

임베디드시스템설계

프로세스와 스레드

Byungjin Ko

Department of Intelligent Robotics

프로그램? 프로세스?

● 프로그램(program)

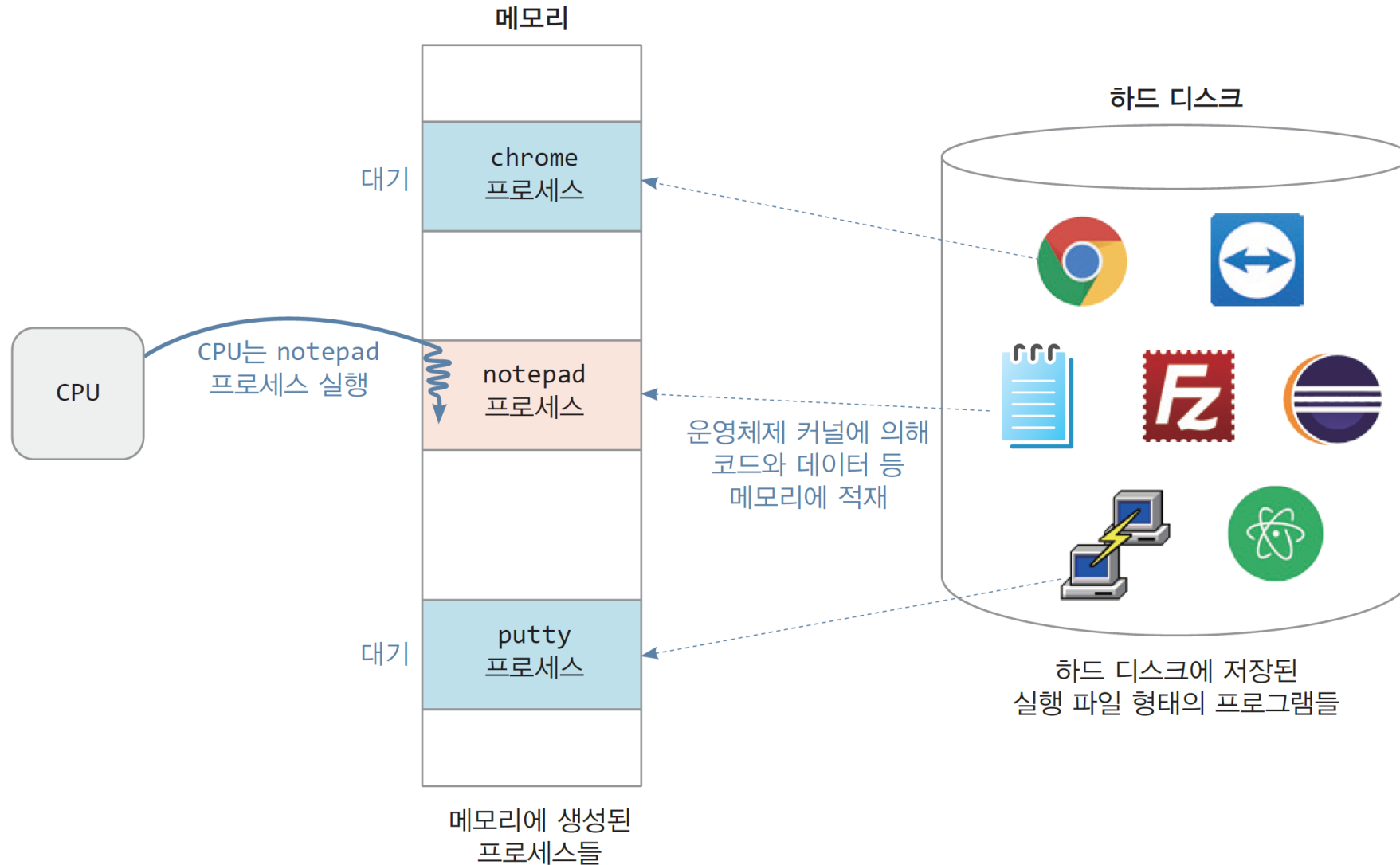
- 하드디스크 등의 저장 매체에 저장된 실행 가능한 파일

● 프로세스(process)

- 프로그램이 메모리에 적재되어 실행 중일 때 프로세스라 부름
 - 실행은 CPU로 의해 실행되고 있거나 또는 실행 대기인 상태를 뜻함
 - 실행에 필요한 모든 자원을 할당 받아야 함
 - 자원 : 코드 공간, 데이터 공간, 스택 공간, 힙 공간
- 프로세스의 특징
 - 운영체제는 프로그램을 메모리 적재하고 이를 프로세스로 다룸
 - 운영체제는 프로세스에게 실행에 필요한 메모리 할당하고 이곳에 코드와 데이터 등 적재
 - 프로세스들은 서로 독립적인 메모리 공간을 가지며 다른 프로세스의 영역에 접근 불허
 - 운영체제는 프로세스마다 고유한 번호(프로세스 ID) 할당
 - 프로세스의 관한 모든 정보는 커널에 의해 관리
 - 프로세스는 실행-대기-잠자기-대기-실행-종료 등의 생명 주기를 가짐
 - 프로세스 생성, 실행, 대기, 종료 등의 모든 관리는 커널에 의해 수행

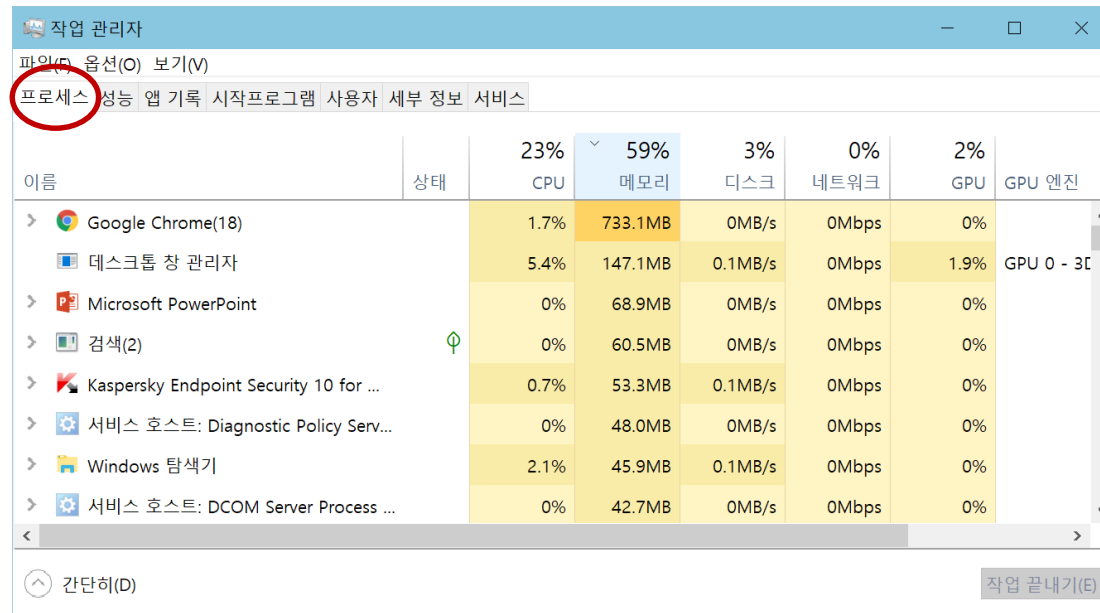
프로그램과 프로세스

- 프로세스들은 상호 독립적인 메모리 공간에서 실행



프로세스 관리

- 프로세스의 생성에서 종료까지, 관리는 모두 커널에 의해 이루어짐
 - 커널 영역에 프로세스 테이블을 만들고, 프로세스들 목록 관리
- 관리 내용
 - 프로세스 생성, 실행, 일시 중단 및 재개, 정보 관리, 프로세스 통신, 프로세스 동기화, 프로세스 중단, 프로세스 컨텍스트 스위칭

