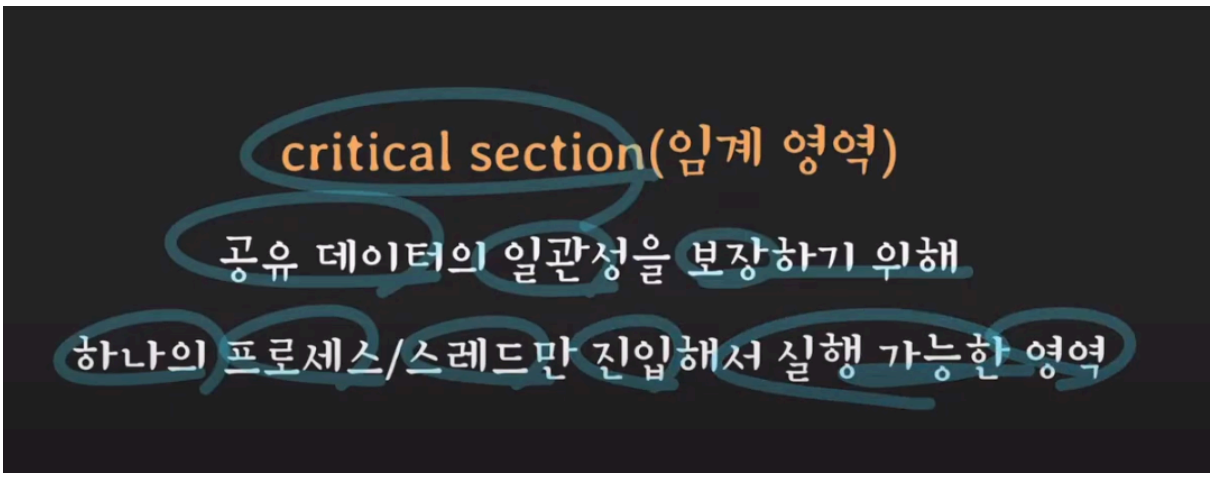
OS 5강

해당 내용은 유튜브 쉬운코드 님의 강의를 개인적인 공부 차원에서 정리하고자 작성한 자료입니다.

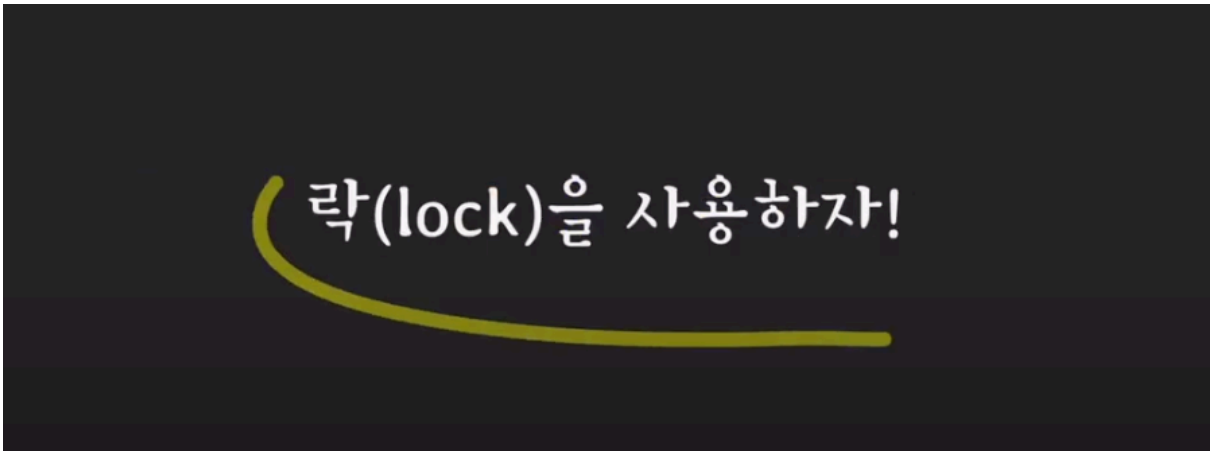
이번 회차는 뮤택스와 세마포어에 대해 다루고 있다



임계 영역은 하나의 프로세스 / 스레드만 진입해서 실행 가능한 영역이라고 했다.

