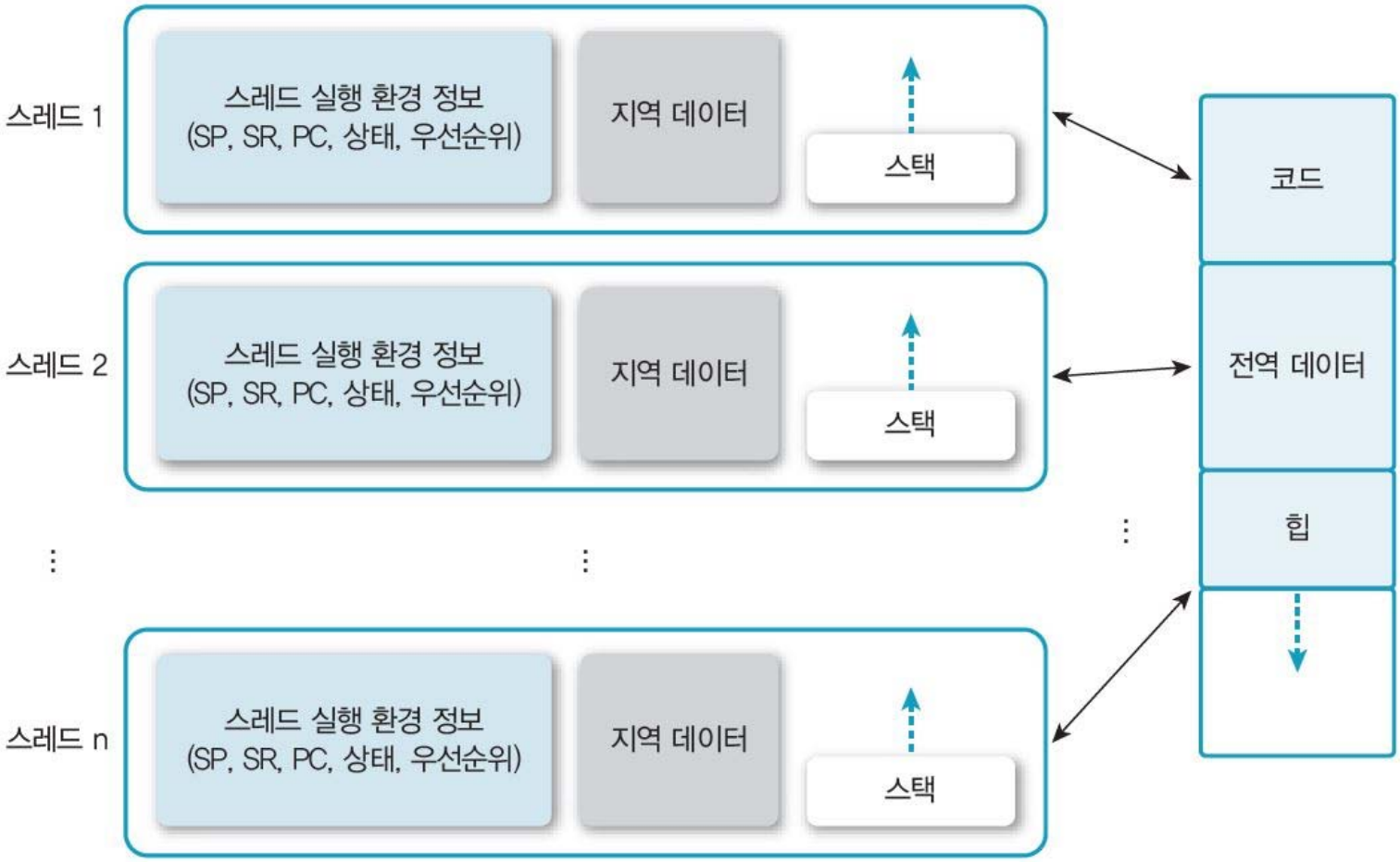
■ 스레드thread의 개념

- 프로세스의 특성인 자원과 제어에서 제어만 분리한 실행 단위
- 프로세스 하나는 스레드 한 개 이상으로 나눌 수 있음
- 프로세스의 직접 실행 정보를 제외한 나머지 프로세스 관리 정보 공유
- 다른 프로시저 호출, 다른 실행 기록(별도 스택 필요)
- 관련 자원과 함께 메모리 공유 가능하므로 손상된 데이터나 스레드의 이상 동작 고려
- 경량 프로세스LWP, Light Weight Process : 프로세스의 속성 중 일부가 들어 있는 것(스레드)
- 중량 프로세스HWP, Heavy Weight Process : 스레드 하나에 프로세스 하나인 전통적인 경우
- 같은 프로세스의 스레드들은 동일한 주소 공간 공유
- CPU 스케줄러가 CPU에 전달하는 일 하나
- CPU가 처리하는 작업의 단위는 프로세스로부터 전달받은 스레드
  - 운영체제 입장에서의 작업 단위는 프로세스
  - CPU 입장에서의 작업 단위는 스레드

정의 3-4 스레드

프로세스의 코드에 정의된 절차에 따라 CPU에 작업 요청을 하는 실행 단위이다.