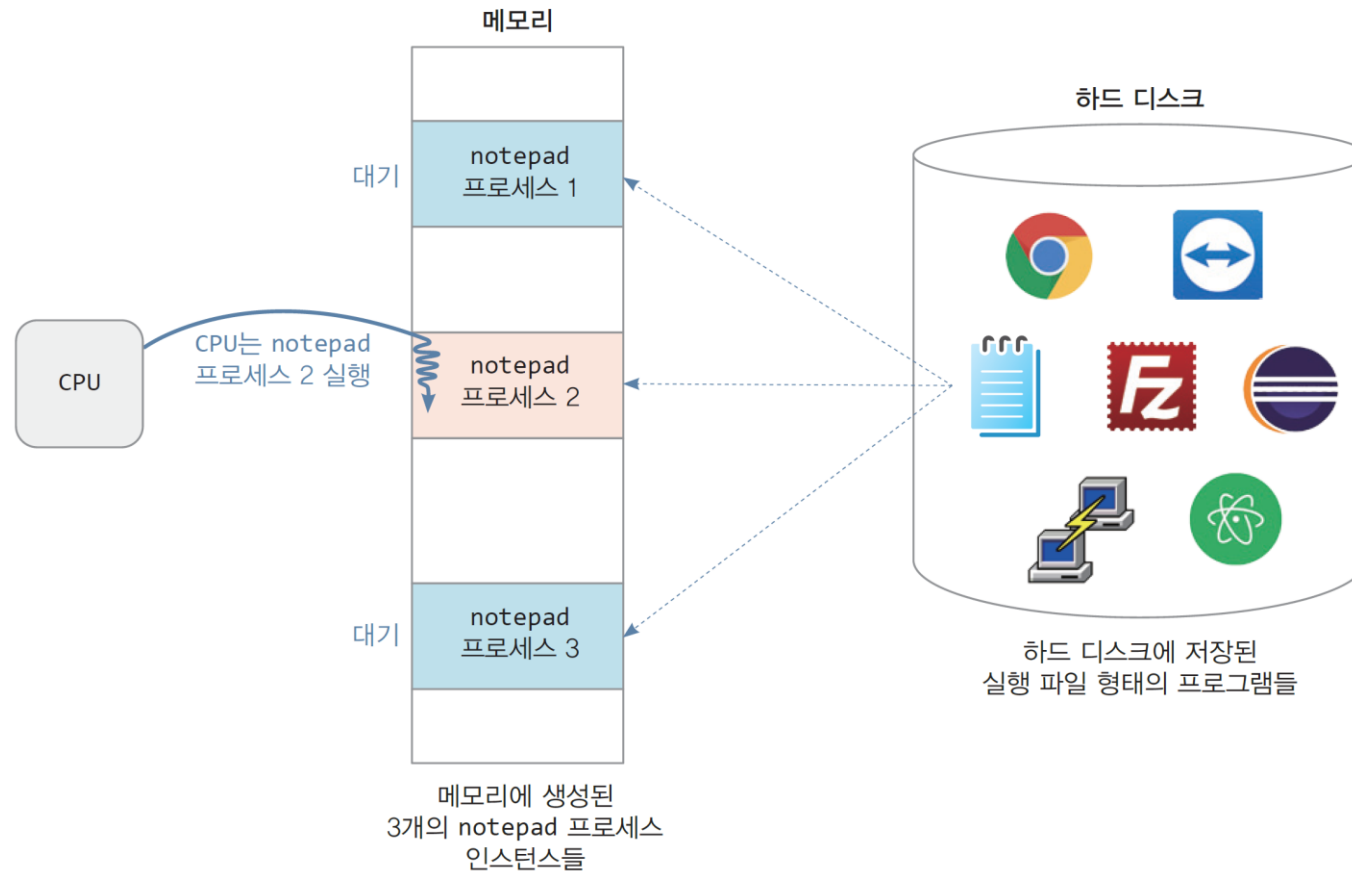


작업 관리자							
파일(F) 옵션(O) 보기(V)							
프로세스 성능 앱 기록 시작프로그램 사용자 세부 정보 서비스							
이름	상태	23% CPU	59% 메모리	3% 디스크	0% 네트워크	2% GPU	GPU 엔진
> Google Chrome(18)		1.7%	733.1MB	0MB/s	0Mbps	0%	
데스크톱 창 관리자		5.4%	147.1MB	0.1MB/s	0Mbps	1.9%	GPU 0 - 3D
> Microsoft PowerPoint		0%	68.9MB	0MB/s	0Mbps	0%	
> 검색(2)		0%	60.5MB	0MB/s	0Mbps	0%	
> Kaspersky Endpoint Security 10 for ...		0.7%	53.3MB	0.1MB/s	0Mbps	0%	
> 서비스 호스트: Diagnostic Policy Serv...		0%	48.0MB	0MB/s	0Mbps	0%	
> Windows 탐색기		2.1%	45.9MB	0.1MB/s	0Mbps	0%	
> 서비스 호스트: DCOM Server Process ...		0%	42.7MB	0MB/s	0Mbps	0%	

간단히(D) 작업 끝내기(E)

프로그램의 다중 인스턴스

- 한 프로그램을 여러 번 실행시키면 어떻게 될까?
 - 프로그램 실행 시마다 독립된 프로세스 생성 -> 프로세스들을 프로그램의 **다중 인스턴스**라고 부름
 - 각 프로세스에게 독립된 메모리 공간 할당
 - 각 프로세스를 별개의 프로세스로 취급



프로세스 구성

코드(code) 영역

- 실행될 프로그램 코드가 적재되는 영역 (text 영역으로도 불림)
 - 사용자가 작성한 모든 함수의 코드
 - 사용자가 호출한 라이브러리 함수들의 코드

데이터(data) 영역

- 프로그램에서 고정적으로 만든 변수 공간
 - 사용자가 선언한 전역 변수가 적재됨
 - 라이브러리에 선언된 전역 변수가 적재됨
- 프로세스 적재 시 할당, 종료 시 소멸

힙(heap) 영역

- 프로세스의 실행 도중 동적으로 사용할 수 있도록 할당된 공간
 - malloc() 등으로 할당받는 공간은 힙 영역에서 할당
 - 힙 영역에서 아래 번지로 내려가면서 할당

