Lock, Semaphores

Critical Section(임계영역) 설정이 왜 필요한가??

- 예시
    - Process A는 cnt += 500을 하고 Process B는 cnt -= 500을 한다고 했을 때

data 영역의 cnt = 1000일때 A와 B가 동시에 접근 한다면 둘 다 cnt=1000인 상태에서 자신들의 할일을 처리함

- B가 A보다 나중에 끝난다고 했을 때 cnt는 500인 상태로 끝남(원래 1000을 유지해야됨)
- 따라서 data 영역의 cnt는 임계영역 설정이 필요함!!

Critical Section 설정 조건!!

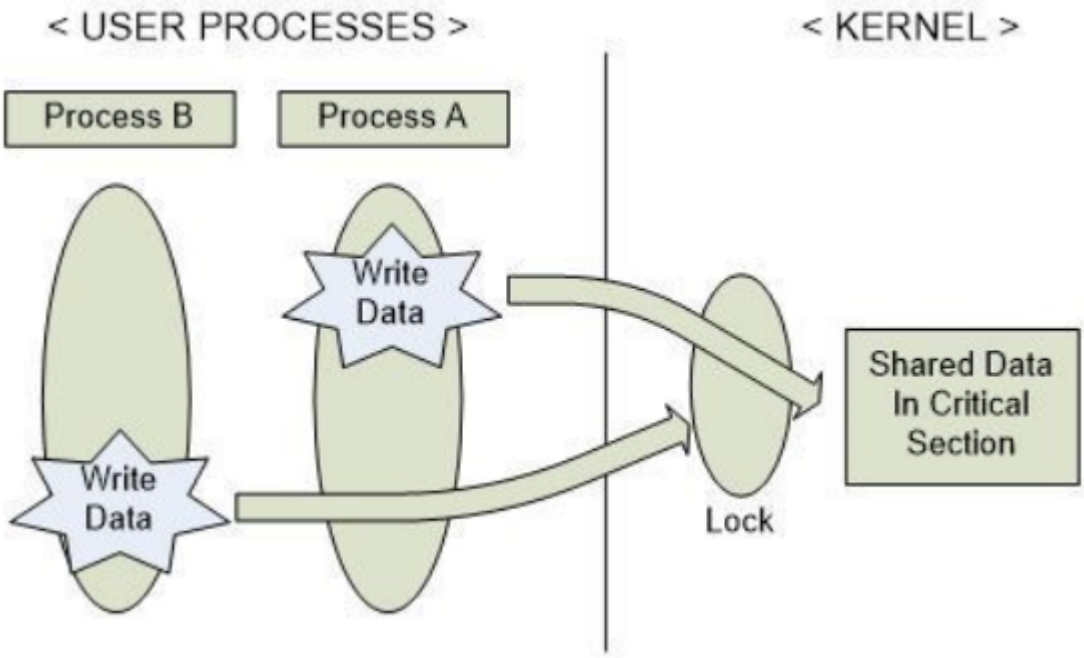
1. 상호배제(Mutual Exclusion)을 만족해야함!!
    - 상호배제란??
        - 특정 프로세스가 공유자원을 사용하고 있을 경우 다른 프로세스가 해당 공유 자원을 사용하지 못하게 제어하는 기법
2. 진행(Progress)을 만족해야함!!
    - 진행이란??
        - 임계 영역 안에서 실행하고 있는 프로세스가 없는 경우, 임계 영역을 실행하고자 하는 프로세스는 반드시 임계 영역을 실행할 수 있어야 함
3. 한정된 대기(Bounded waiting)을 만족해야함!!
    - 한정된 대기란??
        - 임계 영역을 요청한 프로세스는 무한히 대기하면 안된다. 즉, 제한된 대기 시간을 가져야 한다.

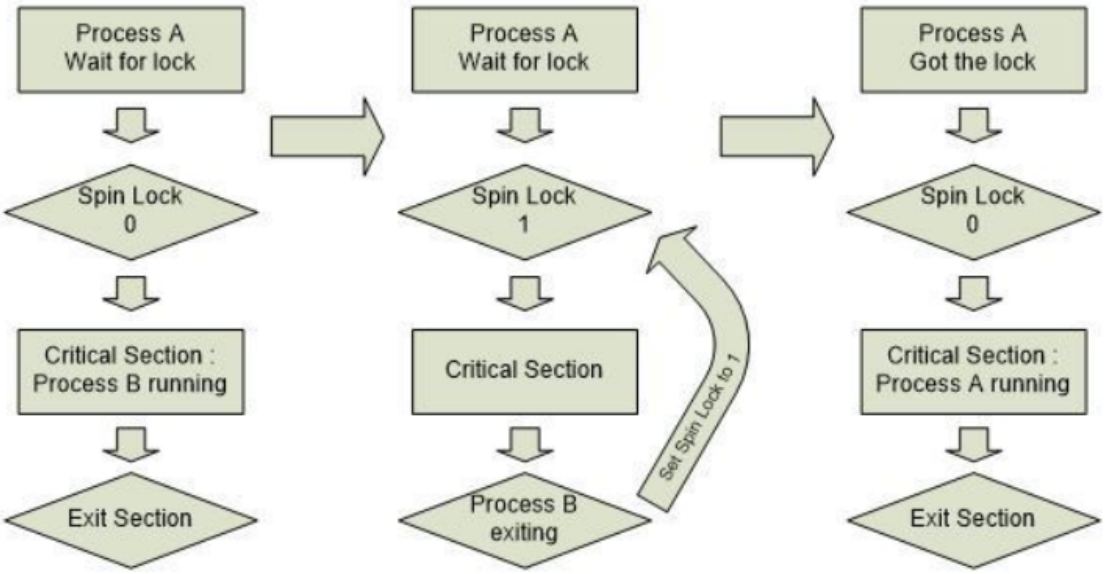
3가지 조건을 모두 만족해야만 임계영역문제를 해결할 수 있는 해결책이라고 볼수 있다

