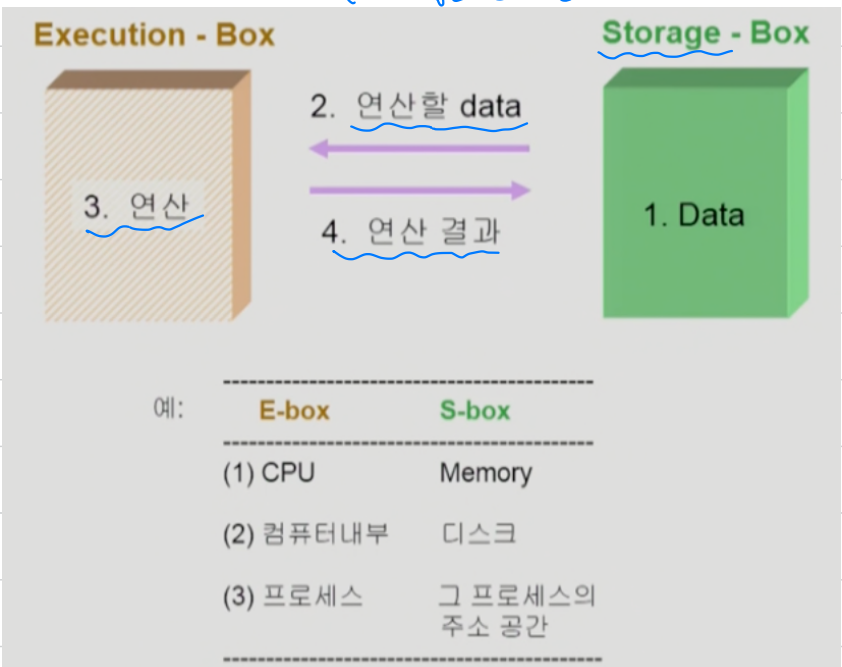
데이터의 접근

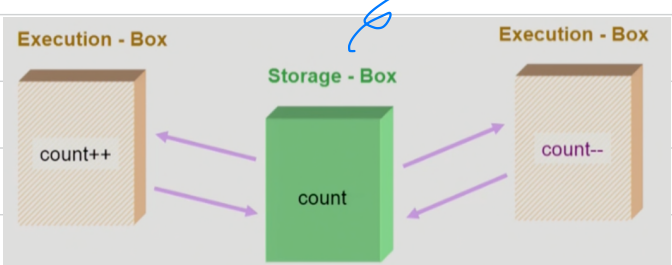
→ 컴퓨터 시스템 안에서 데이터가 접근 되는 패턴



예:	E-box	S-box
(1)	CPU	Memory
(2)	컴퓨터내부	디스크
(3)	프로세스	그 프로세스의 주소 공간

Race Condition

→ 데이터를 동시에 접근하려 할때 ⇒ race condition



S-box를 공유하는 E-box가 여러 있는 경우 Race Condition의 가능성이 있음.

Memory Address Space

CPU space → Multiprocessor system → 한 프로세스가 공유 메모리 사용 중인데 다른 프로세스가 접근

→ 공유 메모리를 사용하는 프로세스들

커널 내부 데이터를 접근하는 루틴들 간 → 인터럽트도 커널 코드임.

(예: 커널모드 수행 중 인터럽트로 커널모드 다른 루틴 수행 시)

(커널의 데이터 접근)

프로세스가 직접 처리하지 못하는 부분이 있어 시스템콜을 써서 커널의 코드가 실행중인데 CPU를 뺏겨 다른 프로세스로 넘어 갔는데 이 프로세스가 또 시스템콜을 통해 커널의 데이터에 접근 한다면 Race-Condition 발생 가능

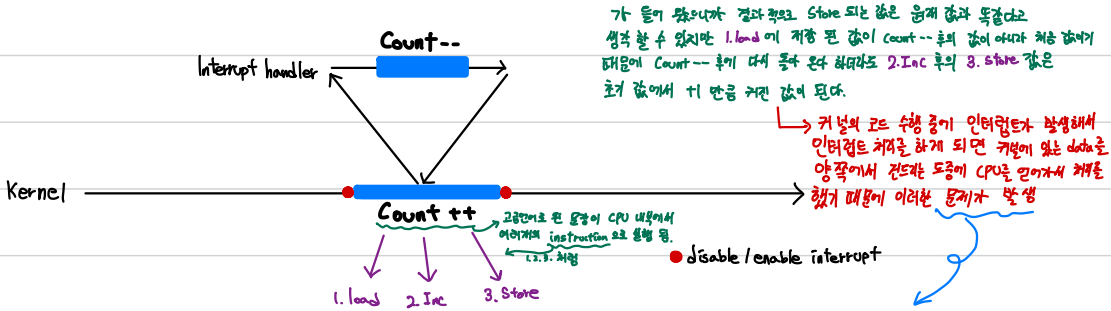
OS에서 race condition은 언제 발생하는가?

