Process 1

상태	완료
생성 일시	@2023년 7월 2일 오후 3:47
수업일	@2023년 7월 2일
유형	정리

프로세스의 개념

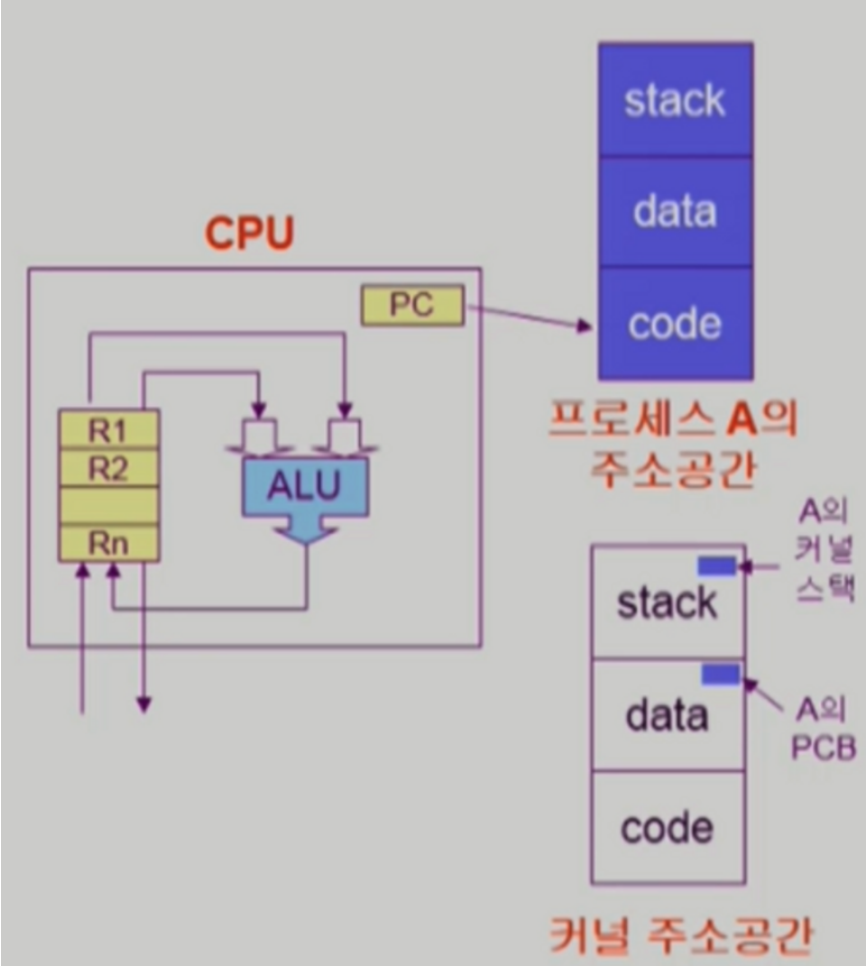
프로세스는 실행 중인 프로그램이다. 디스크에 실행 파일 형태로 존재하던 프로그램이 메모리에 올라가서 실행되기 시작하면 비로소 생명력을 갖는 프로세스가 되는 것이다.

프로세스의 문맥 (Context)

프로세스가 시작해서 종료할 때까지 CPU에서 명령을 한꺼번에 수행하면 좋겠지만, 여러 프로세스가 함께 수행되는 시분할 환경에서는 CPU를 자주 빼앗기고 획득하게 된다. 따라서 CPU를 다시 획득해 명령의 수행을 재개하는 시점이 되면, 이전 CPU 보유 시기에 어느 부분까지 명령을 수행했는지(PC가 어디를 가리키고 있는가) 정확한 상태를 재현할 필요가 있다. 이때 정확한 재현을 위해 필요한 정보가 바로 프로세스의 문맥이다.

즉, 프로세스의 문맥은 그 프로세스의 주소 공간 (코드, 데이터, 스택 상태)를 비롯해 레지스터에 저장된 값, 시스템 콜을 통해 커널에서 수행한 일의 상태, 그 프로세스에 관해 커널이 관리하고 있는 각종 정보 등을 포함하게 된다.

프로세스의 문맥은 3가지로 나눌 수 있다.



😎*하드웨어 문맥*

CPU의 수행 상태를 나타내는 것으로 프로그램 카운터 값과 각종 레지스터에 저장하고 있는 값들을 의미한다. 현재 시점에 프로세스가 명령을 어디까지 수행했는지 판단하는 지표가 된다.

😎*프로세스의 주소 공간*

code, data, stack에 들어 있는 내용이다.

😎*프로세스 관련 커널 자료 구조 (커널 상의 문맥)*

PCB (Process Control Block)

운영 체제가 프로세스에게 CPU나 메모리를 얼마나 줘야 하는지, 보안 상으로 취약한 행위를 하고 있지 않은지 관리하는 역할을 하는데, 이때 프로세스마다 상태를 관리하기 위해 PCB를 둔다.

Kernel Stack

시스템 콜을 하면 PC가 커널의 code(사실상 모든 프로세스가 공유함)를 가리키며 수행되는데, 호출 정보를 커널 스택에 프로세스 별로 스택을 두어서 저장한다.

프로세스의 상태

프로세스는 상태가 변경되며 수행된다.