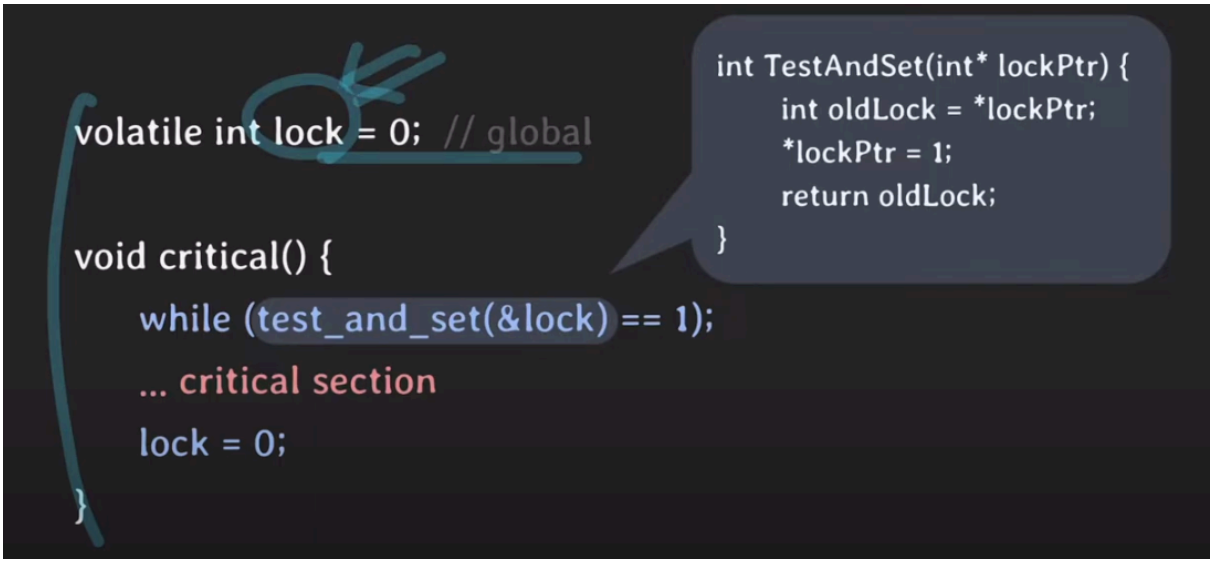
하나의 프로세스 / 스레드만 진입해서 실행하는 것을 mutual exclusion이라고 한다.

어떻게 이걸 보장할 수 있을까?



여러 프로세스나 스레드가 락을 획득하기 위해 경합하고,

획득한 애만 임계영역에 진입 가능하게 하는 것



락이라는 변수를 설정하고,

while loop를 통해 획득한 스레드만 임계영역에 진입하게 해준다.

T1, T2의 스레드가 있고,

먼저 T1 스레드가 먼저 진입한다고 가정해보자.

진입했을때 oldLock의 값은 0이기 때문에 while을 탈출하고

임계영역에 진입하게 되고, lock 변수가 1로 바뀐다.

이때 T2가 이 함수를 실행하려면 lock의 값이 1이기 때문에 루프가 끝나지 않는다.

T1이 정상적으로 임계영역의 코드를 수행하고 lock 값이 0으로 바뀌면

T2가 락을 획득하고 임계영역에 진입할 수 있게 된다.

여기서 문제!!

만약에 T1, T2가 동시에 TestAndSet을 실행하게 되면 둘다 임계영역에 진입할 수 있지 않나요??

→ **불가능하다.**

왜냐면 TestAndSet 메서드는 Atomic(원자적)인 연산이기 때문에

동시에 실행되더라도 CPU (하드웨어)적으로 완전히 하나만 진입할 수 있게 보장되어 있다.

이렇게 구현한 락을 스핀 락(Spin Lock)이라고 한다.

하지만 스핀 락은 CPU를 낭비하게 된다는 단점이 있다.

❌ 스핀락의 주요 단점

1. CPU 낭비 (Busy Waiting) 🔥

- 락을 얻지 못한 스레드가 계속 CPU를 사용하며 루프를 돕니다.
- → **컨텍스트 스위칭은 없지만**, CPU 리소스를 계속 점유하므로 **효율이 매우 떨어짐**.
- 특히 **임계영역이 길어질수록 치명적**입니다.

