동기화

Race condition

여러 프로세스나 스레드가 동시에 공유 자원에 접근할 때 타이밍이나 접근 순서에 따라 결과가 달라질 수 있는 상황

공유 자원에 여러 프로세스/스레드가 동시에 접근했을 때 의도치 않은 동작이 발생할 수 있다.

→ 따라서 프로세스나 스레드를 공유 자원에 동시에 접근하지 못하도록 접근 순서를 제어하는 기법이 필요함.

Critical Section

임계 영역: 공유 데이터의 일관성을 보장하기 위해 **특정 프로세스/스레드만 진입해서 실행** 가능한 코드 영역

특정 프로세스/스레드만 임계 영역에서 진입해서 실행하는 것: `상호 배제(Mutual Exclusion)`

상호 배제를 동기화 기법으로 구현한 것이 뮤텍스와 세마포어.

Mutex

상호 배제를 구현하기 위해 존재하는 락(lock)

뮤텍스 락을 얻은 경우에만 임계 영역에 진입하여 작업을 실행 가능하다.

특징

- boolean 타입의 락 변수
- 한 개의 프로세스/스레드
- Acquire / Release
- Non Busy-wait
    - Busy-wait: CPU를 계속해서 사용하며 특정 조건을 만족할 때 까지 대기하는 방식
    - Non Busy-wait: 대기 큐에서 cpu 자원을 내려놓고 대기하는 방식

자바에서는 `ReentrantLock` 을 사용하여 뮤텍스를 구현할 수 있다.

Semaphore

공유 자원에 접근 가능한 스레드 개수를 관리하는 방식

여러 개의 프로세스/스레드가 공유 자원에 동시에 접근하는 것을 제한하기 위한 정수

- Counting semaphore
    - 세마포어가 획득되면 관리되는 스레드 개수 감소

    - 세마포어를 릴리즈하면 스레드 개수 증가
    - 세마포어 값이 0 이하면 다른 프로세스/스레드는 임계 영역에 접근 불가.
    - 세마포어 값이 음수라면 해당 숫자의 절댓값은 대기 큐에서 기다리는 프로세스/스레드의 개수를 의미.
- Binary semaphore: 세마포어 값이 0 또는 1
    - 최대 1개의 프로세스/스레드가 접근 가능.

자바에서는 `Semaphore` 클래스로 사용 가능.

특징

- 다중 프로세스, 다중 스레드 환경에서 적용
- Wait: 세마포어 값 감소, P 연산
- Signal: 세마포어 값 증가, V 연산
- Non-Busy-wait

차이점

뮤텍스는 락을 소유한 프로세스/스레드만 락에 대한 소유/해제 권리를 가지고 있다.

세마포어는 서로 다른 프로세스/스레드가 세마포어를 감소하고 증가시킬 수 있다.

- 1개의 자원에 대한 상호 배제만 필요하다면 Mutex
- 공유 자원 접근 수를 조절하고, 작업 간 실행 순서 동기화가 필요하다면 세마포어

Binary Semaphore와 Mutex 차이점

- Binary Semaphore
    - 값을 0과 1만 가지는 세마포어
    - 상호 배제를 위해 신호 전달 메커니즘을 사용
    - 세마포어 값이 0이면 잠겨있는 것이고, 1이면 잠금이 해제된 것
- Mutex
    - lock 기반 메커니즘으로 상호 배제를 구현
    - 하나의 스레드만 락에 접근 가능

Binary Semaphore	**Mutex**
Signal/Wait	Acquire/Release
현재 스레드보다 우선순위가 높은 스레드가 세마포어를 해제하고 잠글 수 있다.	한 번에 하나의 스레드만 뮤텍스를 획득 가능

상태 동기화에 사용	임계 구역 문제를 해결하기 위해 사용
일반적으로 프로세스나 스레드 사이에 공유된다. 다른 스레드가 릴리스할 수 있다. → 소유권이 없다.	뮤텍스를 소유한 스레드만 잠금을 해제하므로 소유권이 있다.
다른 프로세스/스레드가 해제할 수 있으므로 뮤텍스보다 빠르다	뮤텍스를 가진 프로세스/스레드만 해제가 가능하므로 비교적 느리다.
공유 자원에 대한 여러 인스턴스가 존재하는 경우 유용	자원에 대한 하나의 인스턴스가 있을 경우 뮤텍스가 유용

Q1: When is a Binary Semaphore useful?

