UI/UX 전문가 과정비트캠프

2021년 01월 20일 16회차

장 해 솔

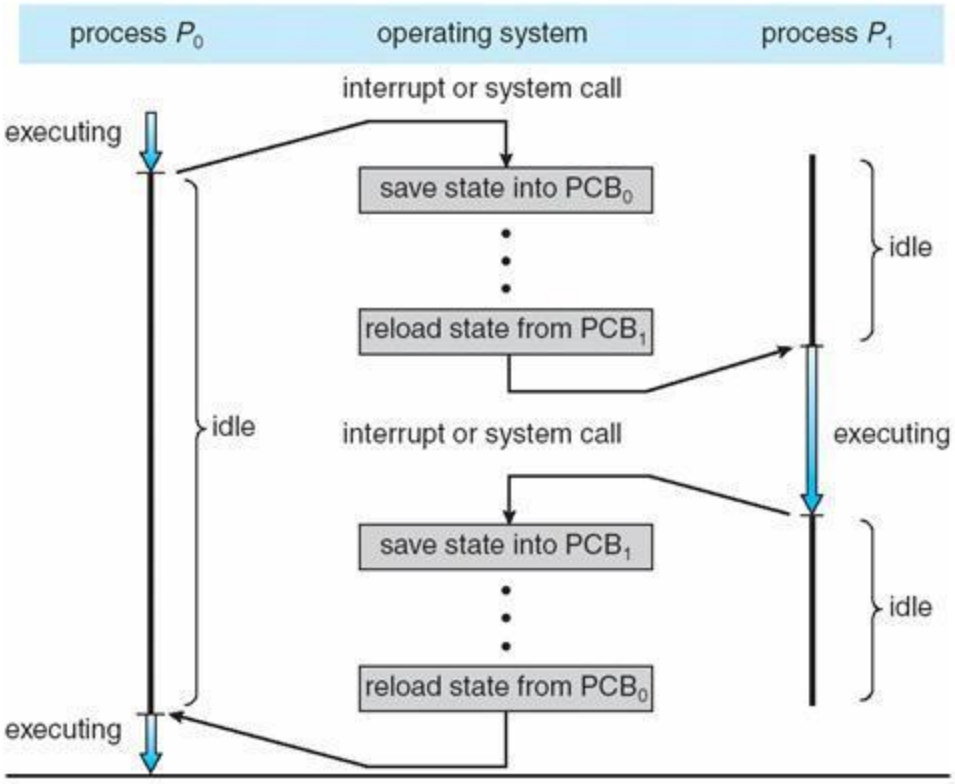
이메일 : wkdgothf@gmail.com

github : https://github.com/legossol

목표

1. Context Switching
2. Spin Lock

1.ContextSwitching



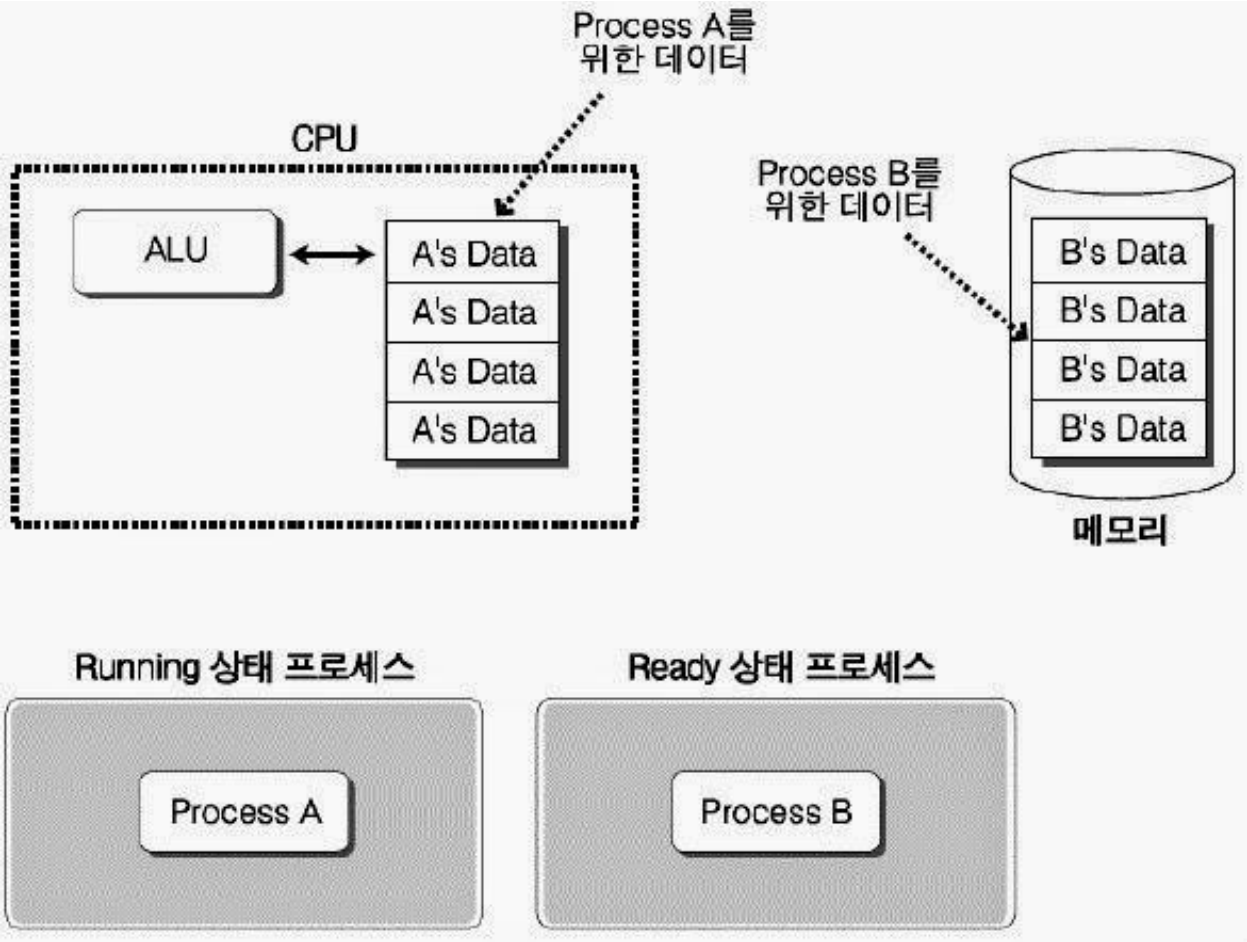
프로세스 P0와 P1이 존재할 때, P0가 CPU를 점유중(excuting)이었고 P1이 대기중(idle)이었는 상태이다가 얼마후에는 P1이 실행이 되고 P0가 대기가 되는 상태가 찾아온다.

