10장 프로세스와 스레드

프로세스

- 실행중인 프로그램
- 포그라운드 프로세스
  - 사용자가 보는 앞에서 실행되는 프로세스
- 백그라운드 프로세스
  - 사용자가 보지 못하는 뒤편에서 실행되는 프로세스
  - 사용자와 직접 상호작용할 수 있는 백그라운드 프로세스도 있지만, 반대도 존재
  - 유닉스 체제의 운영체제 : 데몬
  - 윈도우 운영체제 : 서비스

프로세스 제어 블록 (PCB)

- 운영체제는 빠르게 번갈이 수행되는 프로세스의 실행 순서를 관리하고, 프로세스에 CPU를 비롯한 자원을 배분함 → 이를 PCB에서 이용함
- 커널 영역에서 생성됨
- 운영체제는 수많은 프로세스들 사이에서 PCB로 특정 프로세스를 식별하고 해당 프로세스를 처리하는 데 필요한 정보를 판단함
- 프로세스 생성 시에 만들어지고 실행이 끝나면 폐기됨

PCB 정보들

- `프로세스 ID`
  - 특정 프로세스를 식별하기 위해 부여하는 고유한 번호

- 레지스터 값
  - 프로세스는 자신의 실행 차례가 돌아오면 이전까지 사용했던 레지스터의 중간값들을 모두 복원함.
  - 그래야만 이전까지 진행했던 작업들을 그대로 이어 실행할 수 있기 때문
  - 따라서 PCB안에는 해당 프로세스가 실행하며 사용했던 프로그램 카운터를 비롯한 레지스터 값들이 담김
- 프로세스 상태
  - 현재 프로세스가 어떤 상태인지 PCB에 기록되어야 함
- CPU 스케줄링 정보
  - 프로세스가 언제, 어떤 순서로 CPU를 할당받을지에 대한 정보
- 메모리 관리 정보
  - 프로세스마다 메모리에 저장된 위치가 다름, 따라서 프로세스가 어느 주소에 저장되어 있는지에 대한 정보도 있어야 함
- 사용할 파일과 입출력장치 목록
  - 프로세스가 실행 과정에서 특정 입출력장치나 파일을 사용하면 해당 내용이 명시됨

문맥교환

- 문맥 : 하나의 프로세스 수행을 재개하기 위해 기억해야할 정보
  - PCB에 기록되는 정보들을 문맥이라고 봐도 됨
  - 실행 문맥을 잘 기억해 두면 언제든 해당 프로세스의 실행을 재개할 수 있기 때문에 프로세스가 CPU를 사용할 수 있는 시간이 다 되거나 예기치 못한 상황이 발생하여 인터럽트를 발생하면 운영체제는 해당 프로세스의 PCB에 문맥을 백업함
- 문맥 교환 : 기존 프로세스의 문맥을 PCB에 백업하고, 새로운 프로세스를 실행하기 위해 문맥을 PCB로부터 복구하여 새로운 프로세스를 실행하는 것

프로세스의 메모리 영역

- 코드 영역
  - 기계어로 이루어진 명령어가 저장
  - CPU가 실행할 명령어가 담겨 있기 때문에 쓰기가 금지되어 있음
  - 읽기 전용 공간
- 데이터 영역

    - 프로그램이 실행되는 동안 유지할 데이터가 저장되는 공간
    - 전역 변수
- 힙 영역
    - 프로그래머가 직접 할당할 수 있는 저장 공간
    - 힙 영역에 메모리 공간을 할당했다면 언젠가는 해당 공간을 반환해야 함
    - 메모리 누수 : 메모리 공간을 반환하지 않아 계속 메모리 낭비를 초래
- 스택 영역
    - 데이터를 일시적으로 저장하는 공간
    - 함수의 실행이 끝나면 사라지는 매개 변수, 지역 변수가 대표적임
    - 일시적으로 저장할 데이터는 PUSH, 더 이상 필요하지 않는 데이터는 POP

정적 할당 영역 : 코드 영역과 데이터 영역은 '크기가 고정된 영역'

동적 할당 영역: 힙 영역과 스택 영ㅇ역은 프로세스 실행 과정에서 크기가 변할 수 있는 영역

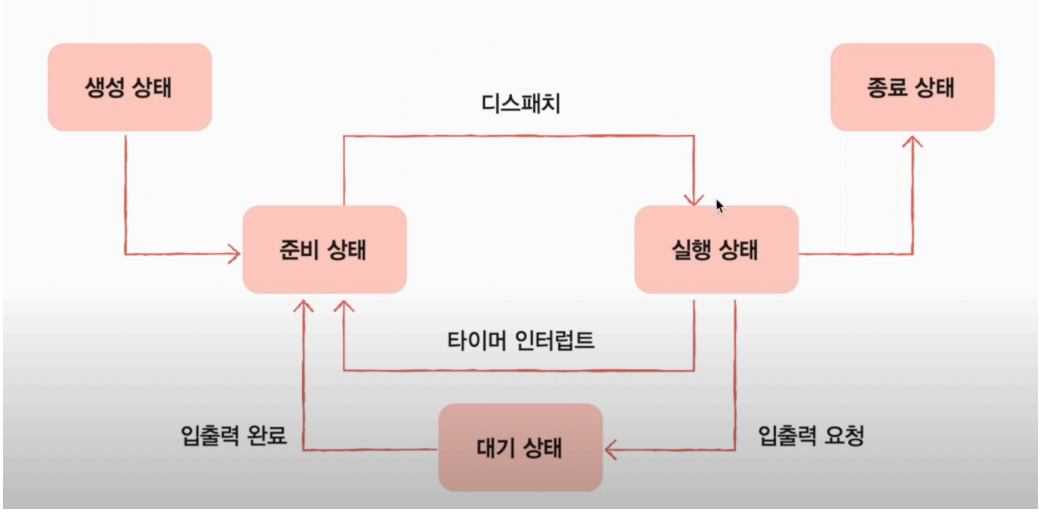
- 일반적으로 힙 영역은 메모리의 낮은 주소에서 높은 주소로 할당되고, 스택 영역은 높은 주소에서 낮은 주소로 할당됨, 따라서 서로 할당되는 주소가 겹칠일이 없음

