**Chapter 10 프로세스와 스레드**

***10-1*** **프로세스 개요**

**프로세스 생성**: 보조기억장치에 있는 프로그램을 메모리에 적재하고 실행하는 순간 프로그램은 프로세스가 된다

**포그라운드 프로세스**: 사용자가 볼 수 있는 공간에서 실행되는 프로세스

**백그라운드 프로세스**: 사용자가 보지 못하는 뒤편에서 실행되는 프로세스

* 사용자와 직접 상호작용할 수 있는 백그라운드 프로세스

ex) 이메일 클라이언트, 음악 스트리밍 앱, 채팅 앱

* 사용자와 상호작용하지 않는 백그라운드 프로세스

ex) 자동 업데이트 서비스, 시스템 백업, 로그 수집 및 분석

* 유닉스 체계의 운영체제에서는 **데몬**이라고 부르고 윈도우 운영체제에서는 **서비스**라고 부름

**<프로세스 제어 블록>**

**운영체제**는 빠르게 번갈아 수행되는 프로세스의 실행 순서를 관리, 프로세스에 CPU를 비롯한 자원을 배분 -> **프로세스 제어블록(PCB)** 이용

**프로세스 제어 블록(PCB)**: 프로세스와 관련된 정보를 저장하는 자료 구조(상품의 태그와 같은 역할)

해당 프로세스를 식별하기 위해 꼭 필요한 정보들이 저장. 메모리에서 커널 영역에 생성.

프로세스 생성 시에 만들어지고 실행이 끝나면 폐기됨

**PCB에 담기는 정보**

1. **프로세스 ID (PID)**: 특정 프로세스를 식별하기 위해 보유하는 고유한 번호

같은 일을 수행하는 프로그램이라도 두 번 실행하면 PID가 다른 두개의 프로세스 생성

1. **레지스터 값**: 프로세스는 자신의 실행 차례가 돌아오면 이전까지 사용했던 레지스터의 중간값들을 모두 복원 -> 작업을 이어서 실행
2. **프로세스 상태**: 프로세스의 상태 정보(준비, 대기, 실행 등)가 저장
3. **CPU 스케줄링 정보**: 프로세스가 언제, 어떤 순서로 CPU를 할당받는지에 대한 정보
4. **메모리 관리 정보**: 프로세스가 어느 주소에 저장되어 있는지에 대한 정보(베이스 레지스터, 한계 레지스터 값, 페이지 테이블 정보)
5. **사용한 파일과 입출력장치 목록**: 프로세스 실행 과정에서 특정 입출력장치나 파일을 사용하면 PCB에 해당 내용이 명시됨

**<문맥 교환>**

**문맥**: 하나의 프로세스 수행을 재개하기 위해 기억해야 할 정보 -> PCB에 표현되어 있음

**문맥 교환**: 기존 프로세스의 문맥을 PCB에 백업하고, 새로운 프로세스를 실행하기 위해 문맥을 PCB로 복구하여 새로운 프로세스를 실행하는 것, 여러 프로세스가 끊임없이 빠르게 번갈아 가며 실행되는 원리.

**문맥교환이 자주 일어나면?** 프로세스들이 동시에 실행되는 것처럼 보임. 하지만 너무 자주 하면 **오버헤드**가 발생할 수 있음

**<프로세스의 메모리 영역>**

커널영역 - PCB

사용자영역 - 코드 영역, 데이터 영역, 힙 영역, 스택 영역

**코드 영역(텍스트 영역)**: CPU가 실행할 **명령어**가 담겨있음. 읽기 전용 공간

**데이터 영역**: 프로그램이 실행되는 동안 유지할 데이터가 저장 ex)전역 변수

* 코드 영역과 데이터 영역은 크기가 고정된 **정적 할당 영역**
* 힙 영역과 스택 영역은 프로세스 실행 과정에서 크기가 변할 수 있는 **동적 할당 영역**

**힙 영역**: 프로그래머가 직접 할당할 수 있는 저장 공간 -> 언젠가는 해당 공간을 반환

메모리 공간을 반환하지 않으면-> 메모리 낭비를 초래(**메모리 누수**)

