22.12.26

📌DeadLock - 교착 상태 회피, 검출, 회복

교착 상태 회피

- 개념
  - 할당하는 **자원의 수**를 교착 상태가 발생하지 않을 수준으로만 **분배**한다.
  - 자원의 총 수와 할당된 자원의 수를 기준으로 시스템을 **안정 상태(Safe state)**와 **불안정 상태(Unsafe state)**로 나눈다.
- 은행원 알고리즘
  - 은행에서 대출 금액이 대출 가능한 범위 내이면(안정 상태이면) 허용되고 아니면 거부되는 것과 유사하여 이러한 이름이 붙음
  - 변수
    - 전체 자원(Total) - 시스템 내 전체 자원의 수
    - 가용 자원(Available) - 시스템 내 현재 사용할 수 있는 자원의 수 (전체 자원 - 모든 프로세스의 할당 자원)
    - 최대 자원(Max) - 각 프로세스가 선언한 전체 자원의 수
    - 할당 자원(Allocaiton) - 각 프로세스에 현재 할당된 자원의 수
    - 기대 자원(Expect) - 각 프로세스가 앞으로 사용할 자원의 수(최대 자원 - 할당 자원)
  - 안정 상태의 정의
    - 하나의 이상의 프로세스에서 **Expect≤ Available**이 성립할 때
  - 단점
    - 프로세스가 모든 사용할 자원을 선언해야 함
    - 시스템의 전체 자원 수는 고정적이지 않음 ex) 새로운 자원 추가
    - 교착 상태가 발생 않는데도 하면 **자원 낭비**

교착 상태 검출

- 개념
  - 교착 상태 예방은 구현이 어렵고, 회피는 자원 낭비의 문제가 있음
  - 운영체제가 프로세스의 작업을 관찰하면서 **교착 상태 발생 여부를 계속 주시**하는 방식
- 방식

  1) 타임 아웃 (가벼운 교착 상태 검출)
  - **일정 시간** 동안 작업이 진행되지 않은 프로세스를 교착 상태로 간주하여 처리하는 방식
  - 문제
    - 엉뚱한 프로세스가 종료됨. 작업 진행 안하는 프로세스가 교착 상태인 것은 아님.
    - 모든 시스템에 적용하기 어려움
  - but 비교적 구현 쉬워서 많이 사용됨.

    ex) 윈도우의 “응답이 없어서 종료합니다”, DB의 체크포인트, 롤백

  2) 자원 할당 그래프
  - 프로세스의 작업 방식을 제한하지 않고, 교착 상태를 정확히 파악 가능
  - 자원 할당 그래프를 유지하고, 갱신하는 비용

교착 상태 회복

- 교착 상태 검출 시, 이를 해결하는 것
- 방식
  - 교착 상태 일으킨 모든 프로세스 종료
  - 교착 상태 일으킨 프로세스 중 우선순위를 통해 순서대로 하나씩 종료

📌세마포어가 여러 개의 프로세스에게 임계 영역을 허용하는 이유

세마포어는 “몇 개의 프로세스에게 해당 자원(혹은 임계 영역)을 허락할 것인지에 대한 동기화 방안” 으로 정리될 수 있다. 이는 스레드 풀의 예시를 통해 설명이 가능하다.

예를 들어 한 서버의 요청을 받을 수 있는 스레드 풀이 10개 라면, 서버는 요청을 10개로 제한하거나 큐를 사용하여 한 시점에 스레드 풀이 10개 이상의 요청을 한 번에 받지 않도록 할 의무가 생긴다. 이 이상의 요청이 스레드풀에 발생하면, 에러가 발생되어야 한다.

이와 같이 사용된 스레드(자원)가 반납 되고, 기다리던 요청(waiting pool)을 받아(signal) 스레드가 사용되는 스레드 풀 방식을 통해 세마포어의 여러 프로세스에게 자원(혹은 임계영역)을 허용하는 이유를 설명해볼 수 있겠다.

📌스프링 순환 참조 문제

A 클래스의 Bean을 만드는 과정에서 B 클래스의 Bean을 주입하려고 B 클래스를 생성한다. 그런데 B 클래스에 A 클래스 Bean을 주입하려고 했는데 A 클래스가 없다.. 이런 식으로 무한 반복

→ 결과적으로 어떠한 Bean도 만들지 못하는 문제를 순환 참조 문제라고 한다.

- 생성자를 통한 의존성 주입 시 발생하는 문제이다.
  - 필드 주입과 생성자 주입은 먼저 빈을 생성한 후, 주입하려는 빈을 찾아 주입한다.
  - 생성자 주입은 먼저 생성자의 인자에 사용되는 빈을 찾거나 빈 팩토리에서 만든다. 그 후에 찾은 인자 빈으로 주입하려는 빈의 생성자를 호출한다. 즉, 먼저 빈을 생성하지 않고 주입하려는 빈을 먼저 찾는다.

    따라서 참조하는 객체가 생성되지 않은 상태에서 빈을 참조하기 때문에 오류가 발생한다.
- 해결책
  - 생성자 메서드에 @Lazy 애노테이션
    - 의존성 주입을 애플리케이션 로딩 시점이 아닌 빈이 필요한 시점에 주입을 받음.
    - 특정 Http 요청을 받았을 때 Heap 메모리가 증가할 수 있으며 메모리가 충분하지 않을 경우 장애가 발생할 수 있다.
  - 애초에 순환 참조가 되지 않도록 설정
  - EventListener

*순환 호출 문제

- 의존성을 필드나 수정자로 주입 받을 경우, 스프링 애플리케이션 로딩 시의 예외가 발생하지 않는다.

- 메소드가 순환 호출될 때, 예외가 발생할 수 있다.