스택(stack) 영역

- 함수가 실행될 때 사용될 데이터를 위해 할당된 공간
 - 매개변수들, 지역변수들, 함수 종료 후 돌아갈 주소 등
 - 함수는 호출될 때, 스택 영역에서 위쪽으로 공간 할당
 - 함수가 return하면 할당된 공간 반환
- 함수 호출 외에 프로세스에서 필요시 사용 가능

사용자 공간 (user space)



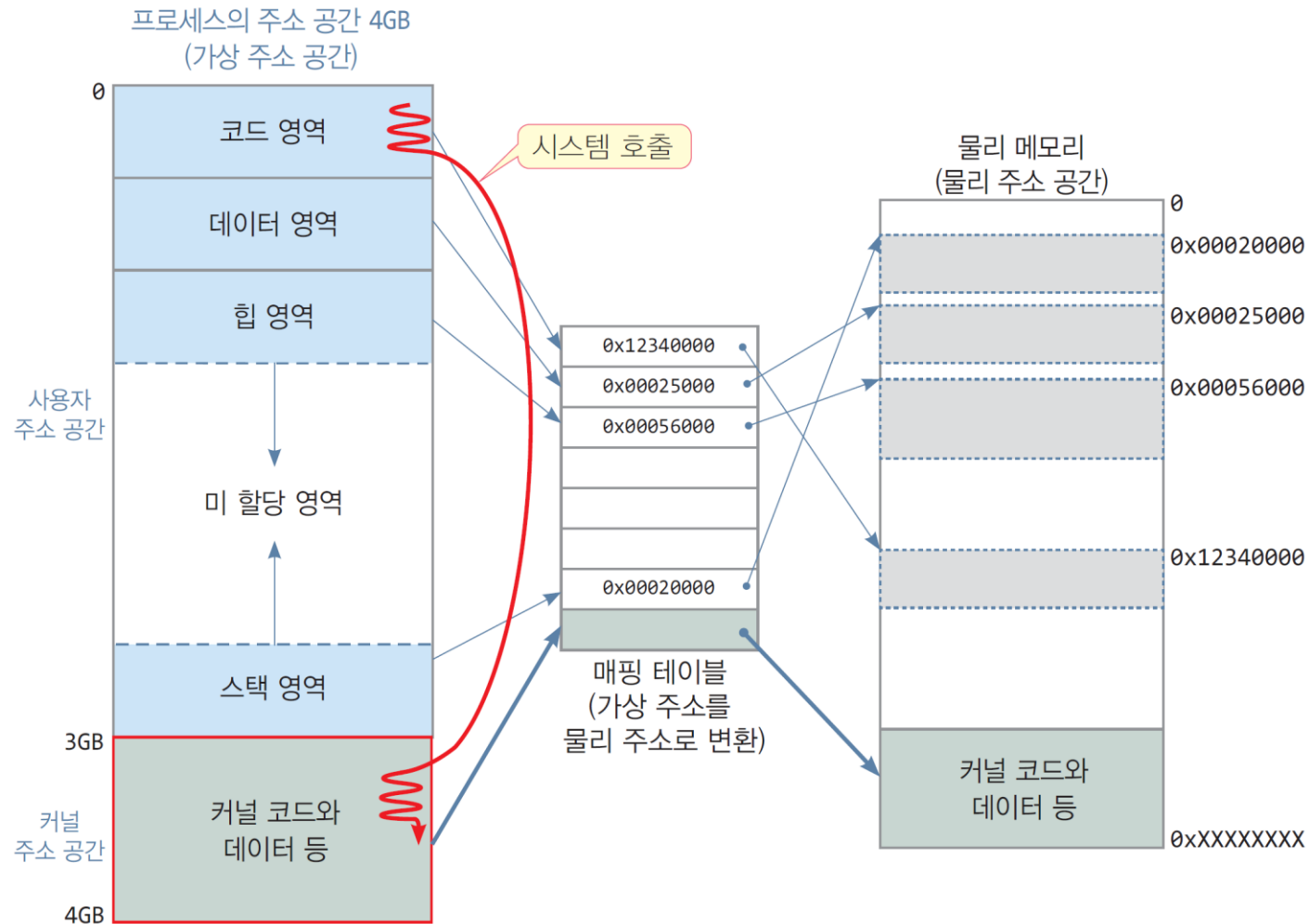
프로세스의 주소 공간

● 프로세스의 주소 공간은 가상 공간

- 가상 공간은 운영체제가 만들어 주는것임
- 프로세스가 사용하는 주소는 가상 주소임
 - 프로세스에서 0번지는 가상 주소 0번지
 - 물리 메모리의 0번지 아님
 - 가상 주소는 0번지부터 시작
 - 프로세스 내의 코드 주소, 전역 변수에 대한 주소, malloc()에 의해 리턴된 주소, 스택에 담긴 지역 변수의 주소는 모두 가상 주소
- 프로세스의 주소 공간(가상 주소 공간)은 사용자나 개발자가 보는 관점에서의 주소임
 - 사용자나 개발자는 프로그램이 0번지부터 시작하여, 연속적인 메모리 공간에 형성되고, 최대 크기의 메모리가 설치되어 있다고 볼 수 있음
- 실제 상황
 - 설치된 물리 메모리의 크기는 주소 공간보다 작을 수 있고,
 - 프로세스의 코드, 데이터, 힙, 스택은 물리 메모리에 흩어져 저장됨
 - 연속적인 메모리 공간이 아님