2. 단일 코어 시스템에서 치명적 🧠

- 락을 보유한 스레드와 락을 기다리는 스레드가 **동일 CPU를 공유**한다면,
- 락 보유자가 **락을 해제할 CPU 시간을 얻지 못해서 교착상태처럼 됨**.

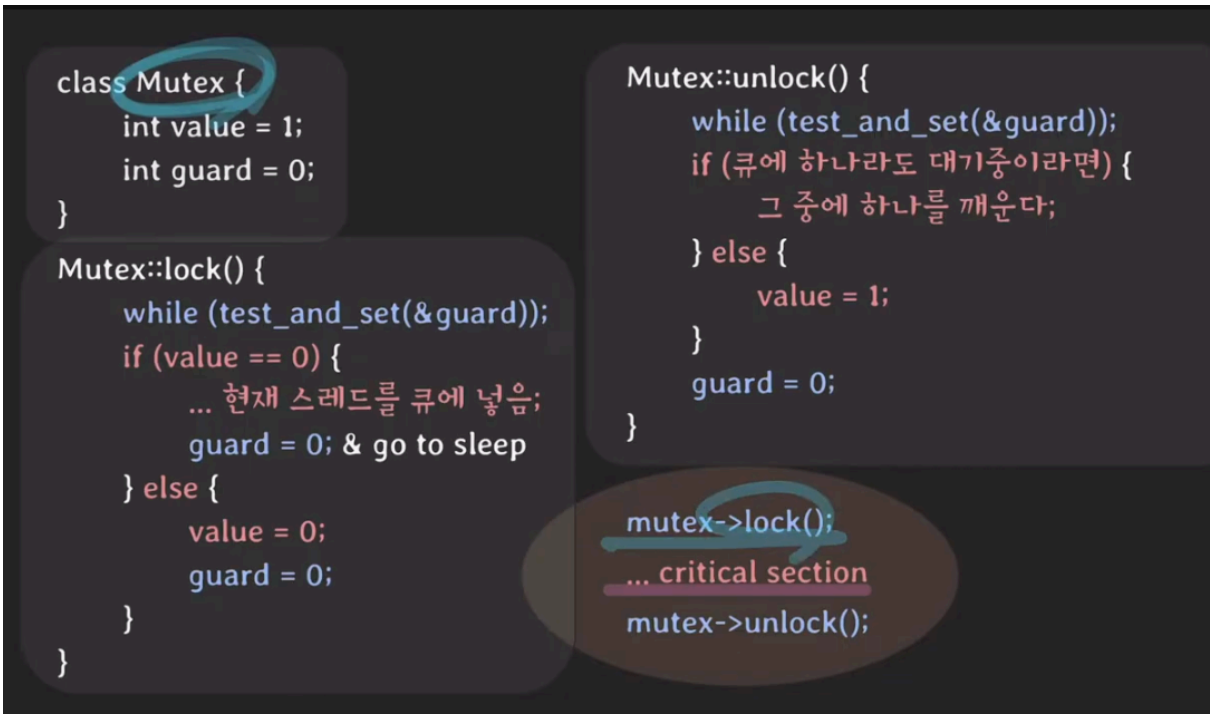
text 복사 편집

```
T1: lock 잡고 임계영역 수행 중 (lock=1)
T2: 계속 while 돌면서 CPU 점유 → T1이 마무리할 기회를 얻지 못함
```

이런 문제를 해결하려면 어떻게 할까?

한 스레드가 진입했다면 쉬고 있어!! 준비되면 깨울게

→ Mutex 방식



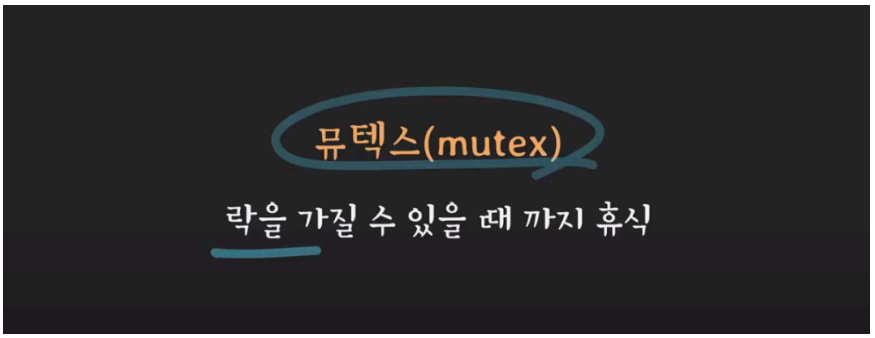
초기 상태에서는 value가 1이기 때문에 guard 락을 획득한 후, value를 0으로 설정하여 락을 점유한다.

