Chapter 3. Process (Part 1)

3.1 Process Concept

Process

- 실행중인 프로그램
- 운영체제의 작업의 단위
- 프로세스가 실행되기 위해 필요한 자원

  ㄴ CPU, memory, files, I/O 디바이스

  ㄴ ex) 실행파일(a.out) 이 HDD에 저장되어 있을 때 > 그것을 memory에 로드 > CPU에서 그것을 fetch 하여 실행할 수 있는 상태 > 이 상태가 process

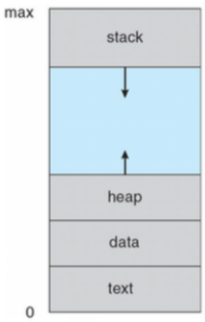
  ㄴ 여기서 끝나지 않고, process는 이제 CPU를 점유해야한다.

Process의 sections

- **Text** section : 실행 가능한 코드를 저장
- **Data** section : 전역 변수를 저장
- **Heap** section : 프로그램 실행동안 메모리를 할당

  ㄴmalloc(C), new(java) 등의 메소드 실행시 이 영역을 사용
- **Stack** section : 함수 호출을 했을 때 사용하는 영역

  ㄴ function parameters, return addresses 등

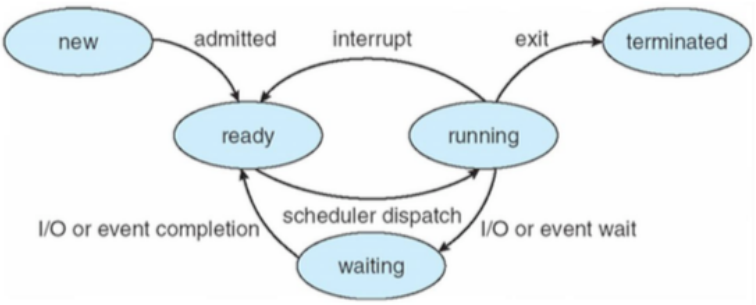


OS가 process를 어떻게 관리하나

a. 프로세스의 생명주기

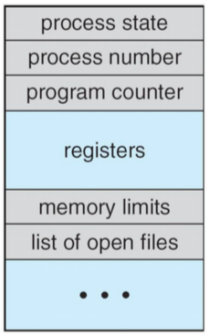
- **New** : 프로세스가 생성
- **Running** : 프로세스가 CPU를 점유해서 명령어들이 실행되는 상태

- **Waiting** : 다른 프로세스가 cpu를 점유하고 있는 상태에서 기다리는 상태
- **Ready** : wait상태에서 이제 cpu를 점유할 준비가 된 상태 → ready queue에 push됨
- **Terminated** : 실행이 끝난 상태



b. PCB(Process Control Block) or TCB(Task Control Block)

- 프로세스가 필요한 정보를 PCB에 저장해 두고, PCB를 가지고 프로세스를 핸들링
- PCB에 저장할 정보
  1. **Process state** : new, ready, running..
  2. **Program counter(PC)** : 실행하는 프로그램의 memory상의 위치
     ㄴ 여기에서 시작하여 +1 씩하면서 다음 명령어를 불러오며 실행
  3. **CPU register** : IR, PC 등
  4. **CPU 스케줄링 정보**
  5. **메모리 관리 정보**
  6. **계정 정보**
  7. **I/O status**



c. Process is

- **single thread** of execution → 한 번에 하나를 실행하는 구조
- 프로세스 여러 개를 한 번에 실행하면 좋겠다. → multitasking, multiprocessing

d. Thread is

- 위에서 말한 thread와는 다른 개념
- lightweight process → 더 가벼운 프로세스, 프로세스 안의 프로세스
- 프로세스 안에서도 여러 개의 thread를 동시에 실행되야 하는 경우가 생김 → multithreading

3.2 Process Scheduling

multiprogramming의 목적

- 동시에 여러 개의 프로세스 실행
- CPU 사용 효율을 높이자

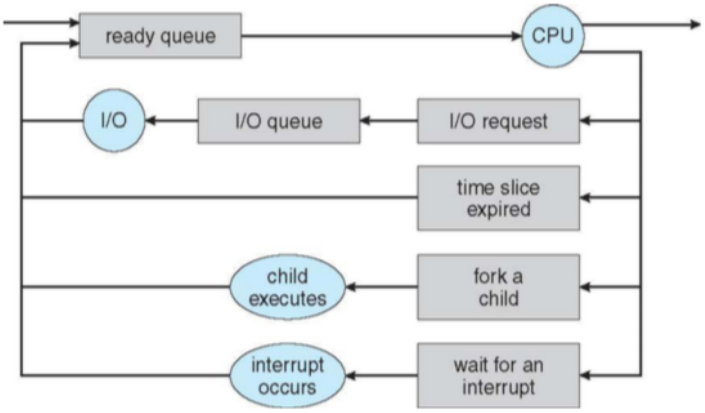
Time Sharing의 목적

- CPU 처리 속도는 아주 빠르다.
- CPU 코어를 프로세스 간 자주 변경 > 사용자가 보기에 여러 프로세스가 동시에 실행되는 것 처럼 보이게 함
- 사실 동시에 실행되는 것은 아니다.