가상 주소 공간의 물리 메모리로의 매핑

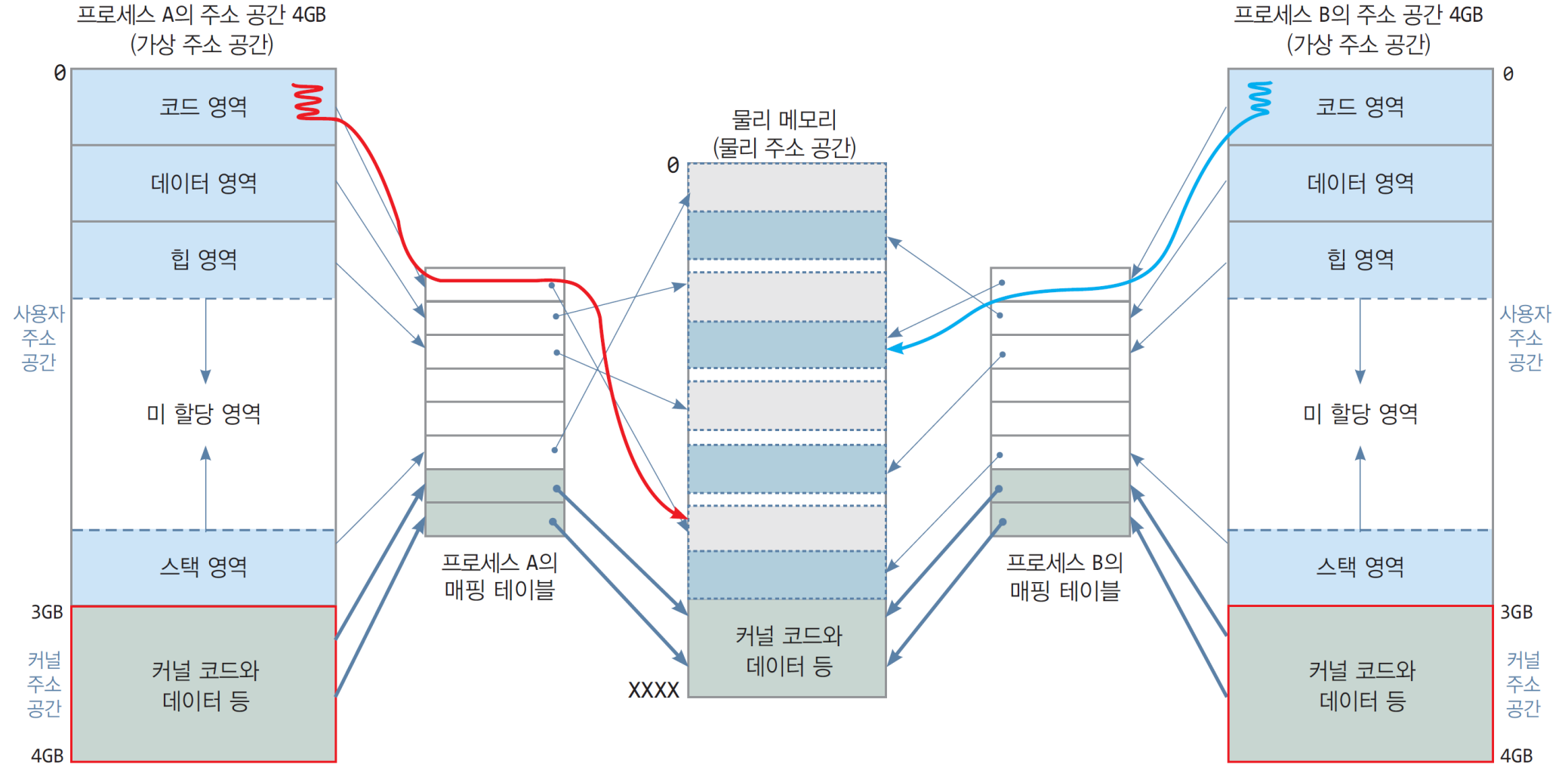
- 사용자는 연속적인 공간(가상 주소 공간)으로 생각 -> 그러나 가상 주소의 데이터가 물리 메모리에 분산되어 있어 어느 물리 번지에 있을지 알 수 없음



프로세스의 주소 공간은 가상 주소 공간

- 프로세스 주소 공간은 각 프로세스마다 주어지는가? yes
 - 프로세스마다 주소 공간은 별개임
 - 매핑 테이블은 프로세스를 생성할 때 프로세스마다 한 개씩 만듦
- 그러면, 프로세스 주소 공간은 충돌하는가? no
 - 프로세스 주소 공간은 가상 주소 공간임
 - 가상 주소가 물리 주소로 매핑되므로, 물리 메모리에서는 충돌하지 않음

프로세스들의 가상 주소 공간과 물리 메모리

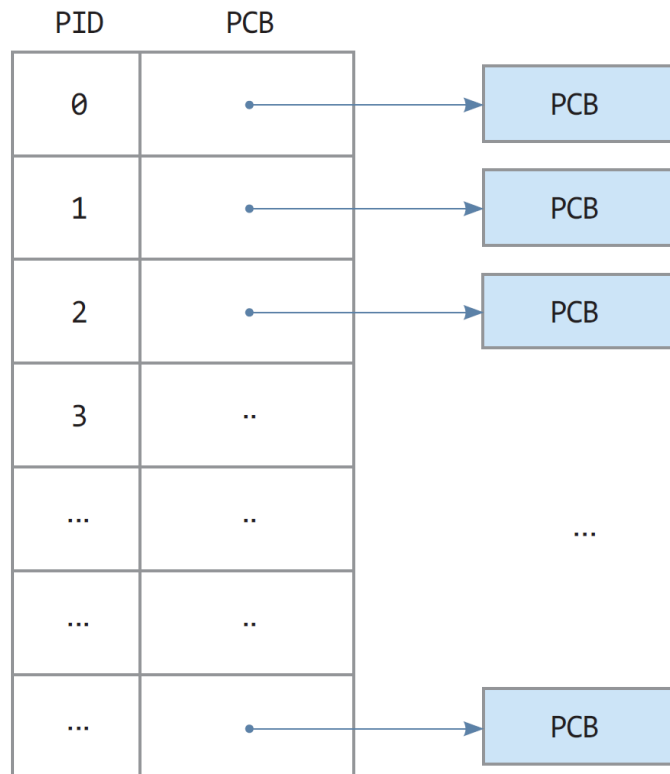


여러 프로세스들이 커널 공간 공유

프로세스 테이블과 프로세스 제어 블록

- 프로세스 테이블(Process Table)
 - 시스템의 모든 프로세스들을 관리하기 위한 표
 - 시스템에 한 개만 있음
 - 구현 방식은 운영체제마다 다름
- 프로세스 제어 블록(Process Control Block, PCB)
 - 프로세스 관리의 핵심 정보를 저장하는 구조체
 - 프로세스당 하나씩 존재
 - 프로세스가 생성될 때 만들어지고 종료되면 삭제
 - 커널에 의해 생성, 저장, 읽혀지는 등 관리
- 프로세스 테이블과 프로세스 제어 블록은 커널 공간에 생성됨
 - 커널 코드(커널 모드)만이 액세스 가능

프로세스 테이블과 프로세스 제어 블록



프로세스 테이블

프로세스 번호(PID)

부모프로세스 번호(PPID)

프로세스 상태(process state)

프로세스 컨텍스트
(PC, SP, 범용 레지스터 등 CPU 레지스터들)

스케줄링 정보
(priority, nice 값, 프로세스가 사용한 CPU 시간 등)

프로세스의 종료 코드(exit code)

프로세스의 오픈 파일 테이블

메모리 관리를 위한 정보들
(페이지 테이블 주소 등)

프로세스사이의 통신 정보들

회계 정보(accounting info)

