쓰레드

💡 작성자: 신채린

Thread 구현하기
    Thread란??
    Thread 구현하기
    Multi-Thread 프로그래밍
    Thread Status
    Thread 우선순위
    join() 메서드
    interrupt() 메서드
    Thread 종료하기
Multi-thread 프로그래밍
    임계 영역 (critical section)
    동기화 (synchronization)
critical section과 동기화
    자바에서의 동기화 구현
    deadlock
    wait()/notify()

Thread 구현하기

- Shared resource가 발생할 수 있는데, 해당 공유 자원을 동시에 사용하면 문제가 될 수 있음
- 때문에 Synchornization 동기화를 통해 순서를 맞춰줘야 함.

Thread란??

- Process
    - 실행중인 프로그램, 프로그램이 메모리에 올라간 상태
    - OS로부터 메모리를 할당 받음
- Thread
    - 실제 프로그램이 수행되는 작업이 최소 단위
    - 하나의 프로세스는 하나 이상의 Thread를 가지게 됨 - CPU 점유

- 두 개가 동시에 돌아가는 것처럼 보이는 것이 Multi thread이다.
- CPU를 점유 할 수 있는 것은 스케줄러인데, 스케줄러가 thread의 CPU를 할당해서 thread가 수행되도록 한다.

Thread 구현하기

- Java Thread 클래스로부터 상속받아 구현
- Runnable 인터페이스 구현

Multi-Thread 프로그래밍

- 동시에 여러 개의 Thread가 수행되는 프로그래밍
- Thread는 각각의 작업 공간 (context)를 가짐
  - Thread가 switching되면, context도 switching된다. (context switching)
  - reference하는 변수나, 여러가지 작업 환경들이 함께 switching 된다.
- 프로세스 안에 여러 thread가 있다고 하면, thread들이 공유하는 자원이 있다.
  - 공유 자원(shared resource)이 있는 경우 race condition이 발생
    - 서로 공유 자원을 가지려고 하는 것이 race condition이다.
  - critical section(임계 영역)에 대한 동기화 (synchronization)의 구현이 필요
    - critical section을 2개의 thread가 동시에 접근하게 되면, 하나가 어떤 값을 가져와서 더하고, 다른 하나는 어떤 값을 뺀다면 꼬일 수 있다.
    - 이런 영역에 대해 순서를 지키자는 것이 동기화이다.
    - critical section에 들어갈 수 있는 thread는 한 번에 1개여야 한다.
      - 이를 위해 Critical Section에 lock을 건다 (Semaphore) - 다른 thread는 접근할 수 없도록 한다.

Thread Status

- Runnable한 상태에서 CPU 를 배분할 수 잇다.
- CPU가 thread를 배분할 수 없는 상태 - not runnable
  - sleep(), wait(), join()

Thread 우선순위

- MIN_PRIORITY(=1) ~ Thread.MAX_PRIORITY(=10)
- 디폴트 우선순위: Thread.NORM_PRIORITY(=5)

· setPriority(int newPriority); · int getPriority();

- 우선 순위가 높은 thread는 CPU를 배분 받을 확률이 높음

join() 메서드

- 다른 thread의 결과를 보고 진행해야 하는 일이 있는 경우, join() 메서드를 활용
- join() 메서드를 호출한 thread가 non-runnable 상태가 됨

interrupt() 메서드

- 다른 thread에 예외를 발생시키는 interrupt를 보냄
- thread가 join(), sleep(), wait() 메서드에 의해 blocking 되었다면 interrupt에 의해 다시 runnable 상태가 될 수 있음

Thread 종료하기

- 데몬 등 무한 반복하는 thread가 종료될 수 있도록 run() 메서드 내의 while 문을 활용
- stop()은 사용하지 않음

Multi-thread 프로그래밍

임계 영역 (critical section)

- 두 개 이상의 thread가 동시에 접근하게 되는 리소스
- critical section에 동시에 thread가 접근하게 되면 실행 결과를 보장할 수 없음
- thread 간의 순서를 맞추는 동기화 (synchronization)이 필요

동기화 (synchronization)

- 임계 영역에 여러 thread가 접근하는 경우, 한 thread가 수행하는 동안 공유 자원을 lock하려 다른 thread의 접근을 막음
- 동기화를 잘못 구현하면 deadlock에 빠질 수 있음

critical section과 동기화

자바에서의 동기화 구현

- synchronized 수행문과 synchronized 메서드를 이용
- synchronized 수행문
  - synchronized(참조형 수식) {}
  - 참조형 수식에 해당되는 객체에 lock을 건다.
- synchronized 메서드
  - 현재 이 메서드가 속해 있는 객체에 lock을 건다.
  - synchronized 메서드 내에서 다른 synchronized 메서드를 호출하지 않는다. (deadlock 방지를 위해)

deadlock

wait()/notify()

- wait()
  - 리소스가 더 이상 유효하지 않은 경우 리소스가 사용 가능할 때 까지 위해 thread를 non-runnable 상태로 전환
  - wait() 상태가 된 thread는 notify()가 호출될 때까지 기다린다.
- notify()
  - wait()하고 있는 thread 중 한 thread를 runnable한 상태를 깨움
- notifyAll()
  - wait() 하고 있는 모든 thread가 runnable한 상태가 되도록 함
  - notify() 보다 notifyAll()을 사용하기를 권장
  - 특정 thread가 통지를 받도록 제어할 수 없으므로 모두 깨운 후 scheduler에 CPU를 점유하는 것이 좀 더 공평하다고 함
- notify보다 notifyAll 권장