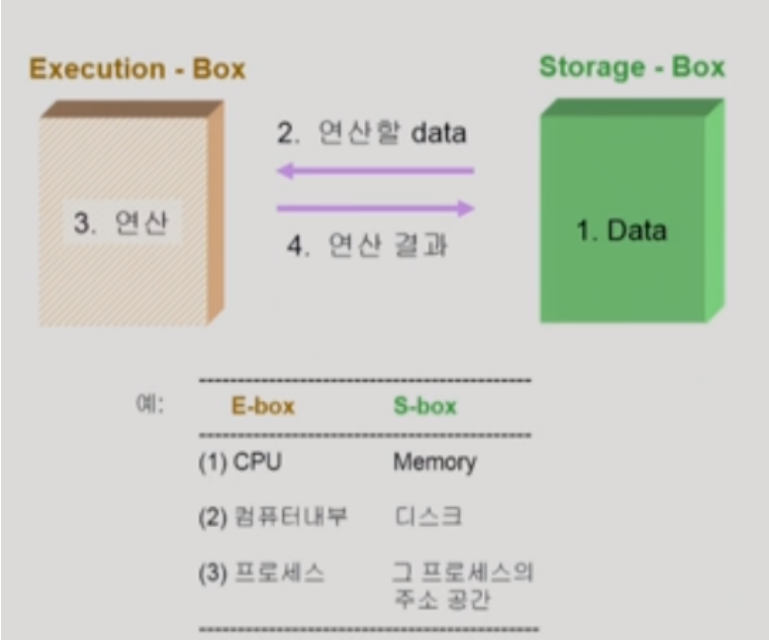
6. Process Synchronization 1

속성

1. 데이터 접근

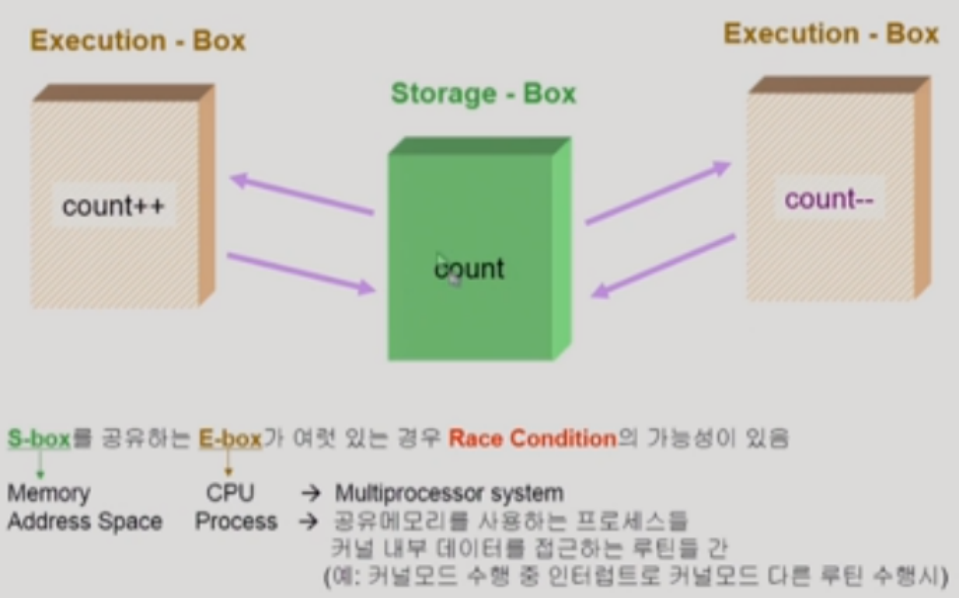


2. Race condition

✔️ 여러 프로세스들이 동시에 공유 데이터를 접근하는 상황

✔️ 테이터의 최종 연산 결과는 마지막에 그 데이터를 다룬 프로세스에 따라 달라짐

- s-box를 공유하는 E-box가 여럿 있는 경우 Race condition의 가능성이 있다. (멀티 프로세서 시스템)
- A가 데이터를 증가, B가 데이터를 감소 시킨다면?