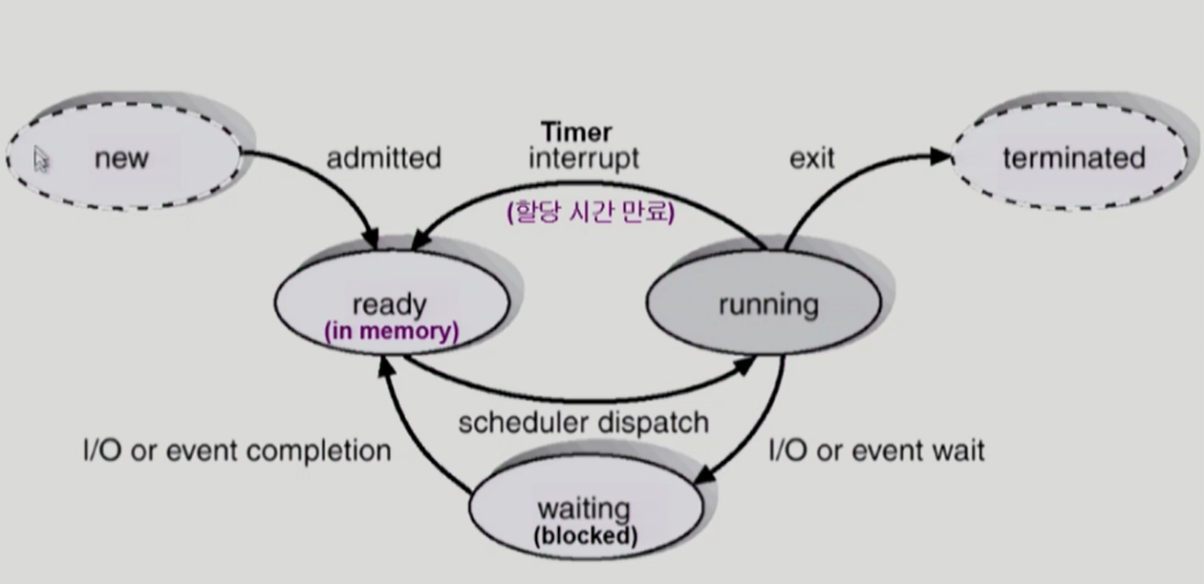
- 실행 (Running)
  - CPU를 잡고 명령을 수행 중인 상태
- 준비 (Ready)
  - 메모리에 프로그램이 올라가 있지만, CPU가 없어서 기다리는 상태

    (CPU만 얻으면 바로 실행 가능한 상태)

    보통 Ready 상태에 있는 프로세스들이 번갈아가면 CPU를 잡았다/놨다 하면서 타임쉐어링을 구현
- 봉쇄 (Blocked, Wait, Sleep)
  - CPU를 할당받더라도 당장 명령을 실행할 수 없는 상태
  - 프로세스 자신이 요청한 이벤트(예 : I/O)가 즉시 만족 되지 않아 이를 기다리는 상태
  - ex) 디스크에서 파일을 읽어와야 하는 경우, 다음 명령이 메모리에 현재 없어서 디스크에서 가져와야할때 : 메모리에 다 올라와 있는 게 아니라고 이전 시간에 설명했음
- 시작 (New)
  - 프로세스가 생성 중인 상태
- 완료 (Terminated)
  - 수행이 끝난 상태

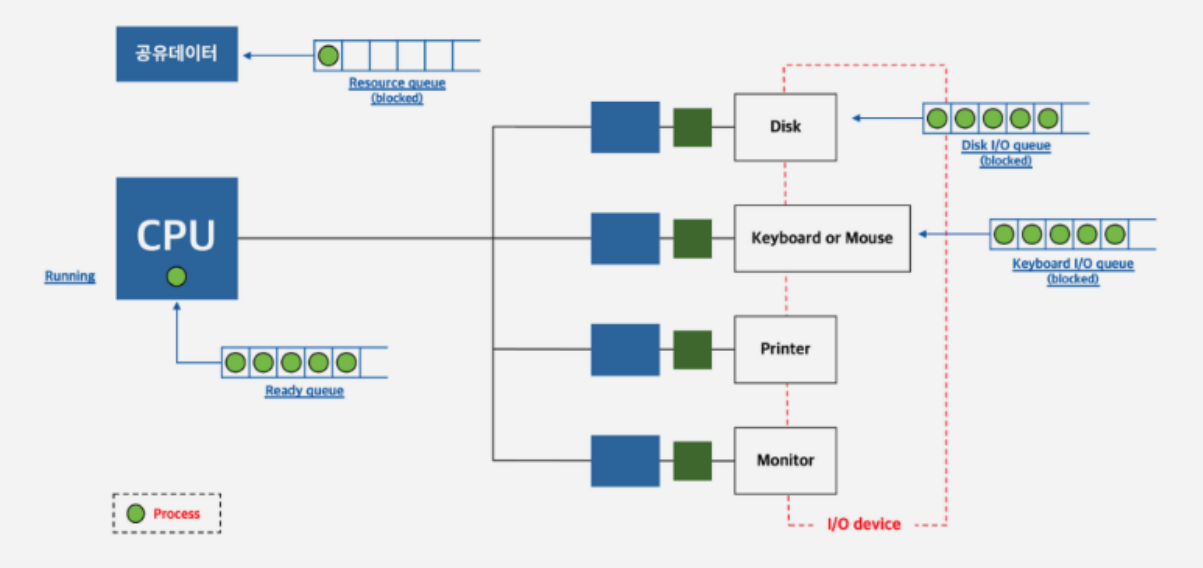
사실 Suspended 상태가 추가로 있지만, 하단에 스케줄러를 설명하면서 같이 언급하려고 한다.