9. 스레드의 개념과 상태 변화(스레드의 개념)



- SP[Stack Pointer] : 스택 포인터
- SR[Sequence Register] : 순서열 레지스터
- PC[Program Counter] : 프로그램 카운터

그림 3-12 스레드의 구조

9. 스레드의 개념과 상태 변화(스레드의 개념)

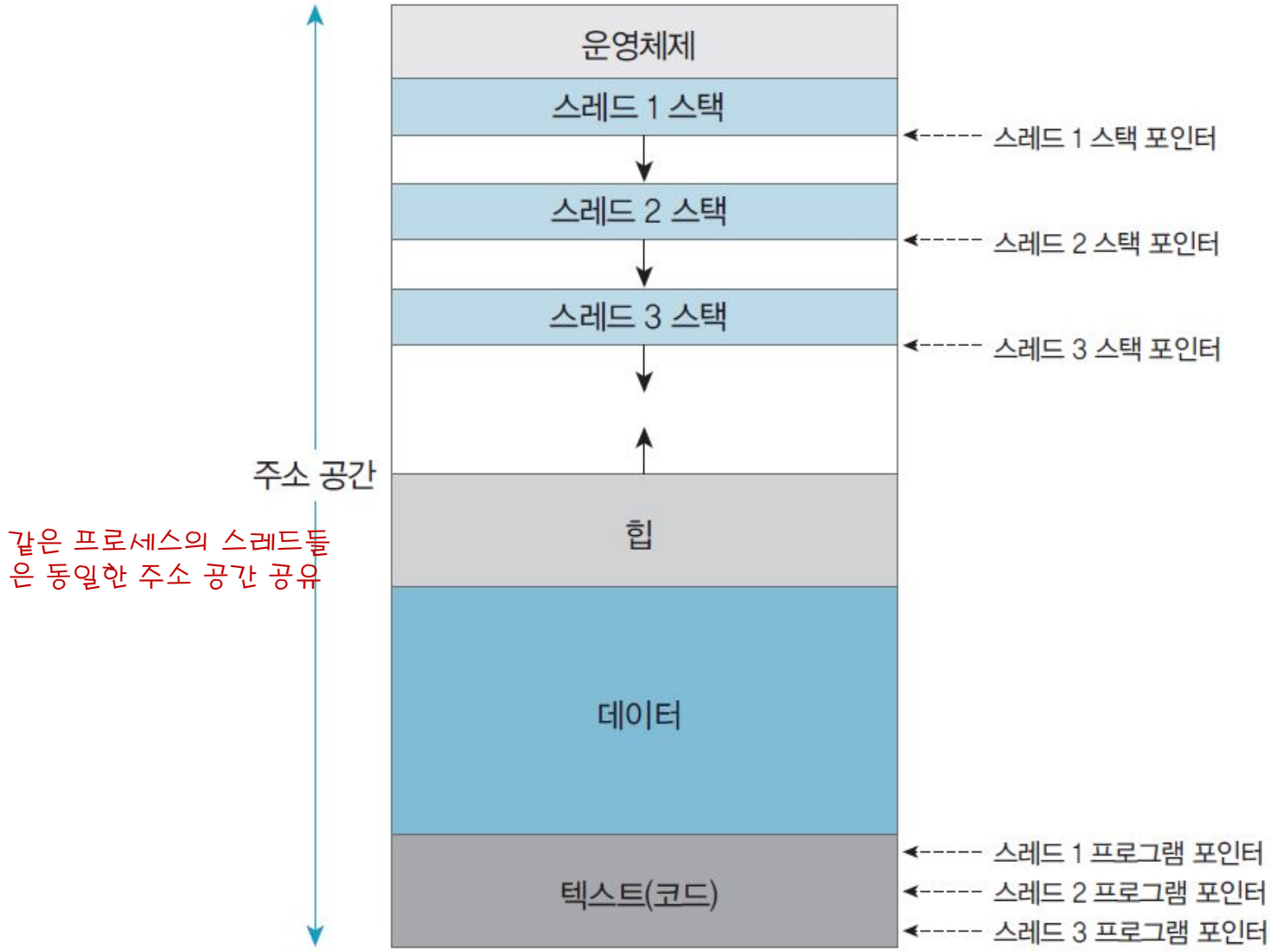


그림 3-13 스레드의 주소 공간

9. 스레드의 개념과 상태 변화(스레드의 개념)

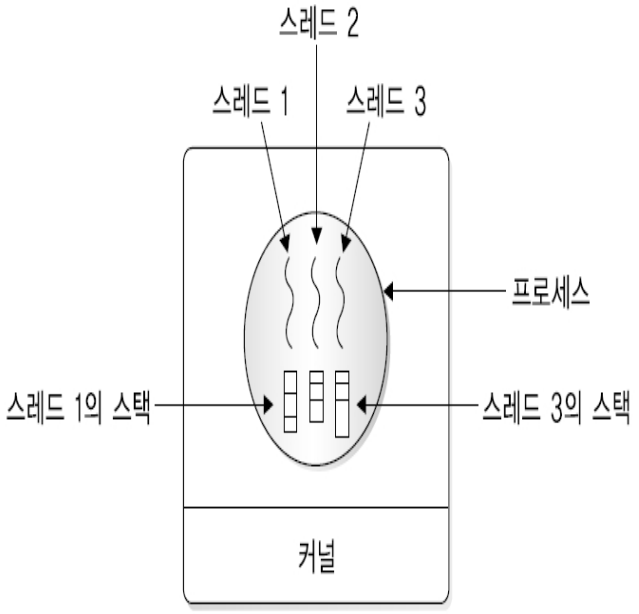
- 스레드의 장점
  - 프로세스에 포함된 스레드들은 공통의 목적 달성을 위해 병렬로 수행
    - 자원을 공유하여 한 프로세스에서 동시 작업 가능
  - 시스템 성능과 효율 향상
    - 하나의 프로세스가 서로 다른 프로세서에서 프로그램의 다른 부분을 동시에 실행 가능
    - 응용 프로그램 하나가 비슷한 작업들을 여러 개 수행
  - 스레드를 이용하여 다음과 같은 이점을 얻을 수 있음

- 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



프로세스와 다중 스레드

9. 스레드의 개념과 상태 변화(스레드의 개념)