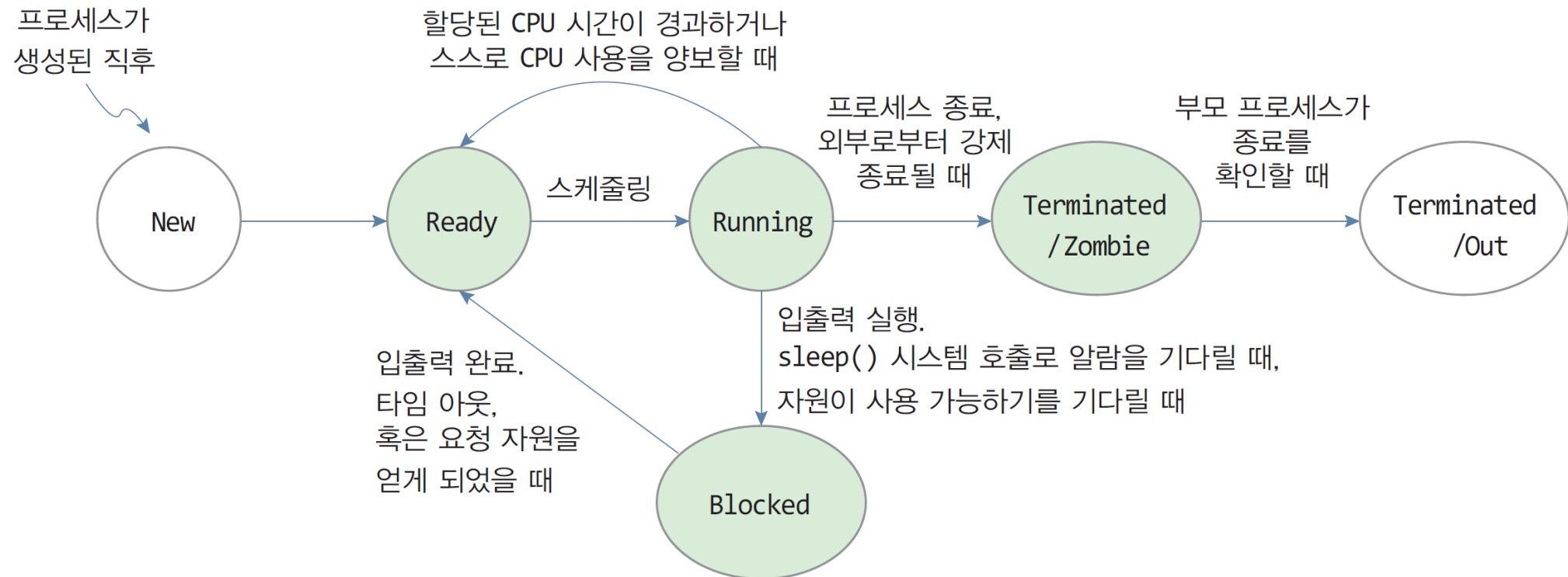
프로세스 소유자 이름(user name)

기타 프로세스가 사용중인 입출력 장치 목록 등

프로세스의 상태 변이

프로세스의 생명 주기

- 프로세스는 탄생에서 종료까지 여러 상태로 바뀌면서 실행
- 상태 정보는 PCB에 기록되고, 상태가 바뀔 때마다 갱신됨



프로세스의 상태 (참고)

- New(생성 상태)
 - 프로세스가 생성된 상태. 메모리 할당 및 필요한 자원이 적재된 상태, PCB에 New 상태로 등록. 실행 준비를 마치면 Ready 상태로 바뀜
- Ready(준비 상태)
 - 프로세스가 스케줄링을 기다리는 '준비 상태'
 - 프로세스는 준비 큐에서 대기
 - 스케줄링되면 Running 상태로 되고 CPU에 의해 실행됨
- Running(실행 상태)
 - 프로세스가 CPU에 의해 현재 실행되고 있는 상태
 - CPU의 시간할당량(타임슬라이스)이 지나면 다시 Ready 상태로 바뀌고 준비 큐에 삽입
 - 프로세스가 입출력을 시행하면 커널은 프로세스를 Blocked 상태로 만들고 대기 큐에 삽입
- Blocked/Wait(블록 상태)
 - 프로세스가 자원을 요청하거나, 입출력을 요청하고(예: read() 시스템 호출) 완료를 기다리는 상태
 - 입출력이 완료되면 프로세스는 Ready 상태로 바뀌고 준비 큐에 삽입
- Terminated/Zombie 상태
 - 프로세스가 불완전 종료된 상태(좀비 상태)
 - 프로세스가 차지하고 있던 메모리와 할당받았던 자원들을 모두 커널에 의해 반환됨. 커널에 의해 열어 놓은 파일도 닫힘
 - 하지만, 프로세스 테이블의 항목과 PCB가 여전히 시스템에서 제거되지 않은 상태
 - 프로세스가 남긴 종료 코드(PCB에 있음)를 부모 프로세스가 읽어가지 않아 완전히 종료되지 않은 상태 - 좀비 상태라고 부름
- Terminated/Out 상태
 - 프로세스가 종료하면서 남긴 종료 코드(PCB에 있음)를 부모 프로세스가 읽어 가서 완전히 종료된 상태
 - 프로세스 테이블의 항목과 PCB가 시스템에서 완전히 제거된 상태

프로세스의 문제점

- unix 기반의 운영체제들은 프로세스를 실행단위(task)로 정의했던 적이 있음
- 다중 프로세스를 이용한 멀티태스킹
 - 다중 프로세스는 여러 프로세스를 동시에 실행 시키는 것을 뜻함
 - 응용프로그램에서 여러 프로세스를 생성하여 동시에 여러 작업 실행
 - 운영체제는 스케줄링을 통해 여러 프로세스(작업)를 번갈아 실행
- 프로세스를 실행 단위로 하는 멀티태스킹의 문제점
 - 프로세스 생성의 큰 오버헤드:
 - 프로세스를 위한 메모리 할당, 부모프로세스로부터 복사
 - PCB 생성, 매핑 테이블(페이지 테이블) 생성 등
 - 프로세스 컨텍스트 스위칭의 큰 오버헤드:
 - CPU 레지스터들을 컨텍스트로 PCB에 저장, 새 프로세스 컨텍스트를 PCB에서 CPU로 옮기는 시간
 - CPU가 참고할 매핑 테이블(페이지 테이블)의 교체 시간
 - CPU 캐시에 새 프로세스의 코드와 데이터가 채워지는데 걸리는 시간 등
 - 프로세스 사이 통신의 어려움:
 - 프로세스가 다른 프로세스의 메모리에 접근 불가
 - 프로세스 사이의 통신을 위한 제 3의 방법 필요
 - 커널 메모리나 커널에 의해 마련된 메모리 공간을 이용하여 데이터 송수신
 - 신호, 소켓, 메시지 큐, 세마포어, 공유메모리, 메모리맵 파일 등
 - 이 방법들은 코딩이 어렵고, 실행 속도 느리고, 운영체제 호환성 부족

스레드 출현 목적

- 프로세스를 실행 단위로 하는 멀티태스킹의 문제점
 - 커널에 많은 시간, 공간 부담 -> 시스템 전체 속도 저하
- 효율적인 새로운 실행 단위 필요 : 스레드 출현
 - 프로세스보다 크기가 작은 실행 단위 필요
 - 프로세스보다 생성 및 소멸이 빠른 실행 단위 필요
 - 컨텍스트 스위칭이 빠른 실행 단위 필요
 - 통신이 쉬운 실행 단위 필요
- 스레드를 실행 단위로 다루는 운영체제를 멀티스레드 운영체제라고 함
 - 오늘 날의 대부분의 운영체제는 멀티스레드 운영체제임