프로세스 상태

- 생성 상태
    - 메모리에 적재되어 PCB를 할당 받은 상태
- 준비 상태
    - 당장이라도 CPU를 할당받아 실행할 수 있지만, 아직 차례가 아니므로 기다리고 있는 상태
- 실행 상태
    - CPU를 할당받아 실행 중인 상태
    - 할당된 일정 시간 동안만 CPU를 사용할 수 있음
    - 할당된 시간을 모두 사용한다면 다시 준비 상태가 되고, 실행 도중 입출력장치를 사용하여 입출력장치의 작업이 끝날 때까지 기다려야 한다면 대기 상태가 됨
- 대기 상태
    - 입출력장치의 작업을 기다리는 상태 (입출력 작업만 있는 것은 아니지만 대부분)
    - 입출력 작업을 요청한 프로세스는 입출력장치가 입출력을 끝낼 때까지 기다려야함

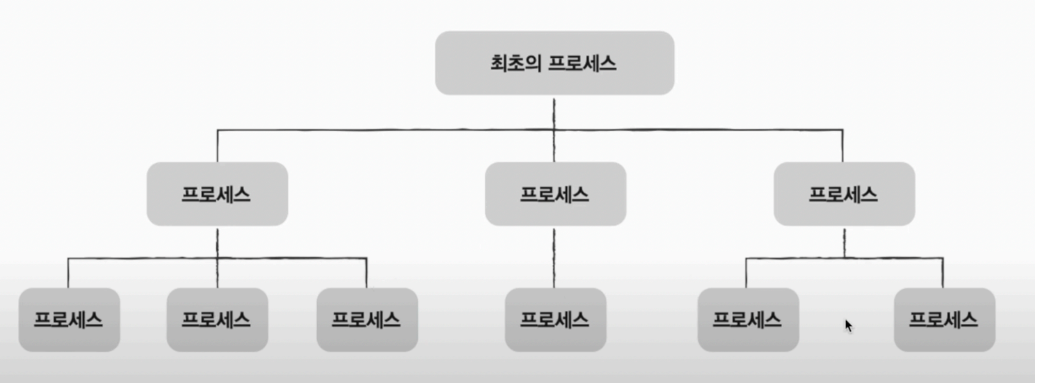
- 입출력 작업이 완료되면 해당 프로세스는 다시 준비 상태로 CPU 할당을 기다림

- `종료 상태`
  - 프로세스가 종료된 상태



프로세스 계층 구조

- 부모 프로세스 : 새 프로세스를 생성한 프로세스
- 자식 프로세스 : 부모 프로세스에 의해 생성된 프로세스
- 부모 프로세스와 자식 프로세스는 다른 프로세스이기에 각기 다른 PID를 가짐
- 일부 운영체제에서는 자식 프로세스의 PCB에 부모 프로세스의 PID인 PPID가 기록되기도함



프로세스 생성 기법

- `fork` : 자신의 복사본을 자식 프로세스로 생성해냄
  - 부모 프로세스의 복사본이기 때문에 부모 프로세스의 자원들 (메모리 내용, 열린 파일의 목록 등의 자삭 프로세스에 상송됨)

- `exec` : 만들어진 복사본은 exec을 통해 자신의 메모리 공간을 다른 프로그램으로 교체함
    - 자신의 메모리 공간을 새로운 프로그램을 덮어쓰는 시스템 호출
- 부모 프로세스와 자식 프로세스 누구도 exec를 호출하지 않는 경우도 있음
- 이 경우 부모 프로세스와 자식 프로세스는 같은 코드를 병행하여 실행하는 프로세스가 됨

스레드

- 프로세스를 구성하는 실행의 흐름 단위
- 하나의 프로세스는 여러 개의 스레드를 가질 수 있음
    - 즉 프로세스를 구성하는 여러 명령어를 동시에 실행할 수 있게 된 것
- 프로세스 내에서 각기 다른 스레드 ID, 프로그램 카운터 값을 비롯한 레지스터 값, 스택으로 구성됨 → 스레드마다 각기 다른 코드를 실행할 수 있음

멀티프로세스와 멀티스레드

멀티프로세스 : 여러 프로세스를 동시에 실행하는 것

멀티스레드 : 여러 스레드로 프로세스를 동시에 실행하는 것

- 프로세스끼리는 기본적으로 자원을 공유하지 않지만, 스레드끼리는 같은 프로세스 내의 자원을 공유한다는 점
- 공유하는 자원과 따로 가지는 자원이 있어, 여러 프로세스를 병행 실행하는 것보다 메모리를 더 효율적으로 사용할 수 있음
- 프로세스의 자원을 공유하기 때문에 서로 협력과 통신에 유리함

- 하나의 스레드에 문제가 생기면 프로세스 전체에 문제가 생길 수 있음
    - 모든 스레드는 프로세스의 자원을 공유하고, 하나의 스레드에 문제가 생기면 다른 스레드도 영향을 받기 때문