T2가 진입하려고 할때는 value 값이 0이기 때문에 guard를 통해 뮤택스(내부) 상태를 확인하고, 자신을 큐에 넣고 sleep 모드로 변경

T1이 임계영역의 코드를 잘 수행하고, unlock() 호출을 통해 대기중인 스레드를 깨우고, guard를 0으로 만들어

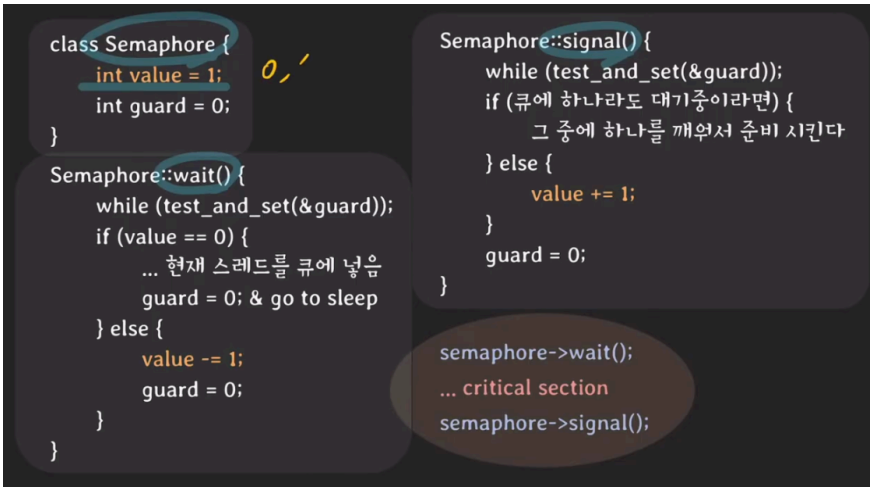
T2가 깨어나서 while문을 탈출하여 임계영역 코드를 수행할 수 있게 만든다.

—> 이 방식이 Mutex 방식이다



만약에 임계 영역에 여러개의 스레드가 들어가게 한다면,

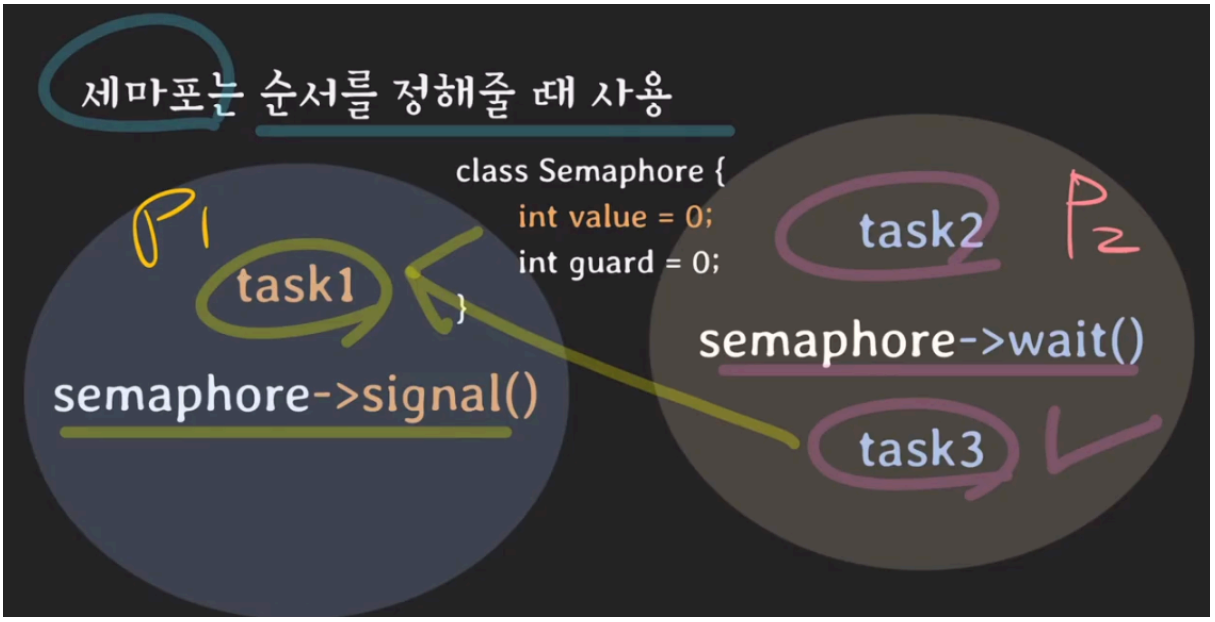
세마포어 방식으로 구현할 수 있다.