프로세스의 상태도



1. 프로세스가 생성되면 new 상태에서 ready 상태가 된다.
2. ready 상태에서 CPU를 얻으면 running 상태가 된다.
3. CPU 얻은 상태에서 내려 놓는 경우
    a. running → terminated
    b. 본인의 역할을 다하면 종료됨
    c. running → waiting
    d. I/O 같은 오래 걸리는 작업을 하면, CPU를 가지고 있어 봐야 명령을 수행할 수 없으므로 waiting 상태가 된다. (CPU 반납)
    e. running → ready
    f. 할당된 CPU 사용 시간이 끝나면 ready 상태가 된다.
    g.

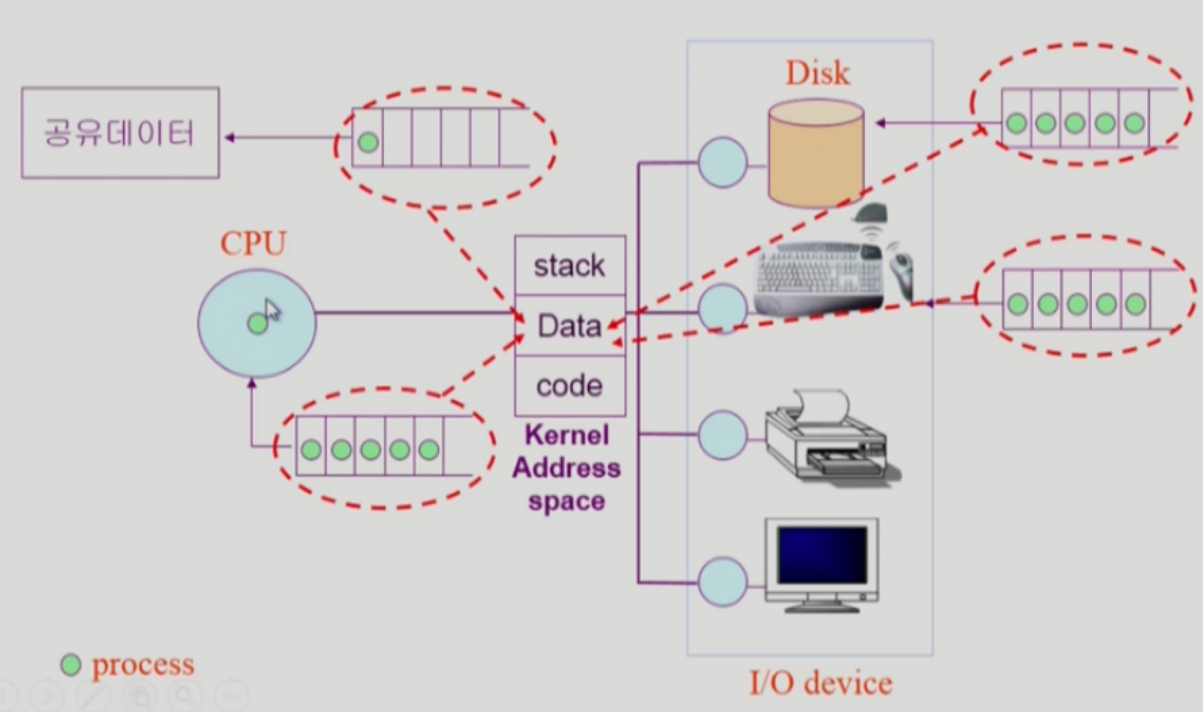
프로세스의 상태 전환 예시



running 상태의 프로세스가 사용자로부터 키보드 입력을 받아서 결과를 확인하고 실행한다고 가정해 보자. 그러면 키보드 I/O 큐에 줄을 서게 된다. 즉, running 상태에서 blocked 상태로 변경된다. 그동안 운영체제는 ready 상태에 있는 다른 프로세스에게 CPU를 전달한다.

키보드 입력을 받으면 키보드 컨트롤러가 CPU에게 인터럽트를 걸어서 알려주고, CPU는 하던 일을 멈추고 운영체제에게 넘어가서 프로세스 상태를 ready로 바꾼다(waiting 상태인). 즉, 키보드 입력이 들어왔기 때문에 CPU를 얻는 권한이 생기며 Ready 큐에 줄을 서게 된다.