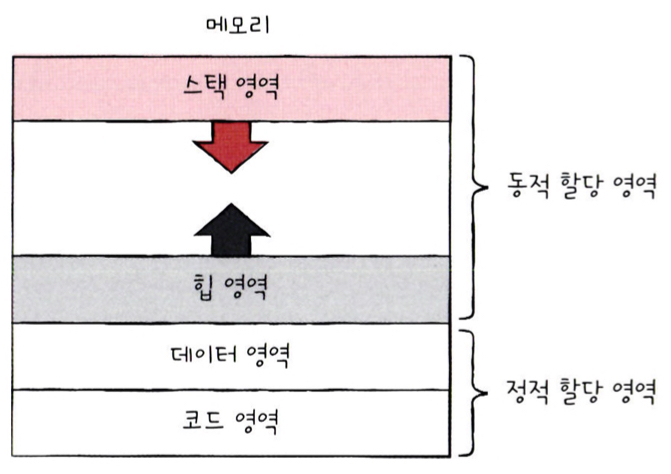
ex) 메모리 반환 방법은 언어마다 다르지만 free(), delete, close(), Dispose(), 가비지 컬렉터가 처리하는 방법등이 존재

힙 영역은 동적 메모리를 할당

**스택 영역**: 데이터를 일시적으로 저장하는 공간 ex)매개 변수, 지역 변수

스택영역은 정적 메모리를 할당

**동적 할당 영역**은 데이터가 쌓일 때 할당되는 주소가 겹치지 않도록 하기 위해 **힙 영역**은 메모리의 낮은 주소에서 높은 주소로 할당, **스택 영역**은 높은 주소에서 낮은 주소로 할당



***10-2*** **프로세스 상태와 계층 구조**

**<프로세스 상태>**

하나의 프로세스는 여러 상태를 거치며 실행. 운영체제는 프로세스의 상태를 PCB를 통해 인식하고 관리

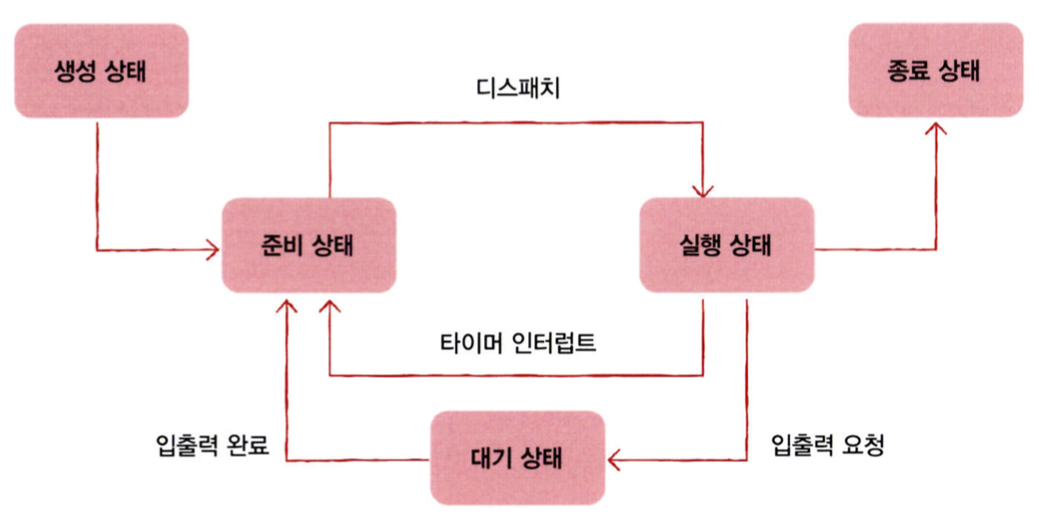
**생성 상태(new)**: 프로세스를 생성 중인 상태. 이제 막 메모리에 적재되어 PCB를 할당받은 상태. 곧바로 실행되지 않고 준비 상태가 되어 CPU의 할당을 기다림

**준비 상태(ready)**: 당장이라도 CPU를 할당받아 실행할 수 있지만 차례를 기다리는 상태. 준비 상태에서 실행 상태로 전환되는 것을 **디스패치(dispatch)**라고 함

**실행 상태(running)**: CPU를 할당받아 실행 중인 상태. 할당된 일정 시간 동안만 CPU 사용 가능. 할당된 시간을 모두 사용한다면 다시 **준비 상태**. 실행 도중 입출력장치를 사용하여 입출력 장치의 작업이 끝날 때까지 기다려야 한다면 **대기 상태**

