item 84

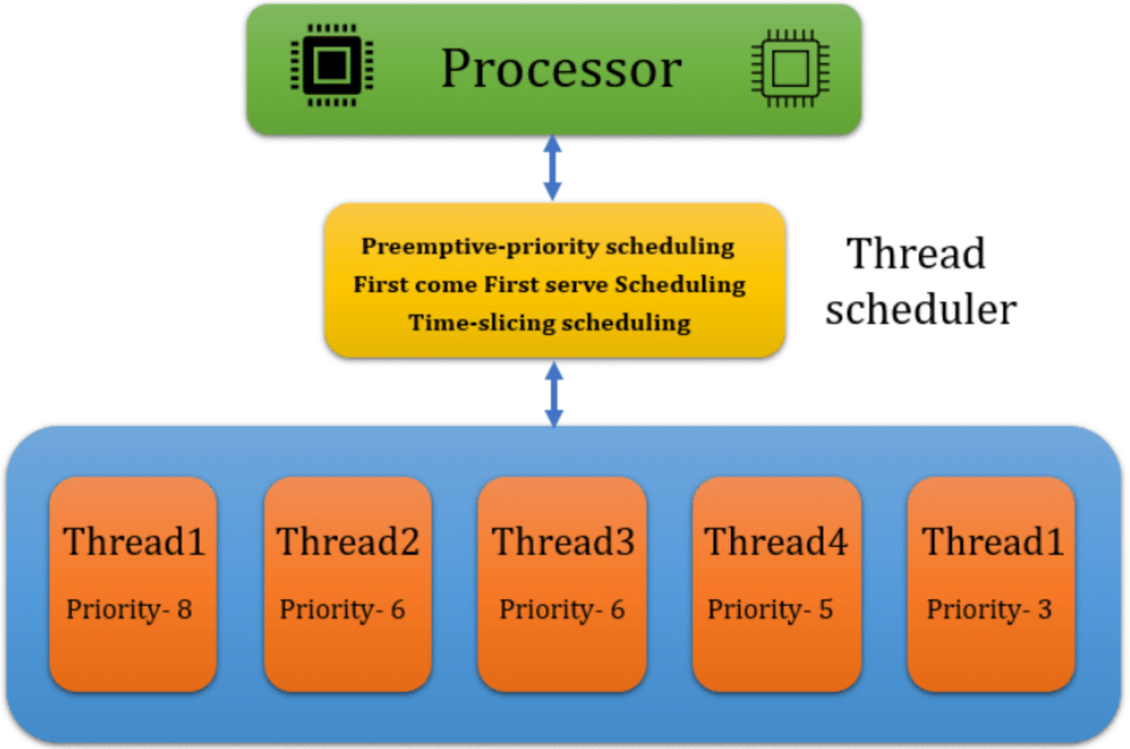
프로그램의 동작을 스레드 스케줄러에 기대지 말라

유현우

스레드 스케줄러란?

- 운영체제에서 다중 스레드를 관리하며, CPU를 사용할 수 있는 스레드를 선택하고, CPU를 할당하는 작업을 말한다.
- 스레드의 우선순위, 실행 시간, 입출력 요청 등의 정보를 고려한 알고리즘을 통해서 CPU를 사용할 수 있는 스레드를 선택한다.
  - 대표적으로는 Round Robin, Priority-based scheduling, Multi-level Queue scheduling 등

스레드 스케줄러란?



하나의 프로세스 내에서 다수의 스레드가 동작하는 형태이기 때문에, 스레드 간의 상호작용과 동기화 문제를 고려해야 한다

스레드 스케줄러란?

이 스레드 스케줄링 정책은 OS다를 수 있기 때문에 정확성이나 성능이 스레드 스케줄러에 따라 달라지는 프로그램(OS에 따라 달라지는 프로그램)이라면 다른 플랫폼에 이식하기 어렵다

이식성 좋은 프로그램을 작성하는 방법

- 실행 가능한 스레드의 평균적인 수를 프로세서 수보다 지나치게 많아지지 않도록 해야 한다.
- 스레드를 busy waiting 상태가 되지 않도록 해야 한다.