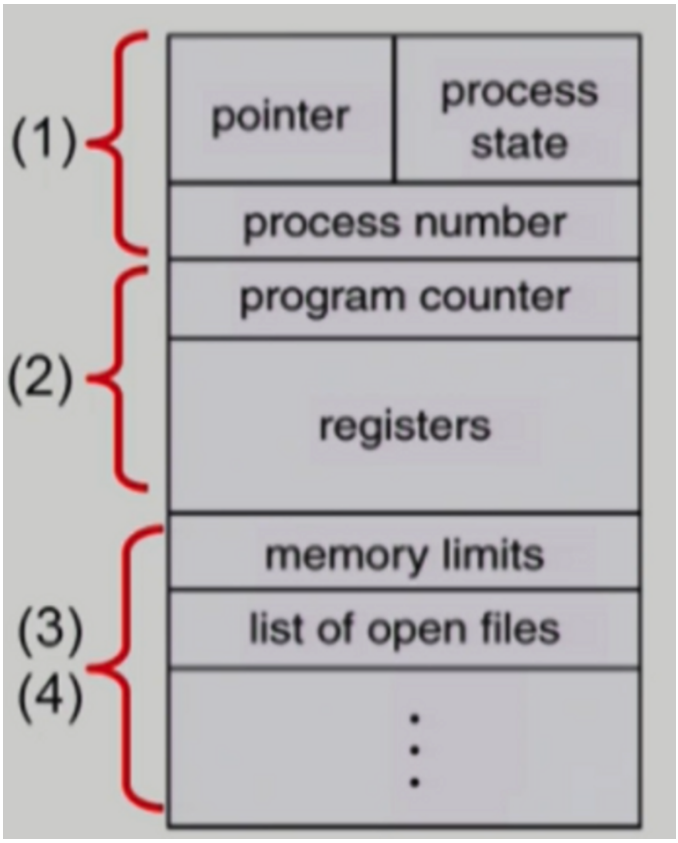
이외에 프로세스는 공유 데이터에 접근하거나(하나의 프로세스가 실행하다가 다른 프로세스가 뺏어 쓰면 안 되기 때문에), 디스크, 키보드, 디스크 등 종류에 따라 그에 맞는 큐에 줄을 서게 된다.



사실 위와 같은 큐들은 모두 커널의 데이터 영역에서 자료구조를 만들어 놓고 프로세스의 상태를 바꿔가면서

레디상태인 애들에게는 CPU를 주고 Blocked 상태인 애들한테는 안주고 하는 것임

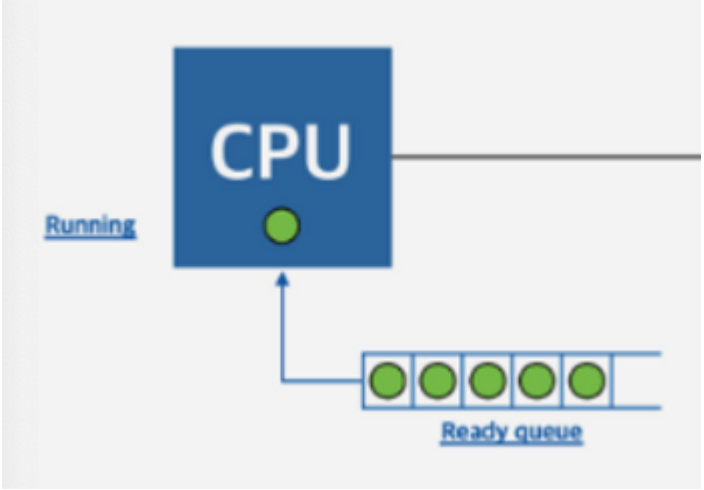
Process Control Block (PCB)



프로세스 제어 블록이란 운영 체제가 시스템 내의 프로세스들을 관리하기 위해 프로세스마다 유지하는 정보들을 담는 커널 내의 자료 구조(커널의 데이터 영역에 위치)를 뜻한다.
PCB는 위 사진과 같은 요소들로 구성되어있다.

(1) OS가 관리 상 사용하는 정보

- 프로세스 상태, 프로세스 ID, CPU 스케줄링 정보, 프로세스 우선 순위 등



사실 큐에 먼저오는 애들한테 CPU가 먼저 할당 되는 것 처럼 표현 되어있지만 사실 뒤에 CPU 스케줄링을 보면

정확하게는 먼저오는 순서대로 처리하는(라운드 로빈과 같은) 것은 아님 우선순위가 높은 순으로 할당을 해줌

(2) CPU 수행 관련 하드웨어 값 (CPU의 어떤 값들을 넣으면서 실행 하고 있었는가)

- 프로그램 카운터, 레지스터 등

  ▼ 레지스터란?

  "레지스터"는 컴퓨터 아키텍처에서 사용되는 용어로, 프로세서 내부에 있는 데이터 저장소를 의미합니다. 레지스터는 프로세서의 동작에 필요한 데이터를 일시적으로 저장하고 처리하는 데 사용됩니다.

  레지스터는 빠른 데이터 접근 속도와 프로세서의 작업 속도 향상을 위해 사용됩니다. 레지스터는 작은 용량을 가지며, 프로세서 내부에 위치하여 데이터에 접근하는 데 걸리는 지연 시간을 최소화합니다.