**대기 상태(blocked)**: 입출력 장치의 작업을 기다리는 상태. 입출력 작업이 완료되면 해당 프로세스는 다시 준비 상태로 CPU 할당을 기다림

**종료 상태(terminated)**: 프로세스가 종료된 상태 -> 운영체제는 PCB와 사용한 메모리를 정리



**프로세스 상태 다이어그램**

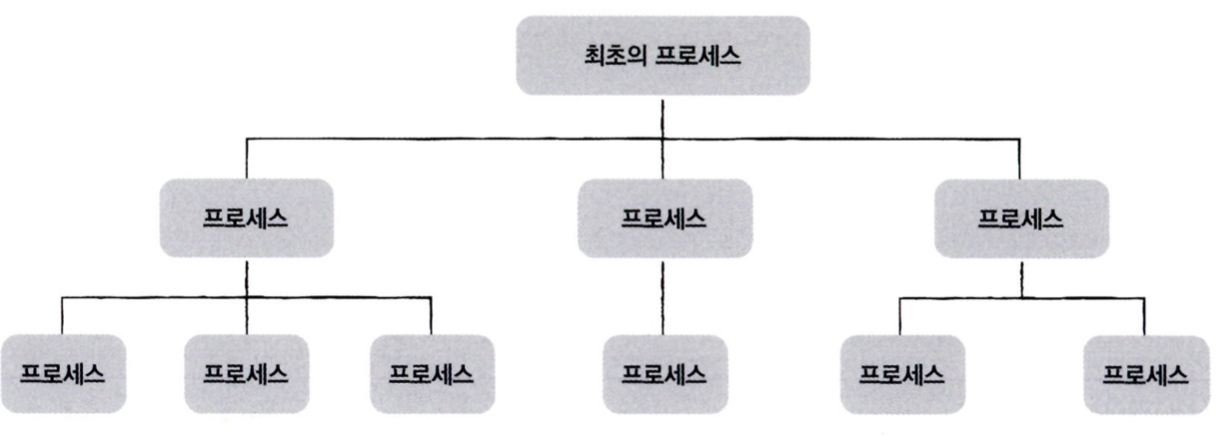
**<프로세스 계층 구조>**

프로세스는 실행 도중 시스템 호출을 통해 다른 프로세스를 생성 가능

**부모 프로세스**: 새 프로세스를 생성한 프로세스

**자식 프로세스**: 부모 프로세스에 의해 생성된 프로세스

이들은 각기 다른 PID를 가지는데 일부 운영체제에서는 자식 프로세스의 PCB에 부모 프로세스의 PID인 **PPID**가 기록되기도 함



**프로세스 계층 구조**

**최초의 프로세스**: 유닉스 운영체제에서는 init, 리눅스 운영체제에서는 systemd, macOS에서는 launchd. 최초의 프로세스의 PID는 항상 1번

**<프로세스 생성 기법>**

**fork**: 자기 자신 프로세스의 복사본을 만드는 시스템 호출 -> 부모 프로세스가 호출

**exec**: 자신의 메모리 공간을 새로운 프로그램으로 덮어쓰는 시스템 호출 -> 자식 프로세스가 호출. exec 호출 시 코드 영역과 데이터 영역의 내용이 실행할 프로그램의 내용으로 바뀌고, 나머지 영역은 초기화

* 부모 프로세스가 자식 프로세스를 fork한 뒤에 부모 프로세스, 자식 프로세스 누구도 exec를 호출하지 않는 경우도 존재 -> 같은 코드를 병행하여 실행하는 프로세스

***10-3*** **스레드**

**<프로세스 생성 기법>**

**단일 스레드 프로세스**: 전통적인 관점에서 하나의 프로세스는 한 번에 하나의 일만 처리. 실행의 흐름 단위가 하나인 프로세스

-> 스레드라는 개념이 도입되면서 하나의 프로세스가 한 번에 여러 일을 동시에 처리 가능

**스레드**: 프로세스를 구성하는 실행의 흐름 단위 . 프로세스 내에서 각기 다른 스레드 ID, 프로그램 카운터 값을 비롯한 레지스터 값, 스택으로 구성. **프로세스 자원은 공유**하며 실행

**<멀티프로세스와 멀티스레드>**

**멀티프로세스**: 여러 프로세스를 동시에 실행하는 것

**멀티스레드**: 여러 스레드로 프로세스를 동시에 실행하는 것

프로세스는 기본적으로 자원을 공유 x, 스레드는 같은 프로세스 내의 자원을 공유 o

프로세스를 복제하면 자원이 중복으로 존재하기 때문에 낭비

**멀티스레드의 장점**: 멀티프로세스보다 메모리를 효율적으로 사용 가능. 자원을 공유하기 때문에 서로 협력과 통신에 유리

**멀티스레드의 단점**: 하나의 스레드에 문제가 생기면 프로세스 전체에 문제가 생길 수 있음