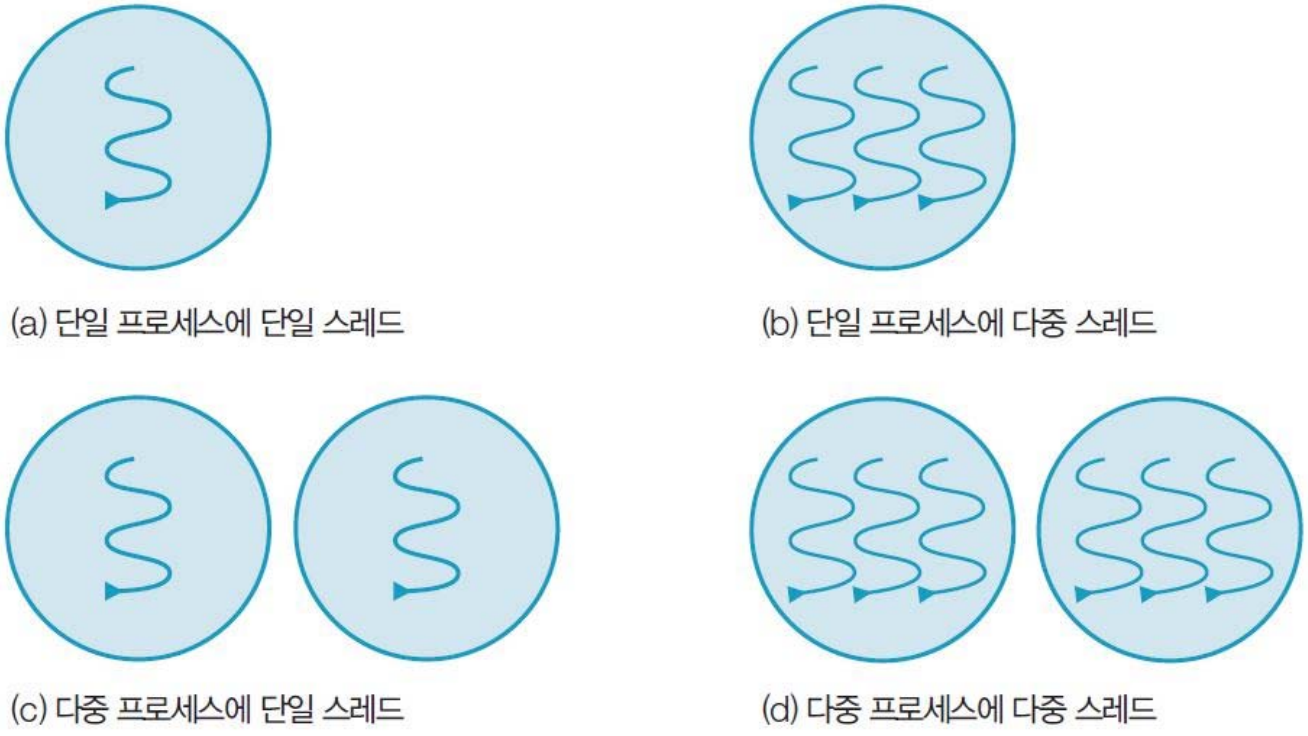
- **멀티 스레딩을 선호하게 된 요인**
  - 소프트웨어 설계
    - 병렬 작업 처리를 더 간단하게 표현 가능
  - 성능
    - 멀티 스레드 응용 프로그램이 작업 완료을 완료하는데 필요한 시간이 크게 감소
  - 협력
    - 한 프로세스에 속한 스레드들은 주소 공간을 공유하기 때문에 이를 이용해 서로 통신 가능
    - 이전에는 커널을 통해 프로세스간 통신 채널을 구축하여 중량 프로세스로 수행

- **스레드와 프로세스는 공통된 많은 연산들을 소유**
  - 예제: create, exit, resume, suspend

- **스레드의 생성은 운영체제가 부모 프로세스와 스레드 간에 공유되는 자원을 초기화 할 필요가 없음**
  - 프로세스의 생성과 종료에 비교하여 스레드의 생성과 종료에 대한 오버헤드를 감소시키기 위함

9-1. 단일 스레드와 다중(멀티) 스레드

■ 스레드와 프로세스의 관계



(a) 단일 프로세스에 단일 스레드

(b) 단일 프로세스에 다중 스레드

(c) 다중 프로세스에 단일 스레드

(d) 다중 프로세스에 다중 스레드

그림 3-14 단일 스레드 프로세스와 다중 스레드 프로세스

- 단일 스레드 : 스레드 용어가 탄생하기 전이라 개념 불확실(도스)
- 다중 스레드 : 프로그램 하나를 여러 실행 단위로 쪼개어 실행한다는 측면에서 다중 처리 (다중 프로세싱)와 의미 비슷

9-1. 단일 스레드와 다중(멀티) 스레드

■ 프로세스 관리 측면에서 단일 스레드와 다중 스레드

(a) 단일 스레드의 프로세스 (b) 다중 스레드의 프로세스

그림 3-15 프로세스 관리 측면에서 살펴본 단일 스레드 프로세스와 다중 스레드 프로세스

스레드별로 실행 환경 정보가 따로 있지만 서로 많이 공유하므로, 프로세스보다 동일한 프로세스의 스레드에 프로세서를 할당하거나 스레드 간의 문맥 교환이 훨씬 경제적