(3) 메모리 관련

- code, data, stack의 위치 정보 등

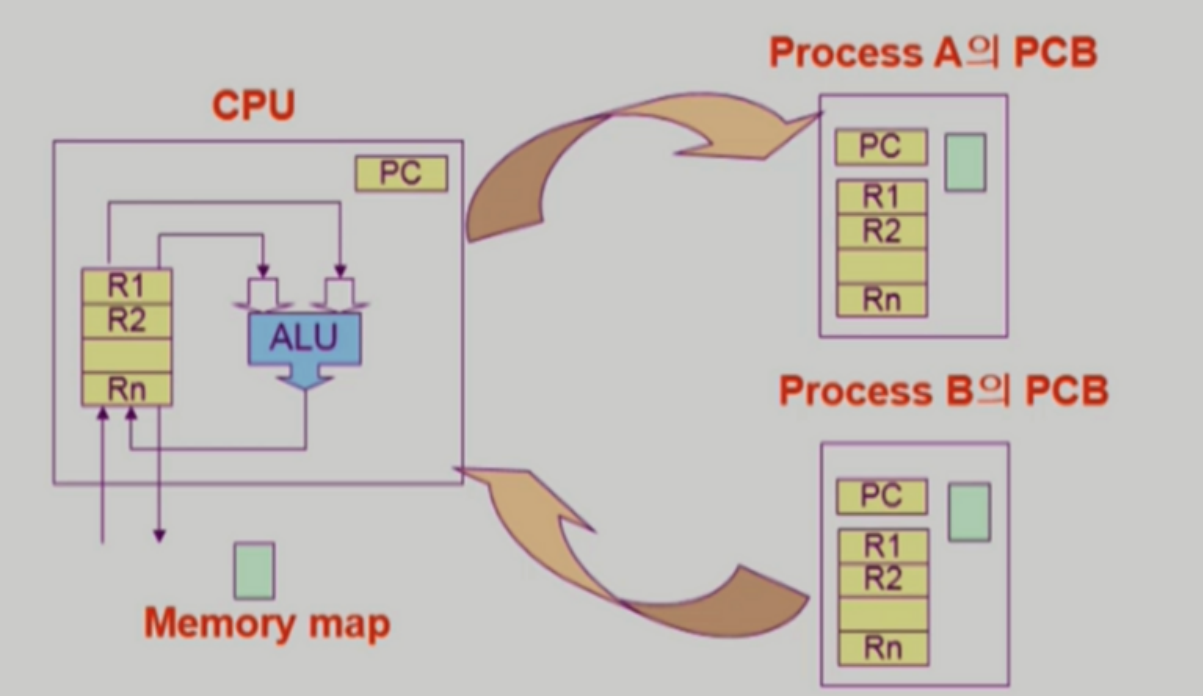
(4) 파일 관련 (리소스 관련)

- 프로세스가 오픈한 파일 정보

문맥 교환 (Context Switch)

문맥 교환이란 하나의 사용자 프로세스로부터 다른 사용자 프로세스로 CPU의 제어권을 넘겨주는 과정을 말한다.

동작 과정

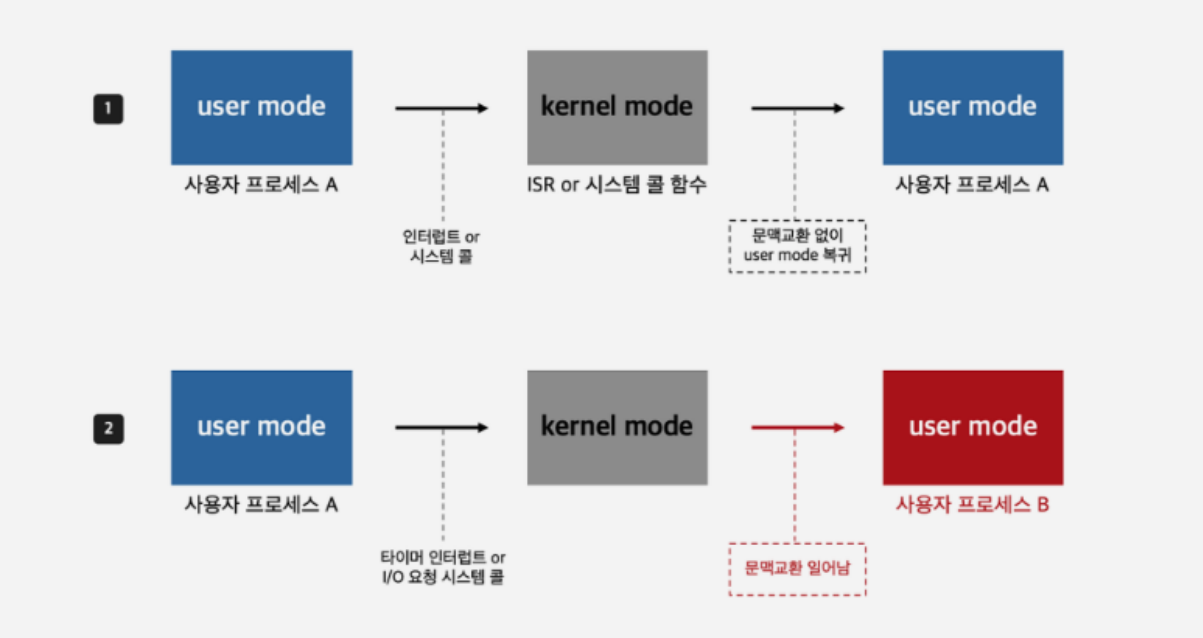


예를 들어 사용자 프로세스가 CPU를 할당 받고 실행되던 중에 타이머 인터럽트가 발생하면 CPU의 제어권은 운영체제에게 넘어가게 된다. 그러면 운영 체제는 타이머 인터럽트 처리 루틴으로 가서 직전까지 수행 중이던 프로세스의 문맥을 저장하고 새롭게 실행 시킬 프로세스에게 CPU의 제어권을 넘긴다. 이 과정에서 원래 수행 중이던 프로세스는 준비 상태로 바뀌고 새롭게 CPU를 할당 받은 프로세스는 실행 상태가 된다.

문맥 교환 중에 원래 CPU를 보유하고 있던 프로세스는 프로그램 카운터 값과 레지스터 값, 메모리 맵 등 프로세스의 문맥을 자신의 PCB에 저장하고, 새롭게 CPU를 할당 받을 프로세스는 예전에 저장했던 자신의 문맥을 PCB로부터 실제 하드웨어로 복원하는 과정을 거친다.