Binary Semaphores are useful when you need to coordinate between multiple threads to control access to a shared resource. They are often used to signal whether a resource is available for use or to manage access to a critical section.

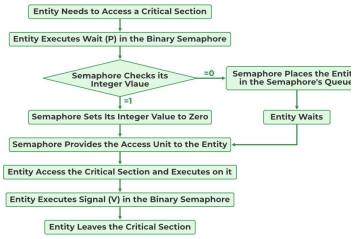
Q2: What is the purpose of a Mutex?

A Mutex is used to provide mutual exclusion, ensuring that only one thread can enter a critical section of code at a time. This prevents conflicts and data inconsistencies that can occur when multiple threads access shared resources concurrently.

Difference between Binary Semaphore and Mutex - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/difference-between-binary-semaphore-and-mutex/

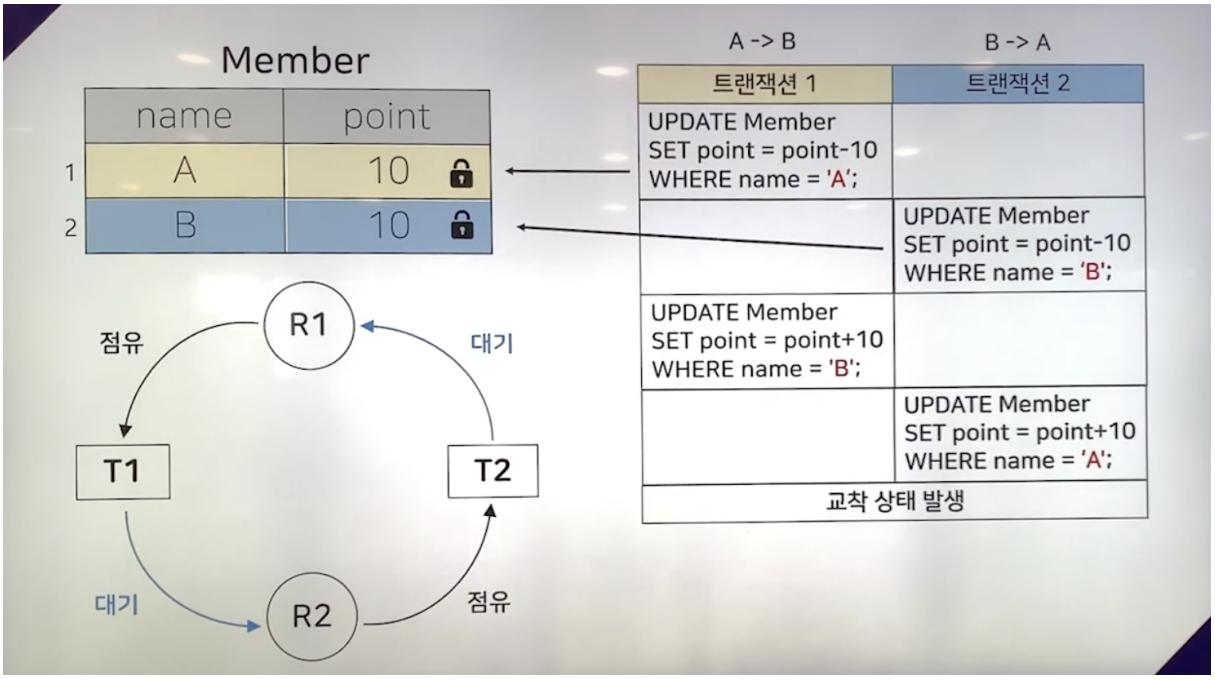


Deadlock

뮤텍스와 세마포어를 잘못 사용했을 때 교착상태, 데드락이 발생할 수 있다.