9-3. 스레드의 사용 예

- 스레드를 이용하여 프로그램의 비동기적 요소를 구현한 예

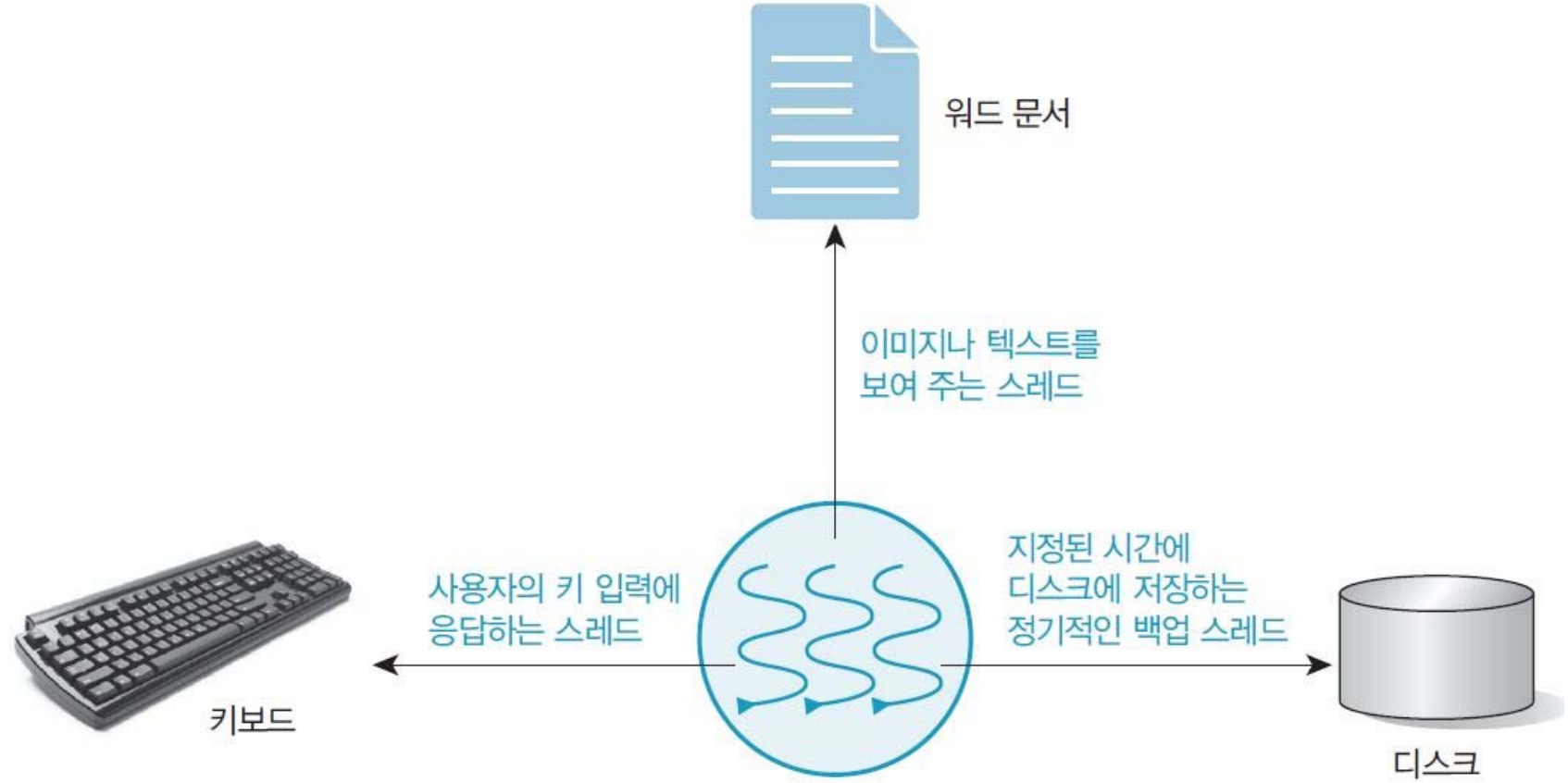


그림 3-16 다중 스레드를 이용한 워드 편집기 프로세스

9-3. 스레드의 사용 예

■ 다중 스레드 개념 적용 예

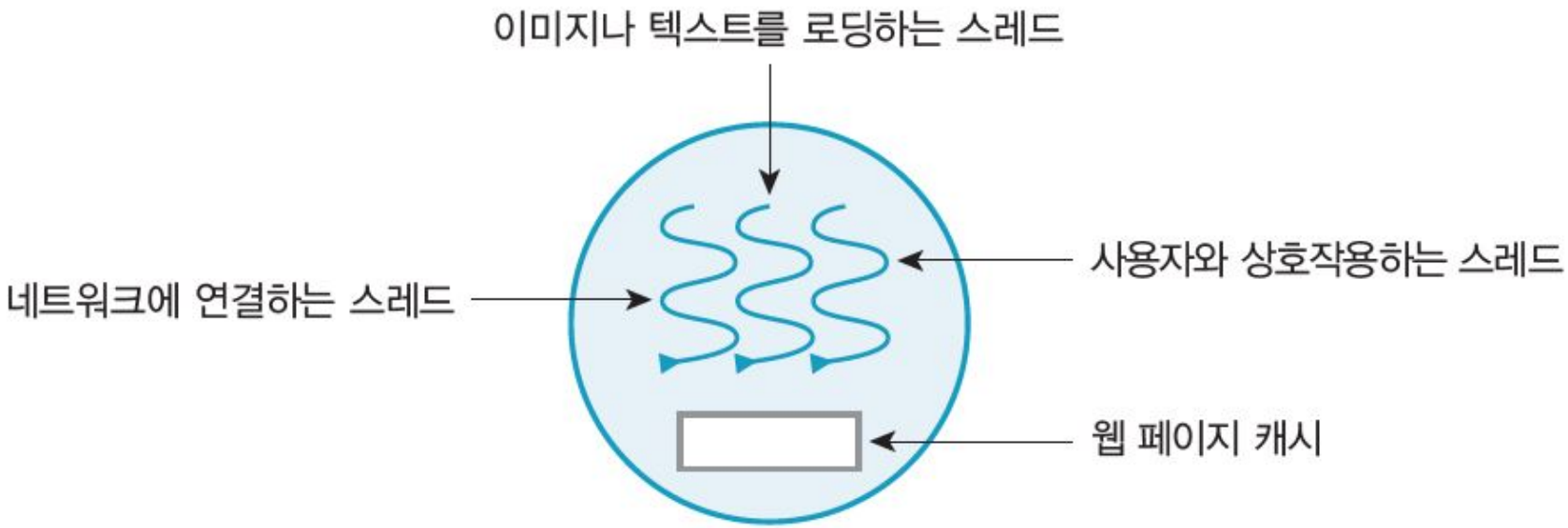


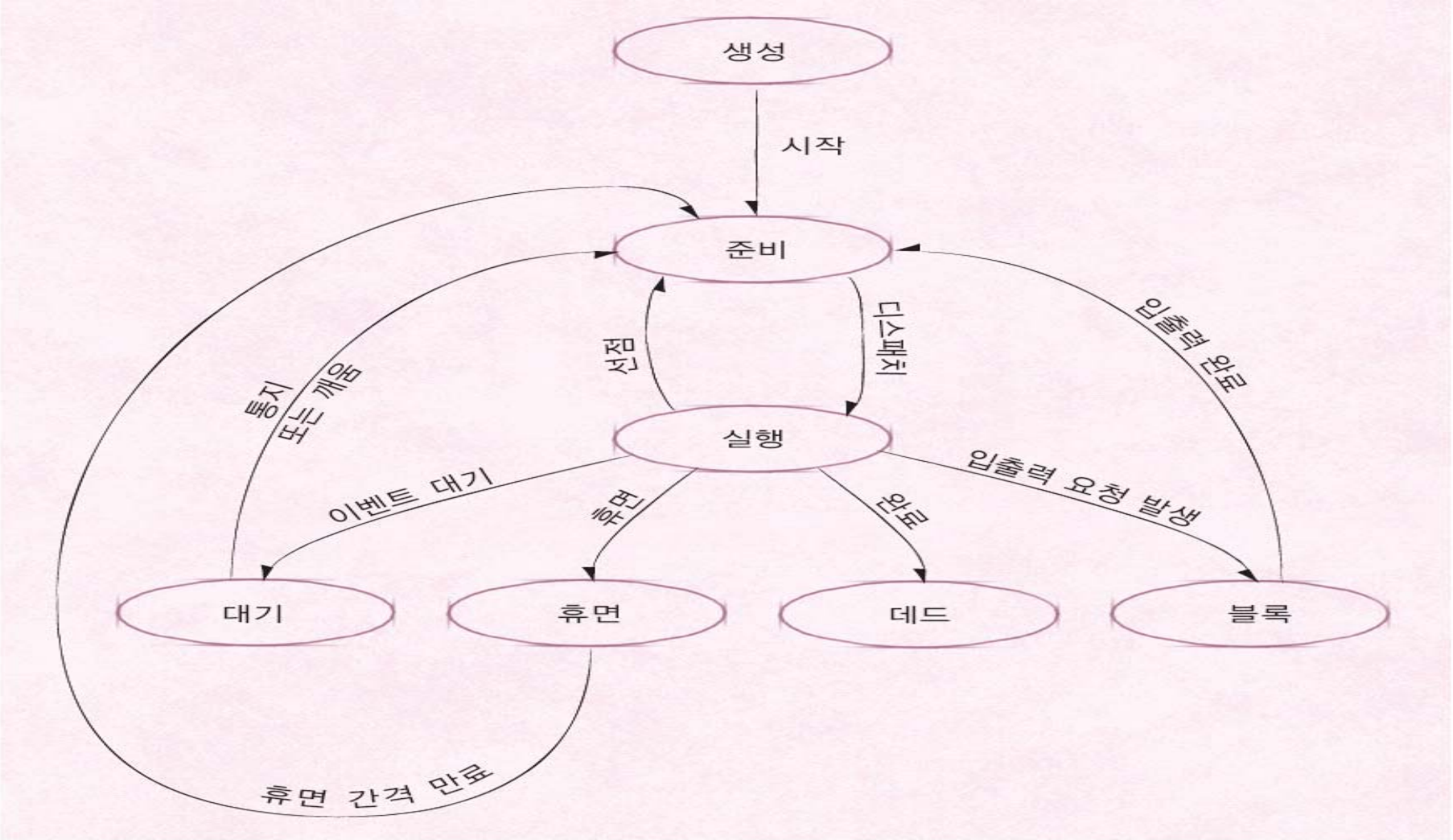
그림 3-17 다중 스레드를 이용한 웹 브라우저 프로세스

9-4. 스레드의 상태 변화

스레드의 상태 변화

- 프로세서 함께 사용, 항상 하나만 실행
- 한 프로세스에 있는 스레드는 순차적 실행, 해당 스레드의 정보를 저장하는 레지스터와 스택이 있음
- 프로세스 생성하면 해당 프로세스의 스레드도 함께 생성. 단, 스레드 생성에서는 운영체제가 부모 프로세스와 공유할 자원 초기화 필요 없음
- 프로세스의 생성과 종료보다는 오버헤드 훨씬 적음
- 스레드 한 개가 대기 상태로 변환 시 전체 프로세스 대기 상태로 변환 않음