스레드 개념

● 스레드는 실행 단위이며 스케줄링 단위

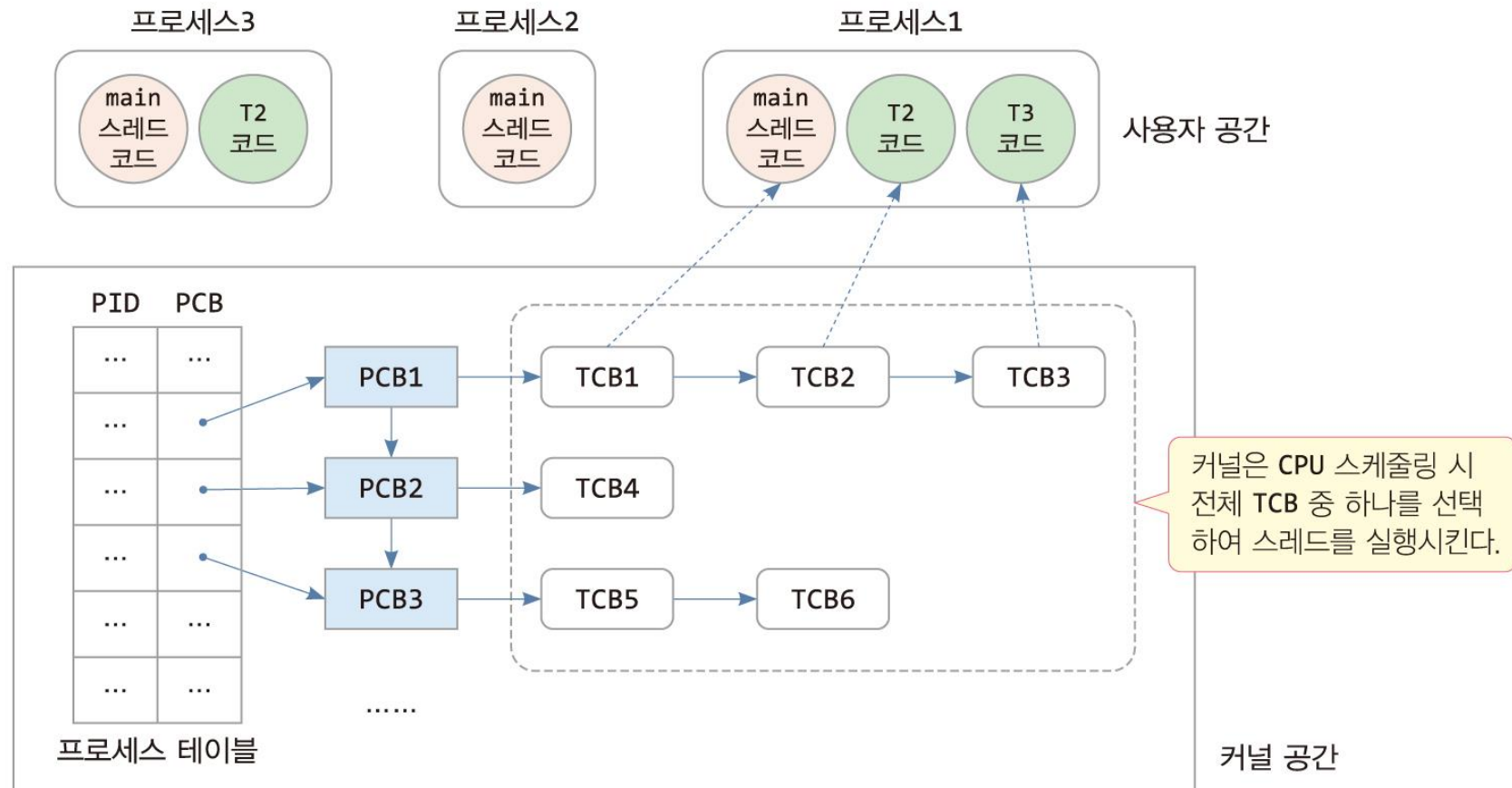
- 스레드는 응용프로그램 개발자에게는 작업을 만드는 단위
 - 하나의 응용프로그램에 동시에 실행할 여러 작업(스레드) 작성 가능
 - 작업은 독립적으로 실행되는 함수로 작성
- 스레드는 운영체제에게 실행 단위이고, 스케줄링 단위
- 스레드는 코드, 데이터, 힙, 스택의 주소공간을 가진 실체
- 스레드의 실행은 운영체제에 의해 제어됨
- 스레드마다 스레드 정보를 저장하는 구조체인 TCB(Thread Control Block) 있음

● 프로세스는 스레드들의 컨테이너

- 프로세스 개념이 여러 스레드를 포함할 수 있는 컨테이너와 같은 역할로 수정됨
- 프로세스는 반드시 1개 이상의 스레드로 구성
 - 프로세스가 생성될 때 운영체제에 의해 자동으로 1개의 스레드 생성 : 메인 스레드(main 스레드)라고 부름

프로세스와 스레드 관리 구성

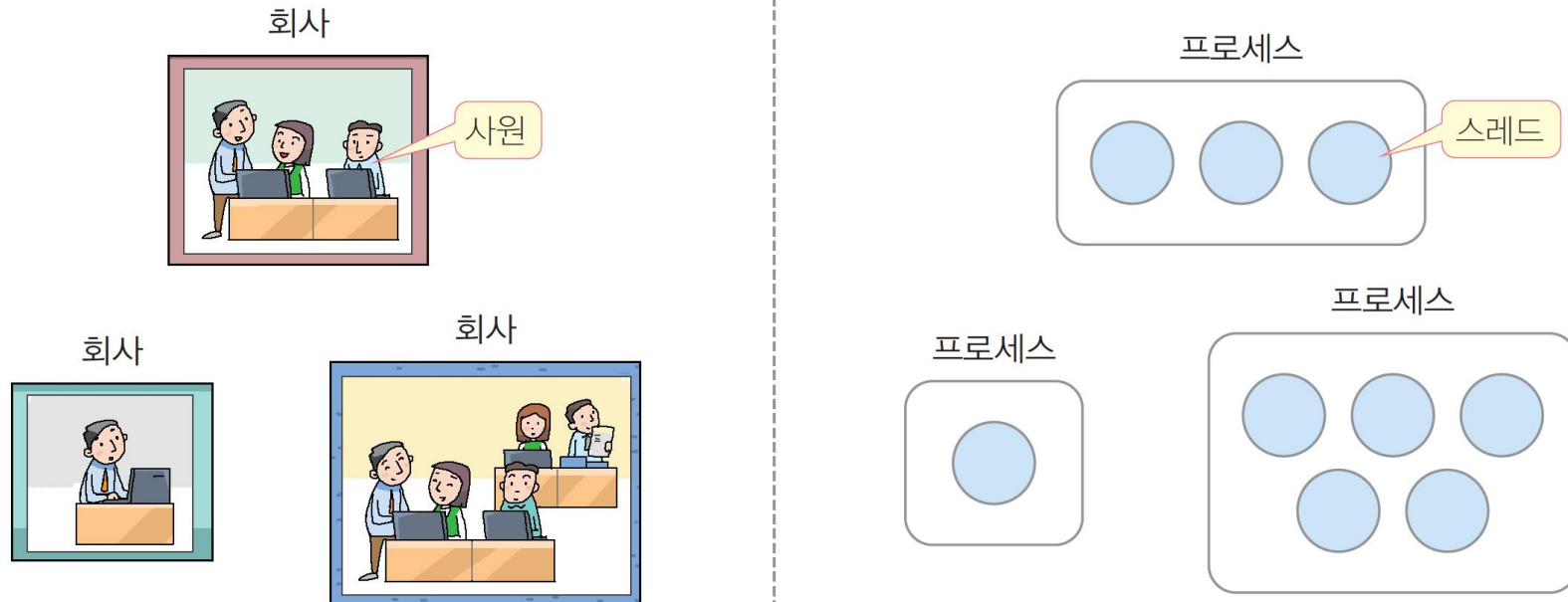
- 운영체제는 응용프로그램을 실행시키기 위해 프로세스를 생성하고 관리함
- 프로세스를 생성할 때 커널은 자동으로 프로세스 내에 1개의 스레드를 생성하며 이를 메인 스레드라고 부름