Scheduling Queues

- **ready queue** : 실행을 위해 기다리는 프로세스
- **wait queue** : ready queue로 가기 전에 기다리는 곳

Queueing Diagram

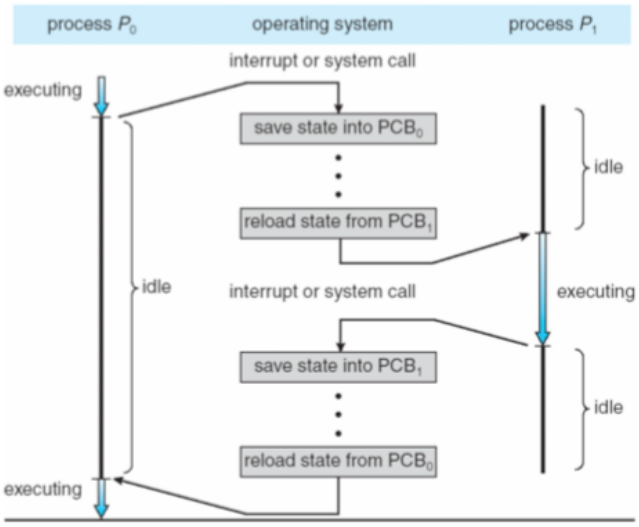


- CPU에서 실행이 끝나고 ready queue로 돌아오는 프로세스도 있지만,
- I/O 요청, time slice, fork 등의 요청 후에 CPU처리를 기다리기 위해 ready queue로 돌아오기도 한다.

Context Switch

- **Context**(문맥) : 프로세스가 사용되고 있는 상태 → 그 정보는 PCB 에 저장되어 있음
- **Context Switch** : 다른 프로세스에게 CPU core를 넘겨주는 것

  ㄴ 현재 프로세스의 context 를 저장 > 새로운 프로세스의 context를 복원 하는 과정을 수행



3.3 Operations on Processes

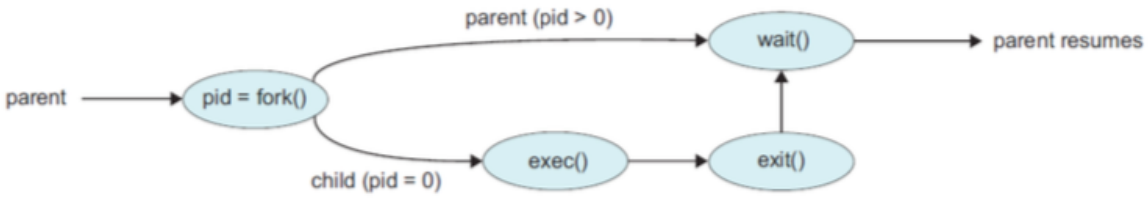
- 운영체제가 process를 생성하고, 종료한다.

프로세스가 새로운 프로세스를 생성한다.

- 생성하는 프로세스 : parent process
- 생성되는 프로세스 : child process
- fork() system call를 사용하여 생성

프로세스 실행의 두 가지 경우

1. parent, child 프로세스가 동시에 실행되는 경우
2. child 프로세스가 실행되는 동안, parent 프로세스는 기다리는 경우



address-space의 두 가지 경우

1. parent와 child 가 같은 기능을 하는 경우 → memory 영역을 복제하여 사용

   ㄴ 각 프로세스의 PCB도 따로 존재

2. parent에서 fork()된 child 가 새로운 프로그램을 로드하는 경우

프로세스 종료

- 마지막 문장을 실행 → return 문
- exit()를 호출하여 강제로 종료

⇒ 종료하기 전, OS는 메모리와 자원을 해제하고 회수 → file, I/O 등

Zombie and Orphan

- **Zombie process** : parent가 child에게 wait()를 호출하지 않고, 신경쓰지 않는 경우 > child는 좀비 프로세스가 된다.
- **Orphan process** : parent가 child에게 wait()를 호출하지 않고, 먼저 종료해 버린 경우 > child는 orphan 프로세스가 된다.

⇒ deamon, background process 를 만들 때 활용