- 실행 상태의 스레드가 대기 상태가 되면 다른 스레드 실행 가능(서로 독립적이지는 않음)
- 프로세스 하나에 있는 전체 스레드는 프로세스의 모든 주소에 접근 가능하여 스레드 한 개가 다른 스레드의 스택 읽기나 덮어 쓰기 가능
- 프로세서는 여러 사용자가 생성하여 서로 경쟁적으로 자원 요구하고 서로 다른 관계를 유지해야 하지만, 스레드는 사용자 한 명이 여러 스레드로 개인 프로세스 하나 소유

4. 스레드의 상태 변화



[그림 4-2] 스레드 생명 주기

*휴면: 예를 들어, 워드프로세서에서 문서의 백업 시

9-5. 스레드의 제어 블록

스레드 제어 블록TCB, Thread Control Block

- 정보 저장
- 프로세스 제어 블록은 스레드 제어 블록의 리스트
- 스레드 간에 보호 하지 않음
- TCB의 내용
  - 실행 상태 : 프로세서 레지스터, 프로그램 카운터, 스택 포인터
  - 스케줄링 정보 : 상태(실행, 준비, 대기), 우선순위, 프로세서 시간
  - 계정 정보
  - 스케줄링 큐용 다양한 포인터
  - 프로세스 제어 블록PCB을 포함하는 포인터