1. Kernel 수행 중 인터럽트 발생 시

2. Process가 system call을 하여 Kernel mode로 수행 중인데 Context switch가 일어나는 경우

3. Multiprocessor에서 Shared memory 내의 Kernel data

OS 에서의 race condition (1/3) (interrupt handler vs Kernel)



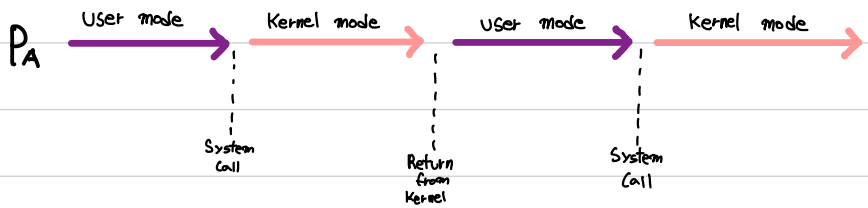
※ Count++ 즉, 커널이 interrupt handler로 Count-- 가 들어 왔으니까 결과적으로 Store 되는 값은 문제 없과 독립적으로 생각 할 수 있지만 1.load 에 저장 된 값이 Count-- 후의 값이 아니라 처음 값이기 때문에 Count-- 후에 다시 돌아 온다 해도 2.Inc 후의 3. Store 값은 초기 값에서 +1 만큼 커진 값이 된다.

→ 커널의 코드 수행 중에 인터럽트가 발생해서 인터럽트 처리를 하게 되면 커널에 있는 data를 엉뚱하게 건드려 문제가 CPU를 빼앗겨서 처리를 했기 때문에 이러한 문제가 발생

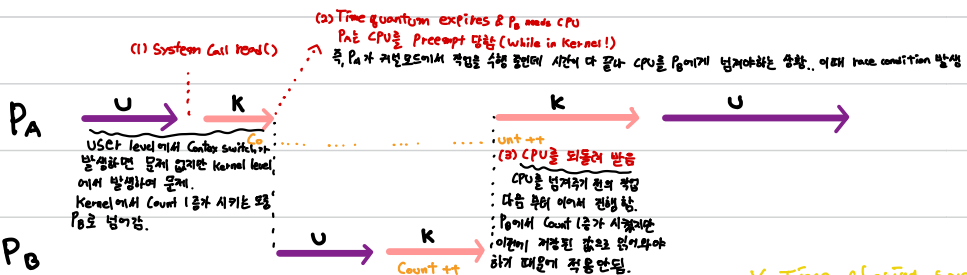
※ 커널모드 running 중 interrupt가 발생하여 인터럽트 처리 루틴이 수행
→ 양쪽 다 커널코드이므로 Kernel address space 공유

문제를 해결하기 위해 커널의 중요한 변수 값을 건드리는 중에는 interrupt가 들어와도 바로 interrupt 처리 루틴으로 넘어가는 것이 아니라 진행중인 작업이 끝날 때까지 interrupt 처리를 하지 않음.

OS에서의 race condition (213) Preempt a process running in kernel?



* 두 프로세스의 address space 간에는 data sharing 이 없음
** 그러나 system call을 하는 동안에는 kernel address space의 data를 access하게 됨 (share)
*** 이 작업 중간에 CPU를 preempt 해버리면 race condition 발생