* 스레드마다 TCB가 만들어지고 서로 연결된다. 프로세스에 속한 스레드들을 관리하기 위해 PCB는 TCB와 연결된다.

프로세스는 스레드들의 컨테이너

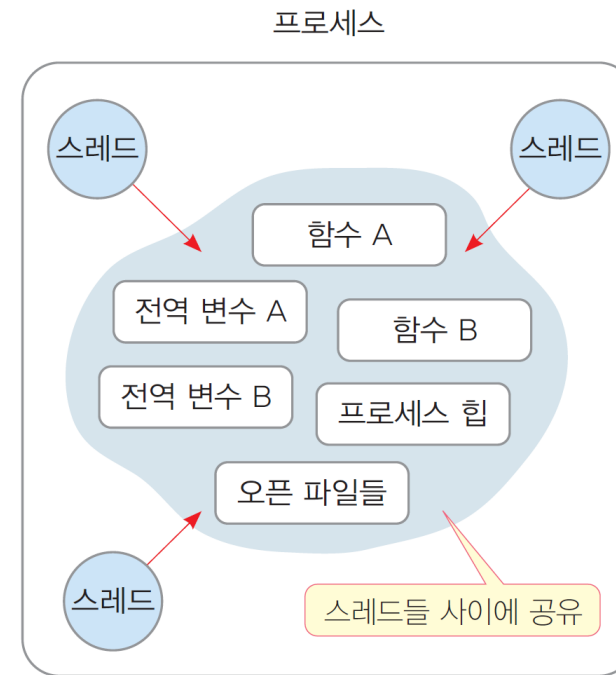
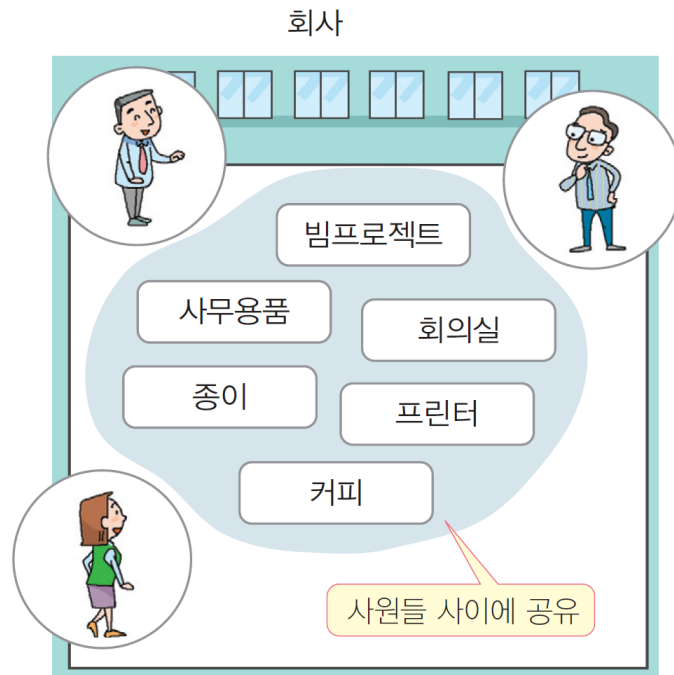
- 프로세스는 회사, 스레드는 직원에 비유
 - 직원은 회사의 목적을 위해 일을 하는 단위
 - 스레드는 프로세스(응용프로그램)의 목적을 위해 동시에 실행될 작업(코드) 단위
 - 직원이 3명이면 동시에 일을 하는 사람이 3명인 것처럼, 프로세스에 3개의 스레드가 있으면 3개의 작업이 동시에 실행



스레드 개념

프로세스는 스레드들의 공유 공간(환경) 제공

- 모든 스레드는 프로세스의 코드, 데이터, 힙을 공유하며, 프로세스의 스택 공간을 나누어 사용
- 공유되는 공간을 이용하면 스레드 사이의 통신 용이