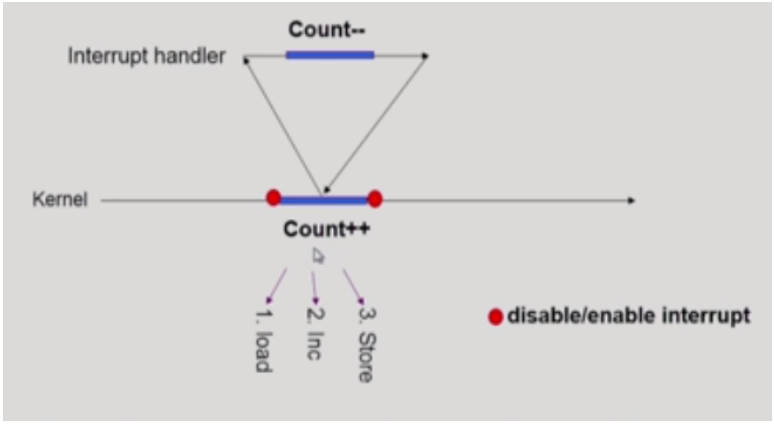


- 커널의 코드가 실행되는 경우 커널의 데이터 변수를 바꿀려고 시도함 * n
- 커널의 코드가 실행되다가 인터럽트가 들어오면 역시 커널의 코드가 실행되기 때문에 문제가 됨.

단, 서로 다른 접근할 때 커널 데이터를 건드릴 때만 race condition이 일어난다. 각자의 주소 공간에 있는 데이터를 건드리면 문제 없다.

3. OS에서 Race condition은 언제 발생하는가?

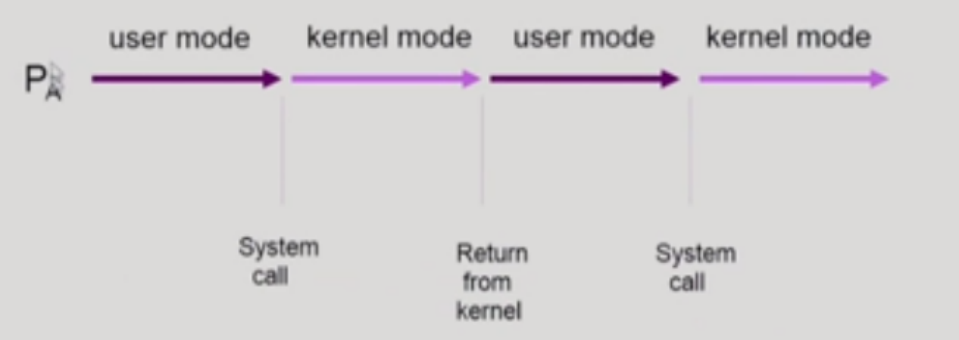
1. 커널 수행 중 인터럽트 발생

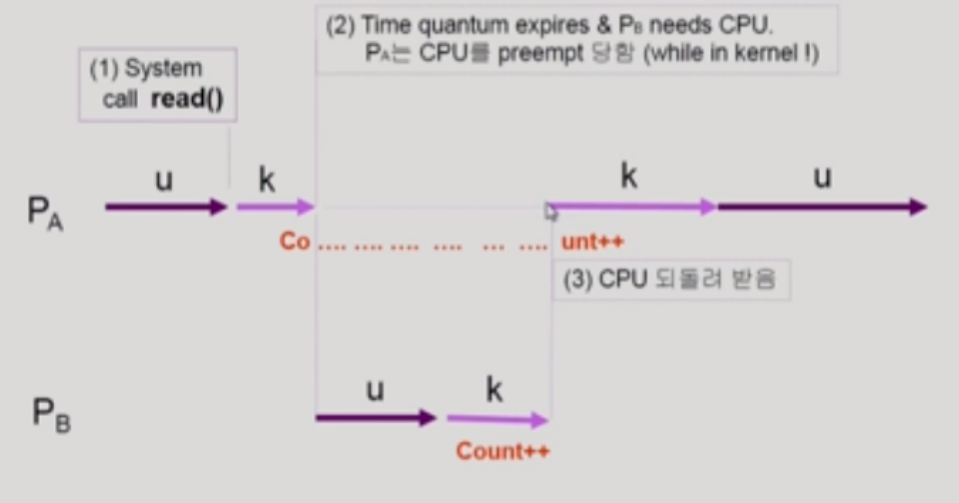


카운트라는 메모리 변수를 1 증가시키기 위해 메모리 변수 값을 CPU 안의 레지스터 로 불르고, +1, 레지스터 값을 메모리의 변수 위치에 넣는 과정.