만약에 value 값을 0,1로만 고정해서 운용한다면

이진 세마포어 (Binary Semaphore) 라고 한다.

세마포어는 작업의 순서를 정할때도 사용될 수 있다.



만약에 task3가 task1이 먼저 수행되고 나서 실행되어야 하는 조건이 있다고 가정하자.

만약에 p2의 작업이 먼저 시작된다면, task2를 수행하고 wait 메서드에 진입하게 될텐데. 이때 value의 값이 아직 0이기 때문에 루프를 탈출하지 못하고 sleep 상태로 들어간다.

그 다음 p1이 task1을 수행하고 signal()을 통해 p2를 깨우게 될것이고, p2는 task3를 수행할 것이다.

이렇게 세마포어 방식은 **특정 작업 이후에 바로 다른 작업이 실행되도록 순서를 강제하는 역할**도 가능하다.

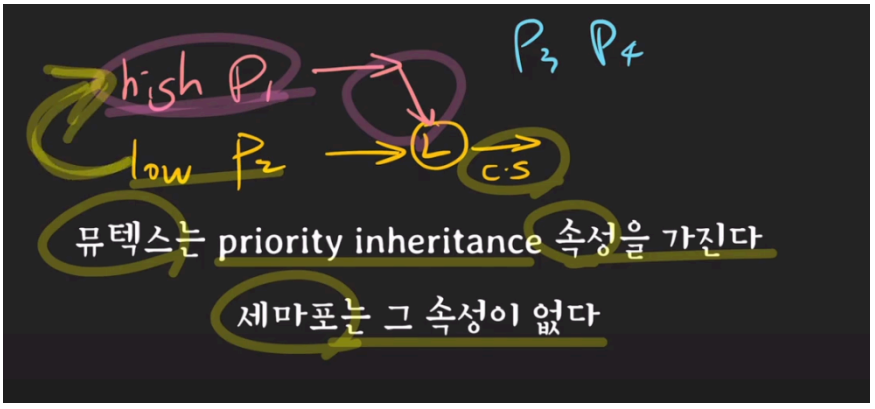
그러면 뮤텍스와 이진 세마포어는 같은거 아니야??

→ 아니다

뮤텍스는 락을 가진 스레드만 락을 해제할 수 있다.

하지만 세마포어는 wait 하는 존재와 signal을 날리는 존재가 다를 수 있다.

핵심적인 사항은 **우선순위 역전 (Priority Inversion)** 문제다.



해당 그림의 예시처럼 p1과 p2의 우선순위가 존재하고,

우선순위가 낮은 p2가 락을 가지고 있다고 가정하자.

만약의 p3, p4가 중간 순위의 우선순위라면, 이 작업들이 수행되느라 p2가 낮은 우선순위 때문에 실행되지 못하고

가장 먼저 실행되어야할 p1가 실행되지 못하고 시스템이 멈춘것 같은 상태가 되어버린다.

이 문제를 해결하기 위해서는 **락을 보유하고 있는 p2의 일시적으로 높은 우선순위를 주어 먼저 실행되도록 하는 방**식이 필요하다.

이걸 **우선순위 상속 (Priority Inheritance)** 라고 한다.

이 문제를 해결하기 위해서는 락을 소유하고 있는 소유자가 누군지 정확하게 알아야 하는데,

이진 세마포어로 구현할 경우 signal과 wait를 아무 스레드나 실행시킬 수 있기 때문에 운영체제가 우선순위의 스레드를 지정하고 상속할 근거가 부족하다.

만약 순서 제어가 필요한 시스템이라면, 이진 세마포어

상호 배제가 중요한 시스템이라면 뮤텍스를 선택하도록 하자.