시스템 콜이나 인터럽트가 발생하면 항상 문맥 교환이 일어나는가?

정답은 X다. 아래 그림을 살펴 보자.



1번 경우는 프로세스가 실행 상태일 때 시스템 콜이나 인터럽트가 발생하여 CPU의 제어권이 운영 체제로 넘어와 원래 실행 중이던 프로세스의 업무가 잠시 멈추고 운영체제 커널의 코드가 실행된 경우다.

(보통은 원래 CPU에게 다시 넘겨준다.)

이 경우에도 CPU의 실행 위치 등 프로세스의 문맥 중 일부를 PCB에 저장한다. 하지만, 하나의 프로세스의 실행 모드가 사용자 모드에서 커널 모드로 바뀌는 것이고 CPU를 점유하는 프로세스가 다른 사용자 프로세스로 변경되는 것이 아니므로 문맥 교환이 아니다.

2번 경우는 타이머 인터럽트가 발생하거나 프로세스가 입출력 요청 시스템 콜을 하여 봉쇄 상태(Blocked)에 들어간 경우다. 이 때는 다른 사용자 프로세스에게 CPU의 제어권을 전달하므로 문맥 교환이 발생한다.

1번의 경우에도 CPU 수행 정보등 context의 일부를 PCB에 save해야 하지만 문맥교환을 하는 (2)의 경우 그 부담이 훨씬 큼 (ex. cache memory flush)

보통 문맥교환이 일어나게 되면 cache메모리를 다 지워버려야함 기존 프로세스가 사용하던 캐시를

하지만 1번 같은 경우에는 이렇게 까지 캐시를 다 지울 필요가 없는 것임 (cache memory flush는 상당한 오버헤드임) CPU 컨텍스트 정도만 save 했다가 다시 복원함

💁"오버헤드(overhead)"는 어떤 작업을 수행하기 위해 추가적으로 발생하는 비용이나 부하를 의미합니다

프로세스를 스케줄링하기 위한 큐

Job Queue

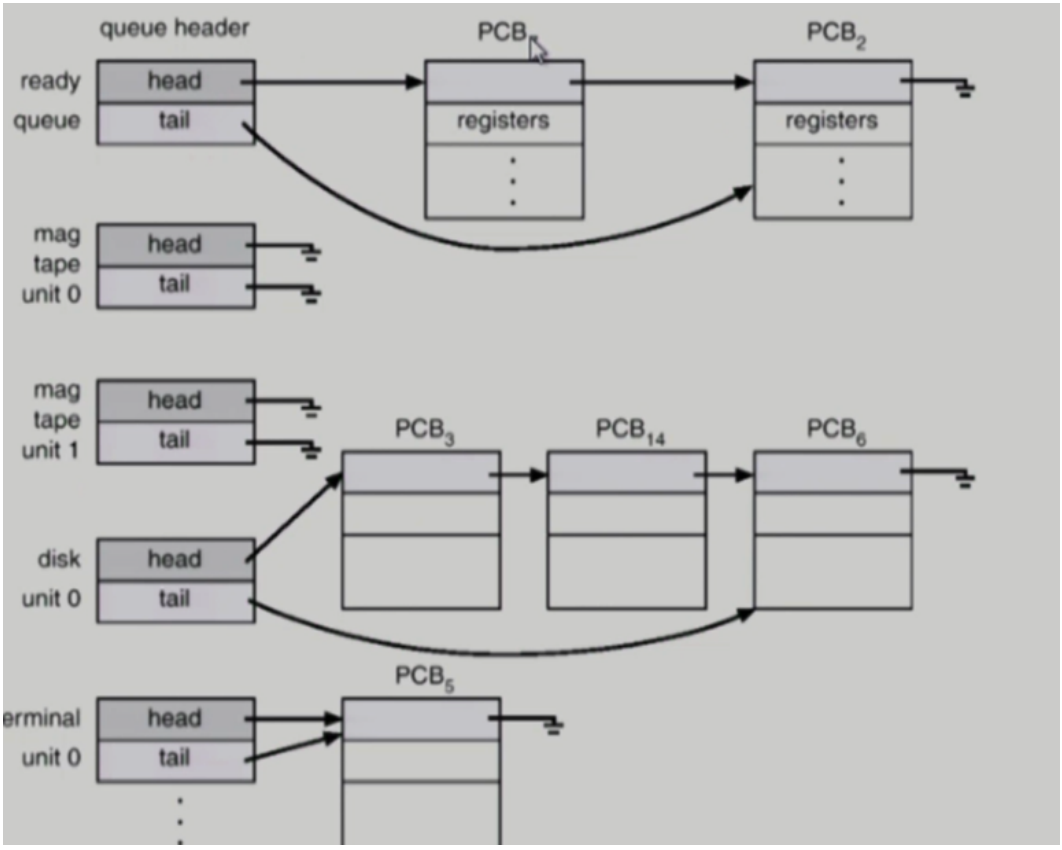
현재 시스템 내에 있는 모든 프로세스 집합이다. (Ready Queue + Device Queues)