정의

프로세스들이 서로의 자원을 기다리며 무한 대기에 빠지는 현상.



트랜잭션 2개가 있다고 가정.

트랜잭션 1은 A에서 B로 점수를 올리는 작업,

트랜잭션 2는 B에서 A로 점수를 올리는 작업이다.

우연찮게 트랜잭션 1과 2가 거의 동시에 진행이 되었는데, 트랜잭션 1은 A에 대한 락을 가지고 있고 2는 B에 대한 락을 가지고 있다.

이 상황에서 트랜잭션 1이 B에 대한 락을 요청했으나 트랜잭션 2가 release하지 않았기 때문에 교착 상태가 발생한다.

조건

다음과 같은 4가지 조건이 동시에 충족될 때 발생할 수 있다.

1. `Mutual Exclusion(상호 배제)` : 한 번에 하나의 프로세스만 해당 자원을 사용할 수 있어야 한다.
2. `Hold and wait(점유와 대기)` : 프로세스가 최소한 하나의 자원을 보유하고 있고, 동시에 다른 자원을 기다리는 상태여야 한다.

   프로세스는 자원을 이미 보유하고 있으면서 다른 자원을 요청하고 있어야 한다.
3. `No Preemption(비선점)` : 이미 할당된 자원을 다른 프로세스가 강제로 빼앗을 수 없다.
4. `Circular Wait(순환 대기)` : 프로세스 간의 자원 요청의 사이클이 형성되어야 한다. 각 프로세스가 자원을 요청할 때 이전 프로세스가 보유한 자원을 요청해야 한다.

해결법

교착 상태 무시

교착 상태가 드물게 발생하는 시스템에서 일반적으로 사용하는 방법

- 윈도우, 유닉스
- 교착 상태 해결 비용 문제 - 발생한다면 사용자가 시스템 재부팅 등의 방법을 거치게 함.

교착 상태 탐지 및 복구

교착 상태가 많이 발생하는 시스템에서 일반적으로 사용하는 방법.

순환 대기 존재 여부에 초점을 맞춤.

탐지

- 자원 할당 그래프 소거
- 프로세스와 자원 사이의 관계를 그래프로 나타내고, 프로세스 간의 순환을 없애는 전략

얼마나 자주 탐지 알고리즘을 호출할 것인가?

- 주기적으로
- 자원을 즉시 할당하지 않았다면
- CPU 이용률이 떨어지는 경우

복구

- 순환 대기가 깨질 때까지 프로세스를 종료시키기
- 순환 대기에 포함된 프로세스의 제어권을 뺏고 롤백
    - 어떤 프로세스를 희생양으로?
    - 우선순위에 따라...
        - MySQL: 작업한 양이 가장 적은 트랜잭션을 희생양으로 삼는다.

교착 상태 예방

- 교착 상태는 필요악: 자원의 효율적 이용을 위해 어쩔 수 없이 교착 상태를 감수해야 한다.
- 현대 시스템에서는 잘 사용 X

교착 상태 회피

- 은행원 알고리즘
    - 시스템을 안전 상태/불안전 상태로 구분하고 불안전 상태일 때는 대기.
    - 안전 상태일 때만 자원을 빌려준다.
    - 자원이 동적으로 움직이는 현대 시스템에서는 잘 사용 X
- 회피의 오버헤드가 크다.
    - '교착 상태 발생한 것 같다!' → 탐지 및 복구
    - 자원을 요청할 때마다 시스템의 상태를 판단하고 회피 → 상대적 오버헤드 크다.