Lock과 Semaphores는 위 3가지 조건을 모두 만족하는가??

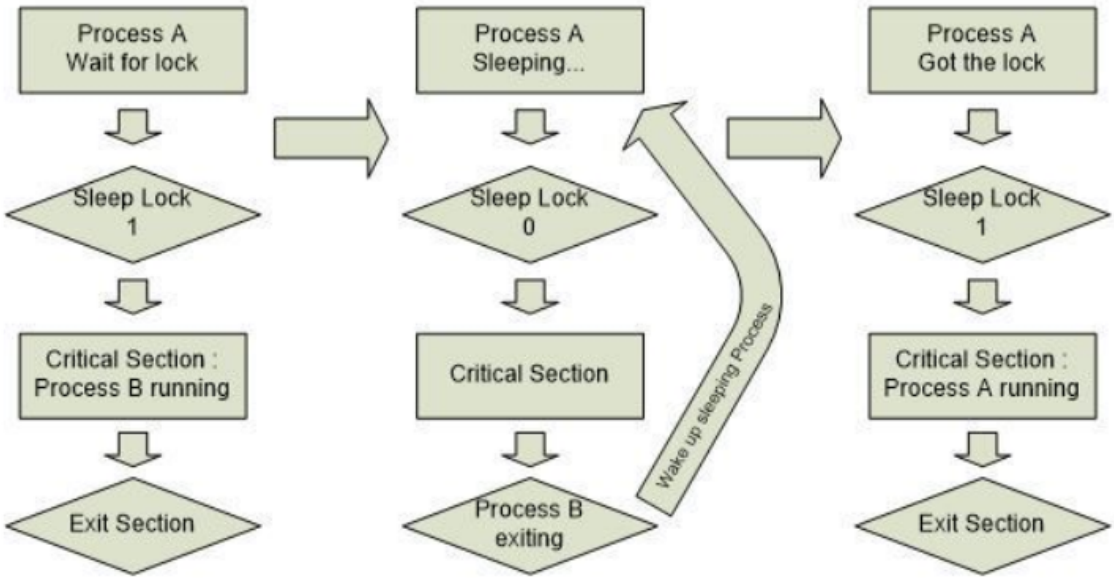
Lock

- read와 write를 구분하지 못한다는 문제점을 가진 상호배제의 해결방안으로 kernel 자체를 하나의 커다란 critical section으로 보자는 생각에서 탄생
-



- lock이란 말 그대로 남이 접근하지 못하도록 잠궈 둔다는 의미로 하나의 프로세스가 공유데이터에 접근하는 동안 다른 프로세스가 공유데이터에 접근하지 못하도록 입구를 잠궈 두는 것으로 lock에는 spin lock과 sleep lock이 있다
- Spin Lock
  - 

- Sleep Lock



- Lock의 문제점
  - 프로세스A와 프로세스B가 동시에 lock=0인 것을 read했을 때, B가 먼저 lock=1을 write하고 B는 read만을 한 채 더 이상 작업이 진행되지 않을 경우, B는 context switching이 발생하게 됨, 그 다음 작업이 수행될 때는 read작업을 다시 하지 않는다, 그러면 context switching이 발생하기 이전 lock=1이었던 것만을 기억한 채 작업을 수행하기 때문에 현재 lock=0이라도 데이터에 접근!!
  - 간단한 해결방법은 read와 write를 하나의 명령어로 만드는 방법

뮤텍스(Mutex, MUTual EXclusion)

- 다중 프로세스의 공유 리소스에 대한 접근을 조율하기 위해 사용되며 Lock과 Unlock 2가지 값만을 가진다.
- 동시에 하나의 스레드에 대해서만 작업이 가능하며 해당 스레드는 락에 대한 소유권을 가지고 직접 락을 해제해야 한다.
- 뮤텍스는 프로세스의 범위를 가지는 객체이므로 뮤텍스를 생성한 프로세스가 종료되면 자동으로 정리된다.

세마포어(Semaphore)

- 동시에 여러 개의 공유 자원 공간을 가진, 여러 개의 카운트 값을 가지는 뮤텍스라고 생각
- 접근할 수 있는 스레드 개수를 정하고 그 개수만큼의 스레드를 관리
- 카운트 수가 1인 세마포어는 바이너리 세마포어(Binary Semaphore) = 뮤텍스
- 카운트 수가 여러 개면 카운팅 세마포어(Counting Semaphore)
- 은행창구로 예를 들 수 있다