Ready Queue

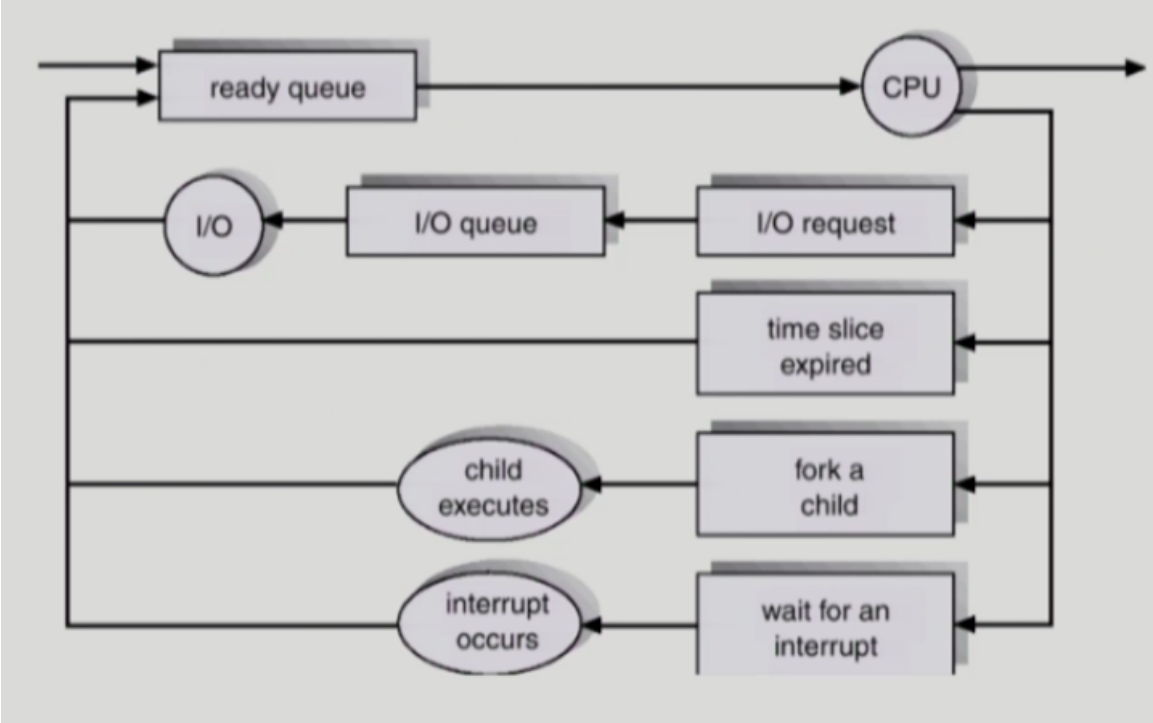
현재 메모리에 있으면서 CPU를 잡아서 실행되기를 기다리는 준비 상태의 프로세스 집합이다.

Device Queues

I/O 디바이스의 처리를 기다리는 봉쇄 상태의 프로세스 집합이다.



위 사진은 앞선 '프로세스의 상태 전환 예시'에서 Queue 그림을 실제 자료 구조에 맞게 그려놓은 것이다.



프로그램이 시작되면 ready queue에 들어가서 줄을 서게 됨 그러다 언젠가 자기 순서가 되면 CPU를 얻을 것임 CPU를 얻은 상태에서 계속 실행되다가 할당 시간이 끝나면 다시 ready queue에 들어가서 줄을 서게 되고

또 CPU를 가지고 있다가 오래 걸리는 작업을 수행할 경우 그 해당하는 작업 큐에 가서 줄 서 있다가 작업이 끝나면 다시 ready queue에 들어가서 줄을 서게 되는 구조임

스케줄러

스케줄러란 어떤 프로세스에게 자원을 할당할지를 결정하는 운영체제 커널의 코드를 지칭한다. 스케줄러에는 3가지 스케줄러가 존재한다.

Long-Term Scheduler (장기 스케줄러 or job scheduler)

- 시작 프로세스 중 어떤 것들을 Ready Queue로 보낼 지 결정한다.
- 프로세스에 메모리를 할당하는 문제에 관여한다. 시작 상태의 프로세스들(NEW 상태에 있는 애들 중) 중 어떠한 프로세스를 Ready Queue에 삽입할 것인지 결정한다.
- degree of Multiprogramming(메모리의 올라가 있는 프로세스의 수)을 제어

  메모리에 프로세스가 너무 많이 올라가도(부분적으로 올라오면 나머지 찾으로 가야하는 시간이 길어질 가능 성) 적게 올라가도(CPU가 놀 가능성) 성능이 안 좋아짐

- Time Sharing System에는 보통 장기 스케줄러가 없다. 무조건 시작 상태의 프로세스는 전부 Ready Queue에 삽입한다.
- 😀 프로세스 입장에서는 천사(나를 메모리로 올려줄)