이때 P0가 실행중에서 대기로 변하게 될 때는 지금까지 작업해오던 내용을 모두 어딘가에 저장해야하는데 그것이 PCB라는 곳이다.

즉, P0는 PCB에 저장해야하고 P1이 가지고 있던 데이터는 PCB에서 가져와야한다.

이러한 과정 즉, P0와 P1이 서로 **대기<->실행을 번갈아가며 하는 것을** 컨텍스트 스위칭이라고 한다.

CPU가 어떤 프로세스를 실행하고 있는 상태에서 인터럽트에 의해 다음 우선 순위를 가진 프로세스가 실행되어야 할 때 기존의 프로세스 정보들은 PCB에 저장하고 다음 프로세스의 정보를 PCB에서 가져와 교체하는 작업을 컨텍스트 스위칭이라 한다. 이러한 컨텍스트 스위칭을 통해 우리는 멀티 프로세싱, 멀티 스레딩 운영이 가능하다.



여기서 PCB란 Process Control Block을 말하며 프로세스의 정보를 저장하는 Block을 의미. 프로세스가 생성되면 같이 생성되며 종료시 삭제된다.

그렇다면 컨텍스트 스위칭은 프로세스만 하는 것인가?

아니다. 쓰레드도 컨텍스트 스위칭을 한다.
어제자 PDF에서 알아 보았듯이 프로세스간 에는 마치 벽이 있는 것처럼 정보를 공유할 수 없다. 하지만 한 프로세스 내에서는 정보 공유가 가능하기에 쓰레드는 정보 공유가 가능하다.

그렇기에 켄텍스트 스위칭에서의 차이를 보자면 쓰레드는 정보를 공유하는 것이 가능하기에 속도가 프로세스 보다 빠르며 또 한가지는 저장되어 있던 정보를 가지고 옴에 있어 프로세스보다 가벼운 것들을 가져오기에 더욱 빠르다.

2. Spin Lock

- Lock을 얻을 수 없다면, 계속해서 Lock을 확인하며 얻을 때까지 기다린다. 이른바 바쁘게 기다리는 busy waiting이다.
- 바쁘게 기다린다는 것은 무한 루프를 돌면서 최대한 다른 스레드에게 CPU를 양보하지 않는 것이다.

- **Lock이 곧 사용가능해질 경우 Context Switching을 줄여 CPU의 부담을 덜어준다.**
  하지만, 만약 어떤 스레드가 Lock을 오랫동안 유지한다면 오히려 CPU 시간을 많이 소모할 가능성이 있다.

- 하나의 CPU나 하나의 코어만 있는 경우에는 유용하지 않다.
  그 이유는 만약 다른 스레드가 Lock을 가지고 있고 그 스레드가 Lock을 풀어주려면 싱글 CPU 사용률 100%를 만드는 상황이 발생하므로 주의해야한다.
  스핀락은 기본적으로 무한 for루프를 돌면서 lock을 기다리므로 하나의 스레드가 lock을 오랫동안 가지고 있다면, 다른 Blocking된 스레드는 busy waiting을 하므로 CPU를 쓸데없이 낭비하게 된다.

장점 : Spin Lock을 잘 사용하면 context switching을 줄여 효율을 높일 수다.
무한 루프를 돌기 보다는 일정 시간 **lock을 얻을 수 없다면 잠시 sleep하는 back off 알고리즘을 사용하는 것이 훨씬 좋다.**

뮤텍스(Mutex)도 락이 있는데 이는 Mutualy Exclusive라는 것이다.
결국 이것도 세마포어이며, 특별히 카운트가 1인 락으로 봐도 무방하다.

이와 반대로 스핀 락은 자지 않고 계속해서 버틴다고 볼 수 있다.