9-5. 스레드의 제어 블록

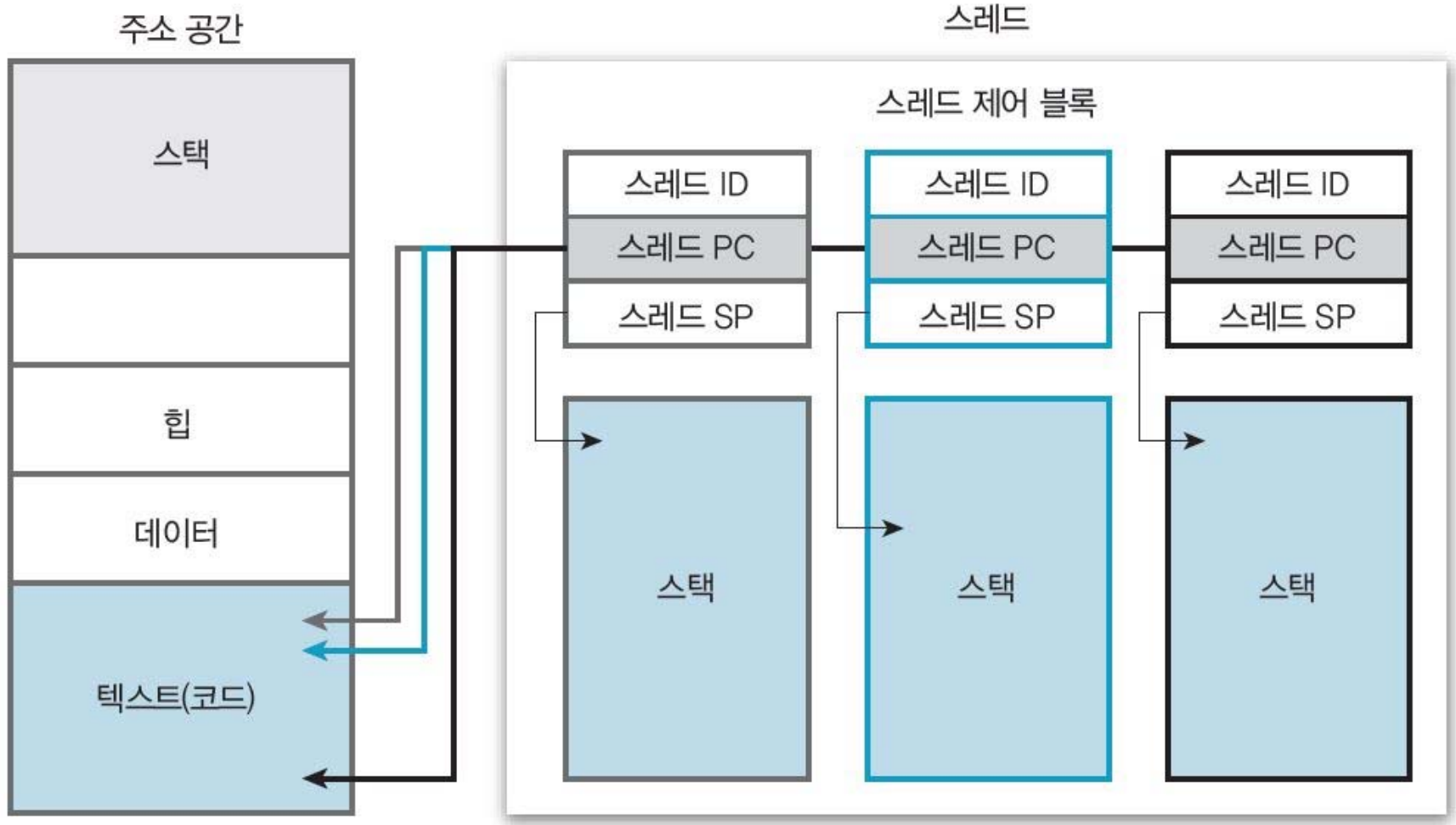


그림 3-19 스레드 제어 블록

9-6. 스레드의 구현

■ 스레드의 운영체제에 따른 구현

- 사용자 수준 스레드user-level thread : 다대일(n:1) 매핑
  - 스레드 라이브러리를 이용하여 작동

- 커널 수준 스레드kernel-level thread : 일대일(1:1) 매핑
  - 커널(운영체제)에서 지원

- 혼합형 스레드multiplexed thread : 다대다(n:m) 매핑
  - 사용자 수준 스레드와 커널 수준 스레드를 혼합한 형태

9-7. 사용자 수준 스레드

■ 사용자 수준 스레드의 개념

- 사용자 영역의 스레드 라이브러리로 구현
  - 스레드 라이브러리 : 스레드의 생성과 종료, 스레드 간의 메시지 전달, 스레드의 스케줄링과 문맥 등 정보 보관(POSIX 표준안의 스레드 확장판인 Pthread, Win32 스레드, 자바 스레드 API 등 주로 사용)

- 스레드와 관련된 모든 행위를 사용자 영역에서 함

- 스레드 교환에 커널이 개입하지 않아 커널에서 사용자 영역으로 전환 불필요

- 커널은 스레드가 아닌 프로세스를 한 단위로 인식, 프로세서 할당(프로세스 테이블 유지)