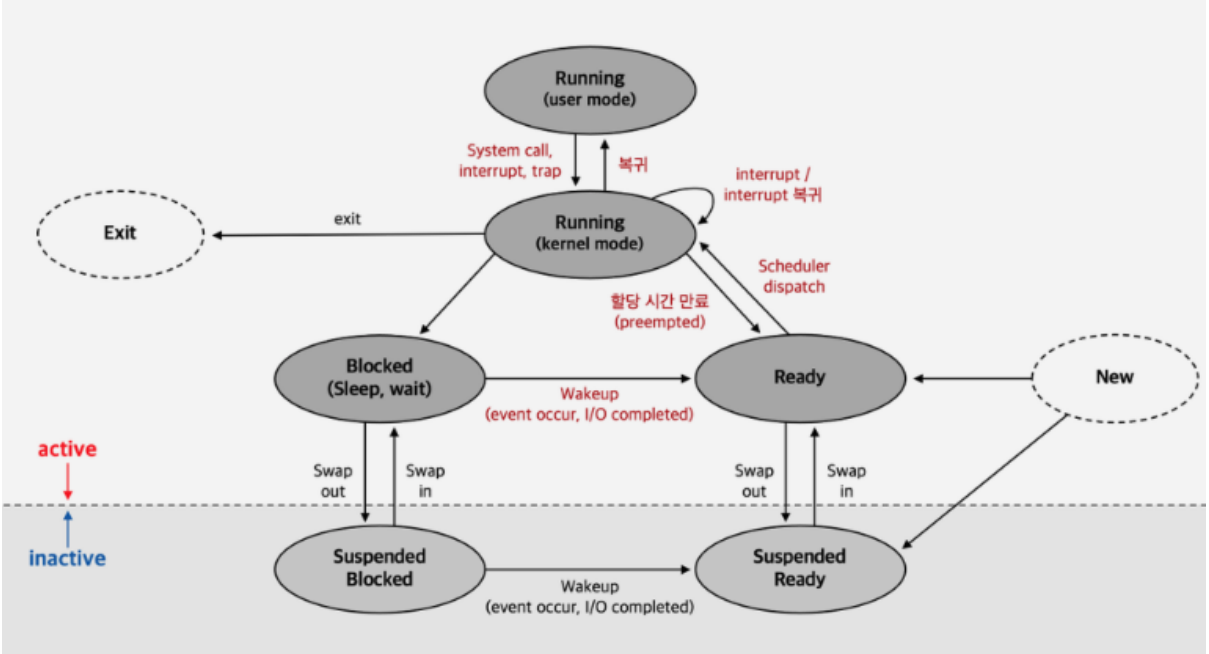
Short-Term Scheduler (단기 스케줄러 or CPU scheduler)

- 어떤 프로세스를 다음 번에 running 할지 결정
- 프로세스에 CPU를 주는 문제
- 충분히 빨라야 한다. (millisecond 단위)

Medium-Term Scheduler (중기 스케줄러 or Swapper)

- 여유 공간 마련을 위해 프로세스를 통째로 메모리에서 디스크의 스왑 영역으로 쫓아낸다. (Swap Out)
  - 메모리에 너무 많은 프로세스가 올라와 있으면
- degree of Multiprogramming(메모리의 올라가 있는 프로세스의 수)을 제어 : 전체적인 시스템의 성능을 좋아지게 만듬
- 주로 봉쇄 상태에 있는 프로세스들을 스왑 아웃하고, 그래도 메모리 공간이 부족하면 준비 큐 대기열 후반부에 있는 프로세스들을 스왑 아웃한다.
- 😂 프로세스 입장에서는 악마(나를 메모리에서 방출할)

현대는 장기 스케줄러가 없고 중기 스케줄러가 있는데 장기와 다르게 ready에 몇 개를 올릴지가 아니라 일단 실행되는 순간 다 메모리에 올려버리고 그다음 필요 없는, 정리가 필요한 애들을 디스크 영역으로 보내 버림



프로세스의 상태 (Suspended 추가)

중기 스케줄러의 등장으로 인해 프로세스의 상태에는 실행, 준비, 봉쇄 외에 하나의 상태가 더 추가된다. 외부적인 이유로 프로세스의 수행이 정지된 상태를 나타내는 중지 (suspended, stopped) 상태가 바로 그것이다(즉 메모리를 통째로 빼앗긴). 중지 상태에 있는 프로세스는 외부에서 재개해야만 다시 활성화될 수 있다. 이 중지 상태의 프로세스는 스왑 아웃된 프로세스라고 생각하면 된다. (메모리를 조금도 보유하지 않음)

Suspended (stopped)

- 외부적인 이유로 프로세스의 수행이 정지된 상태
- 프로세스는 통째로 디스크에 swap out 된다
- (예) 사용자가 프로그램을 일시 정지시킨 경우 (break key)
  시스템이 여러 이유로 프로세스를 잠시 중단시킴
  (메모리에 너무 많은 프로세스가 올라와 있을 때)

중지 상태는 중지 준비 상태와 중지 봉쇄 상태로 세분화할 수 있다. 준비 상태에 있던 프로세스가 중기 스케줄러에 의해 디스크로 스왑 아웃되면 이 프로세스의 상태는 중지 준비 상태가 된다. 이에 비해 봉쇄 상태에 있던 프로세스가 중기 스케줄러에 의해 스왑 아웃되면 이 프로세스의 상태는 중지 봉쇄 상태가 된다. 참고로 중지 봉쇄 상태이던 프로세스가 봉쇄되었던 조건을 만족하게 되면 중지 준비 상태로 바뀐다.(CPU 관점에서 아무 일을 할 수 없고, 메모리를 조금도 보유하고 있지는 않지만, I/O 같은 작업이 진행 중이었다면 끝이 났을 때 PCB에 반영)

Blocked vs Suspended

Blocked: 자신이 요청한 이벤트가 만족하면 Ready

Suspended: 외부(사용자도 포함임 리눅스에서 컨트롤 + Z : Break key 누른 경우)에서 resume 해 주어야 Active