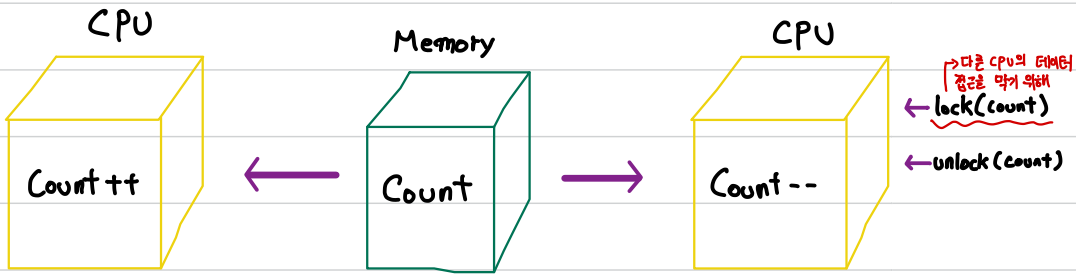


*** 해결책: 커널 모드에서 수행 중일 때는 CPU를 preempt 하지 않음
커널 모드에서 사용자 모드로 돌아갈 때 preempt

* Time sharing system 이라는 것은 real time system 이 아니다. 조금 시간이 더 걸리더라도 시스템이 문제가 생기는 게 아니기 때문에 오히려 이런 문제를 쉽게 해결 가능.

OS 에서의 race condition (3/3) multiprocessor

※ 자주 등장하는 경우는 아니지만 위의 두가지 경우로는 해결이 안됨.



어떤 CPU가 마지막으로 count를 store 했는가? → race condition

multiprocessor의 경우 interrupt enable/disable로 해결되지 않음

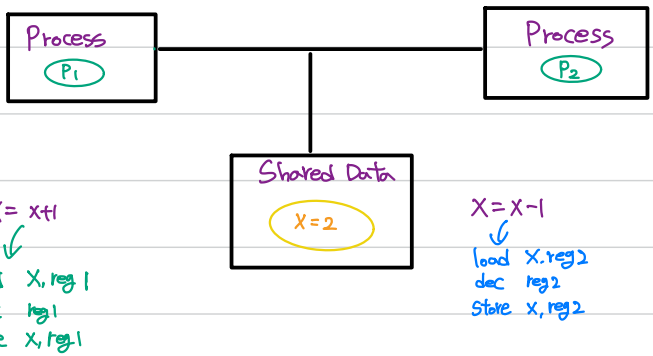
(방법1) 한 번에 하나의 CPU 만이 커널에 들어갈 수 있게 하는 방법

(방법2) 커널 내부에 있는 각 공유 데이터에 접근할 때마다 그 데이터에 대한 lock / unlock을 하는 방법

Process Synchronization 문제

- 공유 데이터(shared data)의 동시 접근(concurrent access)은 데이터의 불일치 문제(inconsistency)를 발생 시킬 수 있다.
- 일관성(consistency) 유지를 위해서는 협력 프로세스(cooperating process)간의 실행 순서(orderly execution)를 정해주는 메커니즘 필요.
- Race Condition
  - v 여러 프로세스들이 동시에 공유 데이터를 접근하는 상황
  - v 데이터의 최종 연산 결과는 마지막에 그 데이터를 다룬 프로세스에 따라 달라짐.
- race condition을 막기 위해서는 concurrent process는 동기화(synchronize) 되어야 한다.

Example of a Race Condition



※ 사용자 프로세스 P1 수행중 timer interrupt가 발생해서 context switch가 일어나서 P2가 CPU를 잡으면?

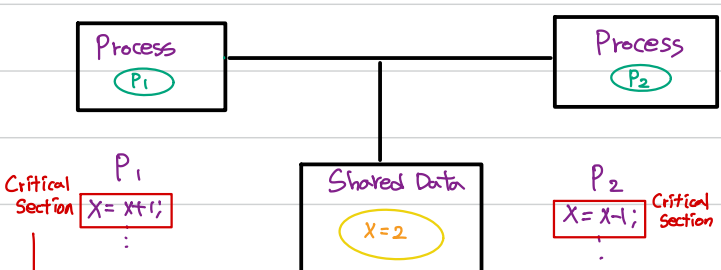
⇒ 보통은 문제가 생기지 않는다. 특별히 두 프로세스가 Shared memory를 쓰고 있다던지 P1이 커널에 있는 데이터를 건드리는 동안에 P2로 넘어간다던지 할 때 생기는 문제... 단순히 P1에서 P2로 넘어간다고 생기는 문제는 아니다.

The Critical-Section Problem

— n개의 프로세스가 공유 데이터를 동시에 사용하기를 원하는 경우

— 각 프로세스의 code segment에는 공유 데이터를 접근하는 코드인 critical section이 존재

- Problem
  - 한 프로세스가 critical section에 있을 때 다른 모든 프로세스는 critical section에 들어갈 수 없어야 한다



→ 공유 데이터에 접근하는 코드를 실행 중이면 CPU가 다른 프로세스에게 넘어 가더라도 공유데이터를 접근하는 Critical Section에 들어가지 못하도록 함.