9-7. 사용자 수준 스레드

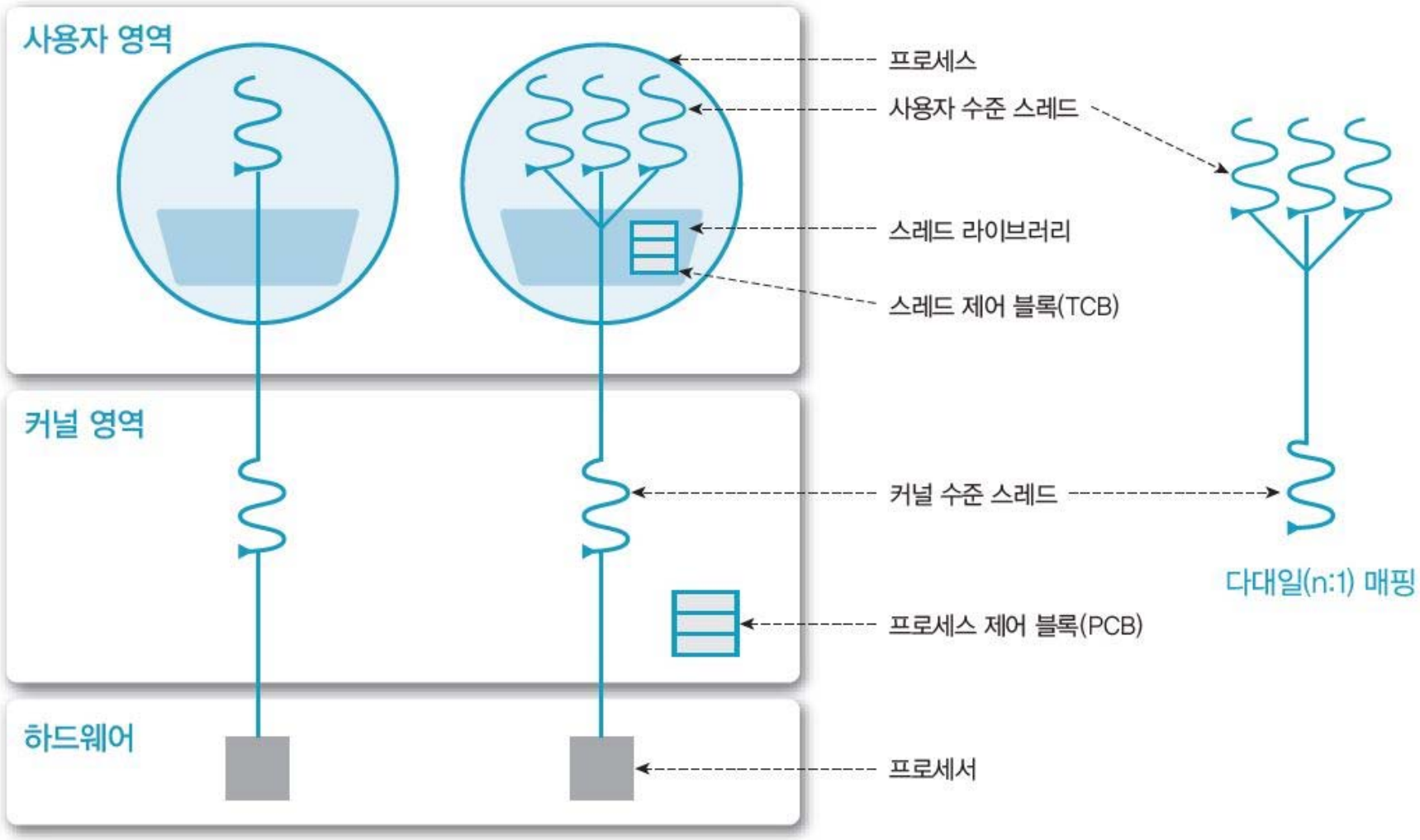


그림 3-20 사용자 수준 스레드 : 다대일(n :1) 매핑

9-7. 사용자 수준 스레드

사용자 수준 스레드의 장점

- 이식성 높음 : 커널에 독립적 스케줄링을 할 수 있어 모든 운영체제에 적용
- 오버헤드 적음 : 스케줄링, 동기화 위해 커널 호출 않으므로 커널 영역 전환 오버헤드 감소
- 유연한 스케줄링 가능 : 커널이 아닌 스레드 라이브러리에서 스레드 스케줄링 제어하므로 응용 프로그램에 맞게 스케줄링 가능

사용자 수준 스레드의 단점

- 시스템의 동시성 지원하지 않음 : 스레드가 아닌 프로세스 단위로 프로세서 할당하여 다중 처리 환경을 갖춰도 스레드 단위로 다중 처리 불가능
  동일한 프로세스의 스레드 한 개 가 대기 상태가 되면 이 중 어떤 스레드도 실행 불가
- 확장 제약이 따름 : 커널이 한 프로세스에 속한 여러 스레드에 프로세서를 동시에 할당 할 수 없어 다중 처리 시스템에서 규모 확장 곤란
- 스레드 간 보호 불가능 : 스레드 간 보호에 커널의 보호 방법 사용 불가
  스레드 라이브러리에서 스레드 간 보호를 제공해야 프로세스 수준에서 보호 가능

9-8. 커널 수준 스레드

커널 수준 스레드의 개념

- 사용자 수준 스레드의 한계 극복 방법으로, 커널이 스레드와 관련된 모든 작업 관리
- 한 프로세스에서 다수의 스레드가 프로세서 할당받아 병행 수행, 스레드 한 개가 대기 상태가 되면 동일한 프로세스에 속한 다른 스레드로 교환 가능. 이때도 커널 개입하므로 사용자 영역에서 커널 영역으로 전환 필요
- 커널이 직접 스케줄링하고 실행하기에 사용자 수준 스레드의 커널 지원 부족 문제 해결
- 커널이 전체 프로세스와 스레드 정보를 유지하여 오버헤드가 커짐
- 커널이 각 스레드 개별적으로 관리하여 동일한 프로세스의 스레드 병행 수행
- 동일한 프로세스에 있는 스레드 중 한 개가 대기 상태가 되도 다른 스레드 실행 가능
- 스케쥴링과 동기화를 하려면 더 많은 자원 필요
- 윈도우 NT·XP·2000, 리눅스, 솔라리스 9 이상 버전의 운영체제가 대표적