- 만약 중간에 인터럽트가 들어오게 되면 앞의 과정을 멈추고 인터럽트가 실행되는데 여기서 -1을 하고 작업을 마무리하면 앞의 과정에서 +1을 해도 데이터가 원하는 데로 바뀌지 않는다.
- 인터럽트가 들어와도 인터럽트를 처리하지 않으면 race condition 문제를 해결 한다.

2. Process가 system call을 하여 kernel mode로수행 중인데 context switch가 일어나는 경우

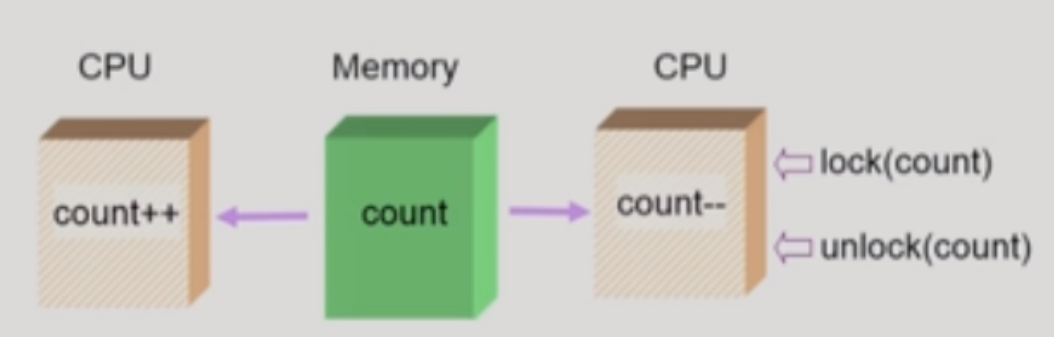




위의 과정에서 count ++2 가 될 것 같지만 count ++1 이 된다. Pa의 count 변수값을 계속해서 가져가기 때문이다.

- 커널 모드에서 수행 중일 때는 CPU를 `preempt`하지 않음. 커널모드에서 사용자 모드로 돌아갈 때 `preempt`

3. Multiprocessor에서 shared memory 내의 kernel data

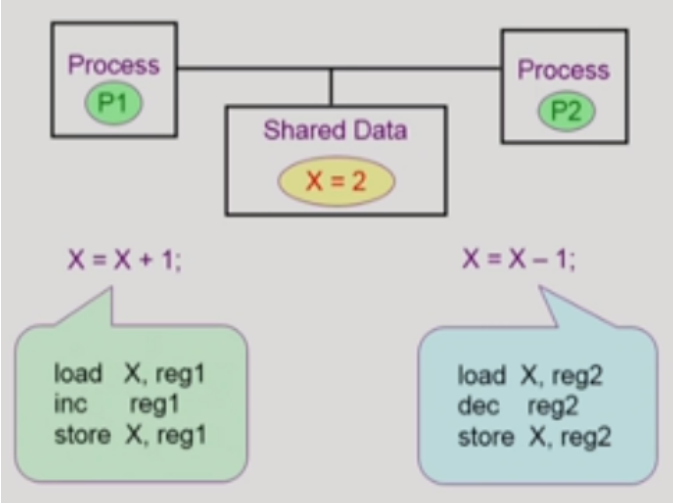


- 어떤 CPU가 마지막으로 count를 store했는가? race condition
  1. 한번에 하나의 CPU만이 커널에 들어갈 수 있게 하는 방법
  2. 커널 내부에 있는 각 공유 데이터에 접근할 때마다 그 데이터에 대한 lock/unlock을 하는 방법

4. Process Synchronization 문제

- 공유 데이터(shared data)의 동시 접근(concurrent access)은 데이터의 불일치 문제(inconsistency)를 발생시킬 수 있다.
- 일관성(consistency) 유지를 위해서는 협력 프로세스(cooperating process) 간의 실행 순서(orderly execution)를 정해주는 메커니즘 필요

❗ race condition을 막기 위해서는 concurrent process는 동기화(synchronize)되어야 한다.



5. The Critical-Section Problem

- n개의 프로세스가 공유 데이터를 동시에 사용하길 원하는 경우
- 각 프로세스의 code segment에는 공유 데이터를 접근하는 코드인 critical section이 존재
- 하나의 프로세스가 critical section에 들어갈 수 없어야 한다!