스레드 개념

● 스레드가 실행할 작업은 함수로 작성

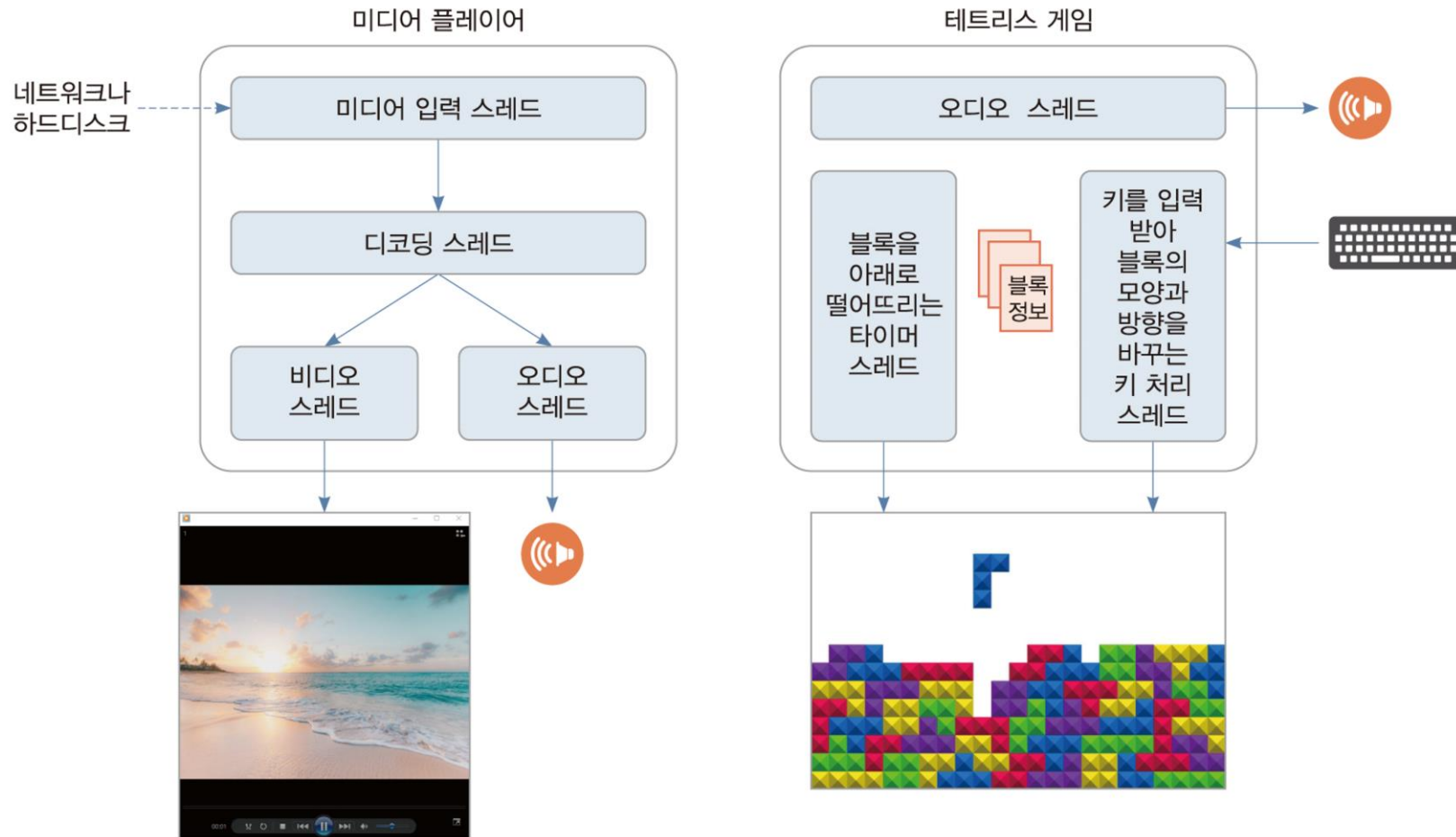
- 응용프로그램 개발자는 스레드가 실행할 작업을 함수로 작성
 - 함수를 실행할 스레드 생성을 운영체제에게 요청할 때 스레드 생성
 - 운영체제는 TCB 생성, 함수의 주소를 스레드 실행 시작 주소로 TCB에 등록
 - 스레드 생성은 곧 TCB 생성
- 운영체제는 TCB 리스트로 전체 스레드 관리
 - 스레드 스케줄 : TCB 중에서 하나 선택, 스레드 단위로 스케줄
 - TCB에 기록된 스레드의 시작 주소를 CPU에 적재하면 실행 시작됨

● 스레드의 생명과 프로세스의 생명

- 스레드로 만든 함수가 종료하면 스레드 종료
- 스레드가 종료하면 TCB 등 스레드 관련 정보 모두 제거
- 프로세스에 속한 모든 스레드가 종료될 때, 프로세스 종료

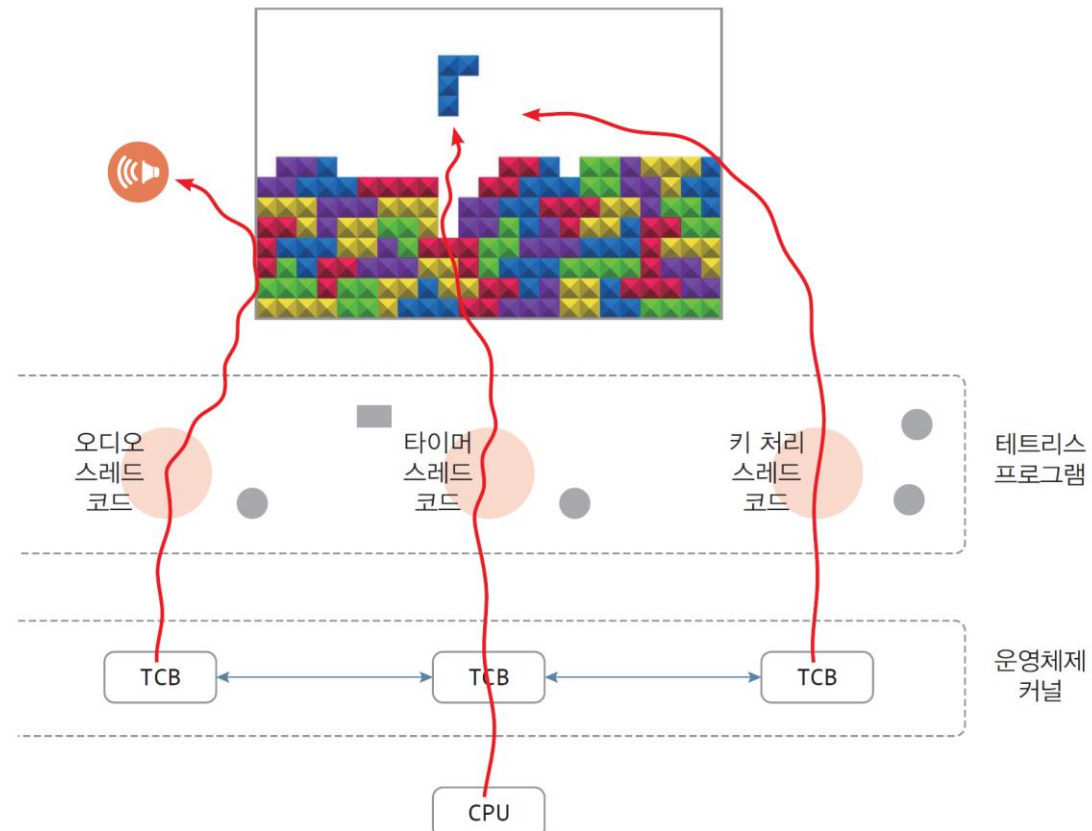
멀티스레드 응용프로그램 사례

- 현재 대부분의 멀티태스킹 응용프로그램은 멀티스레딩 기법으로 개발함



멀티스레딩 분석

- 운영체제 커널이 스케줄링을 통해 한개의 TCB를 선택하고 CPU에게 TCB에 저장된 주소에서 스레드 코드를 실행하게 함



* 함수는 다른 함수에 의해 호출되어 실행되지만,
스레드 함수의 코드는 CPU가 바로 실행하도록 커널에 의해 제어됨

멀티스레딩과 concurrency, parallelism

concurrency(동시성)

- 1개의 CPU에서 2개 이상의 스레드가 동시에 실행 중인 상태
 - 스레드가 입출력으로 실행이 중단될 때 다른 스레드 실행
 - 타임 슬라이스 단위로 CPU를 사용하도록 번갈아 스레드 실행
- concurrency 사례 - 3개의 스레드가 1개 CPU에 의해 동시 실행



parallelism(병렬성)

- 2개 이상의 스레드가 다른 CPU에서 같은 시간에 동시 실행
- parallelism 사례 - 3개의 스레드가 3개의 CPU에 의해 동시 실행