9-8. 커널 수준 스레드

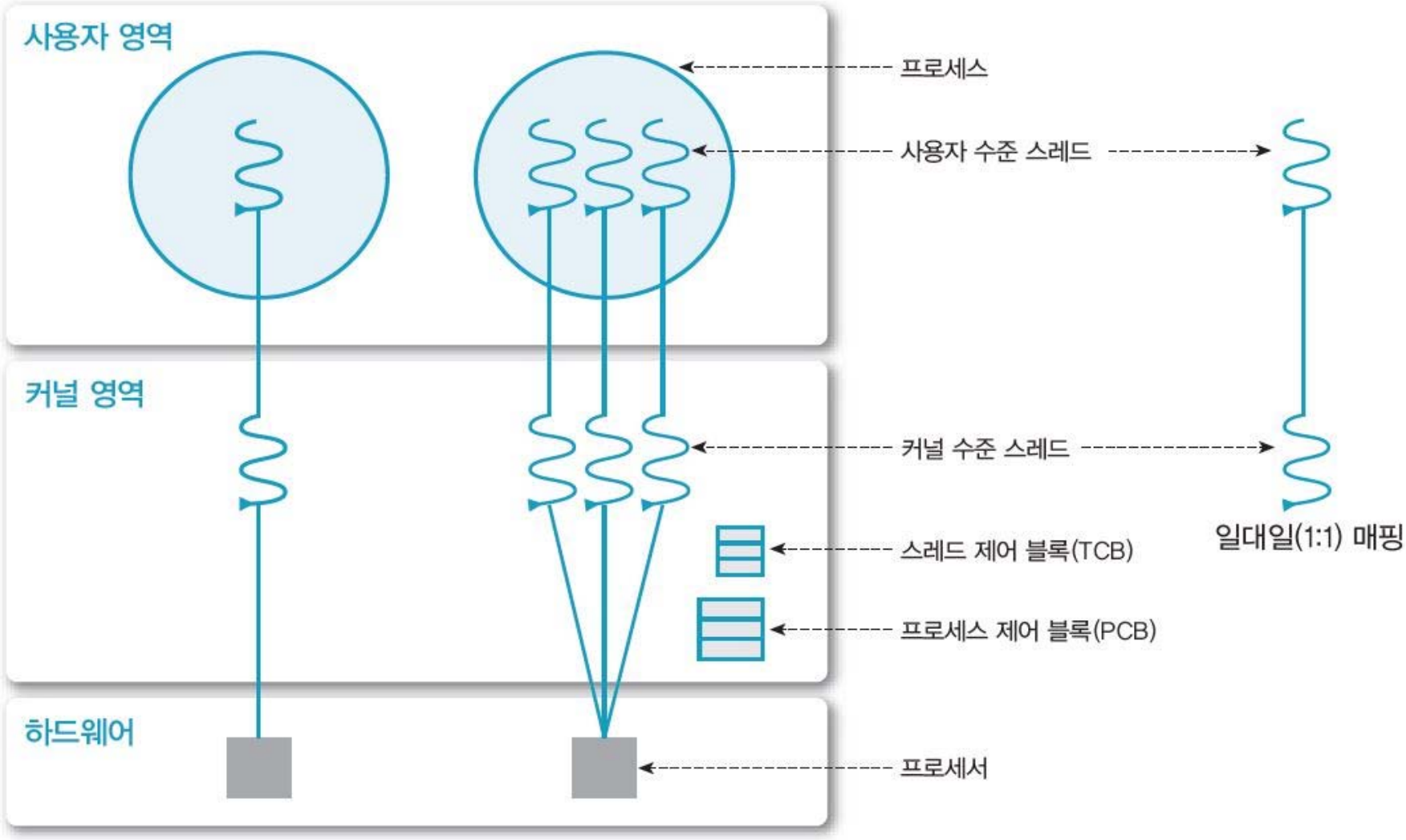


그림 3-21 커널 수준 스레드 : 일대일(1:1) 스레드 매핑

9-9. 혼합형 스레드

■ 혼합형 스레드의 개념

- 사용자 수준 스레드와 커널 수준 스레드 혼합한 구조
- 시스템 호출을 할 때 다른 스레드를 중단하는 다대일(n:1) 매핑의 사용자 수준 스레드와 스레드 수를 제한하는 일대일(1:1) 매핑의 커널 수준 스레드 문제 극복 방법
- 스레드 라이브러리가 최적의 성능 지원하도록 커널이 경량 프로세스 수동적으로 조절, 사용자 수준 스레드와 커널 수준 스레드가 다대다(n:m)로 매핑
- 커널 영역에서 병렬 처리 정도를 매핑이 결정할 수 있어 병행 실행이 의미가 없을 때 다대일(n:1) 매핑도 사용 가능
- 스레드 풀링 이용 일대일(1:1) 매핑으로 오버헤드 감소 가능
  - 스레드 풀링thread pooling : 시스템이 관리하는 스레드의 풀pool을 응용 프로그램에 제공, 스레드를 효율적으로 사용할 수 있게 하는 방법
    - 미리 생성한 스레드의 재사용으로 스레드 생성하는 시간 줄여 시스템의 부담 경감.
    - 동시 생성 가능한 스레드 수 제한하여 시스템의 자원 소비 감소, 응용 프로그램의 전체 성능 일정 수준 유지
- 솔라리스 9 이전 버전과 ThreadFiber 패키지가 있는 윈도우 NT·2000 가 대표적

9-9. 혼합형 스레드

■ 스레드의 특징

- 중량 프로세스와 특징이 동일하면서도 매우 효율적

- 다중 스레드 시스템인 매크는 커널이 여러 개의 요청에 동시에 서비스하도록 할 수 있음

- 다중 프로세스에서는 각 프로세스를 다른 프로세스와 독립적 실행, 각각 프로그램 카운터, 스택, 레지스터, 주소 공간 보유. 그러므로 프로세서가 하나인 시스템에서는 파일 서버가 디스크 입출력을 기다리는 동안 대기 상태가 될 수 있음. 동일한 주소 공간에서 서버 하나가 대기 상태에 있는 동안 독립적인 다른 서버를 만들 수 없기 때문. 그러나 스레드가 여러 개인 프로세스 하나는 서버 스레드 한 개가 대기 상태일 동안 동일 프로세스의 다른 스레드 실행 가능(작업량 증가하고 시스템 전체의 성능 향상)