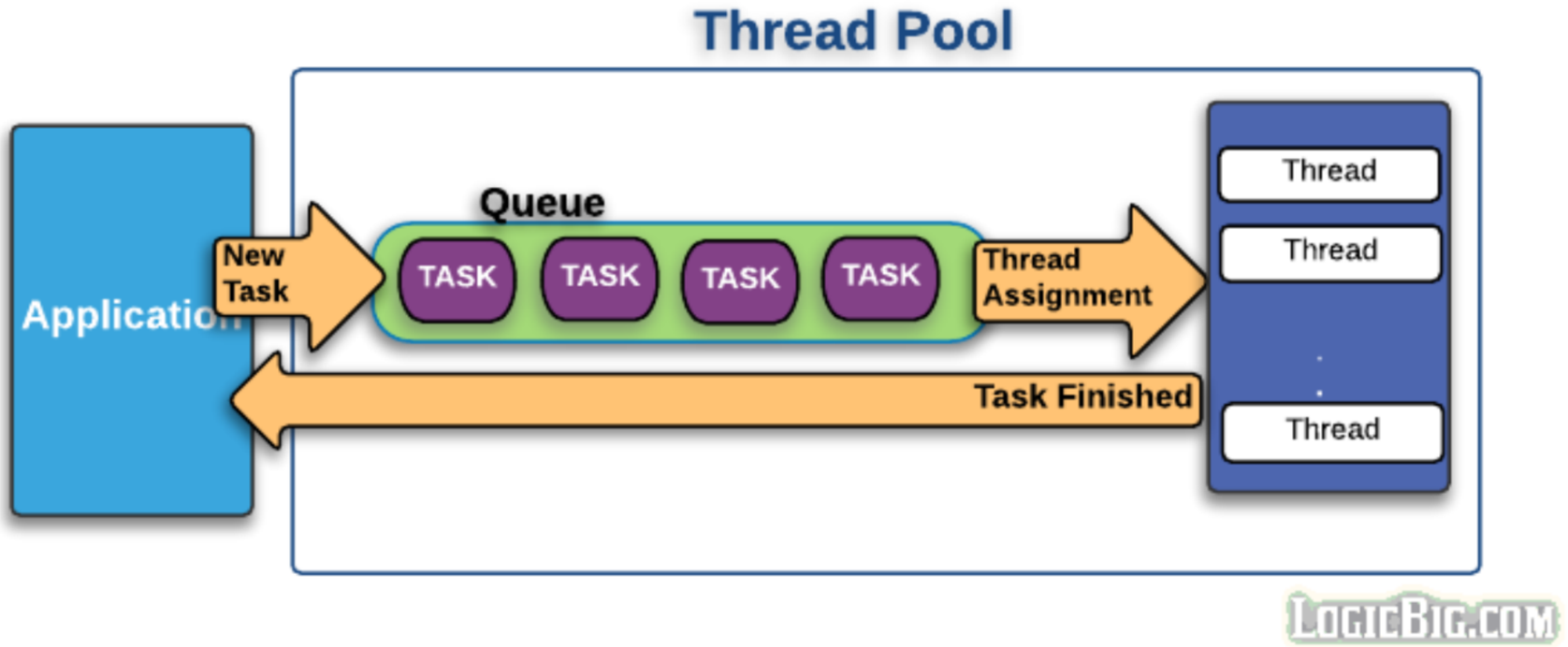
전체 스레드수 = 대기중인 스레드수(실행 가능하지 않은 스레드 수) + 실행중인 스레드 수

이식성 좋은 프로그램을 작성하는 방법

- 실행 가능한 스레드의 평균적인 수를 프로세서 수보다 지나치게 많아지지 않도록 해야 한다.
- 스레드를 busy waiting 상태가 되지 않도록 해야 한다.

실행가능한 스레드 수를 어떻게 적게 유지할까?

각 스레드가 무언가 유용한 작업을 완료한 후에는 다음 작업이 생길 때까지 대기하도록 하는 것



스레드 풀 크기를 적절히 설정하고, 작업은 짧게 유지한다.

(너무 짧으면 작업 분배가 성능을 떨어뜨릴 수도 있음)

이식성 좋은 프로그램을 작성하는 방법

- 실행 가능한 스레드의 평균적인 수를 프로세서 수보다 지나치게 많아지지 않도록 해야 한다.
- 스레드를 busy waiting 상태가 되지 않도록 해야 한다.

busy waiting(바쁜 대기) 이란?

OS에서는 원하는 자원을 얻기 위해 기다리는 것이 아니라 **권한(공유 자원 접근)을 얻을 때까지 확인**하는 것을 의미한다.

- 스레드 스케줄러의 변덕에 취약할 뿐 아니라, CPU의 자원을 쓸데 없이 낭비하기 때문에 좋지 않은 쓰레드 동기화 방식이다.
- 스레드의 동기화를 위해서 Busy Waiting Method를 사용할 것이 아닌, 뮤텍스 세마포어(Mutual Exclusion) 또는 Monitor를 사용해야 한다고 알려져 있다.

Sleeping 이란?

권한을 얻기 위해 기다리는 시간을 Wait Queue 에 실행 중인 Thread 정보를 담고 다른 Thread에게 CPU를 양보하는 것을 의미한다.

- 커널은 권한 이벤트가 발생하면 Wait Queue 에 담긴 Thread 정보를 깨워 CPU에 부여한다.
- 기다리는 시간이 예측 불가능한 상황에서 쓰인다.
- Wait Queue에 넣는 자원 비용 + Context Switching 비용이 들어가게 된다.

CountDownLatch - busy waiting 예제

```java
public class SlowCountDownLatch {
  private int count;

  public SlowCountDownLatch(int count) {
    if (count < 0)
      throw new IllegalArgumentException(count + " < 0");
    this.count = count;
  }

  public void await() {
    while (true) {
      synchronized(this) {
        if (count == 0)
          return;
      }
    }
  }
  public synchronized void countDown() {
    if (count != 0)
      count--;
  }
}
```

- 여러 스레드가 동시에 await()를 호출하면, 각각의 스레드가 조건이 만족될 때까지 계속 해서 루프를 돕니다.
- CPU 리소스 소모뿐만 아니라 캐시 히트율 저하 등 컨텍스트 스위칭 비용을 초래할 수 있습니다.
- 다양한 스케줄링 정책을 가진 운영체제에서 예상치 못한 성능 차이를 초래할 수 있다.

이식성 나쁜 코드

- Thread.yield
- Thread Priority

Thread.yield

```
class ThreadA extends Thread{
    private boolean stop = false;
    private boolean flag = true;

    public void setFlag(boolean flag){
        this.flag = flag;
    }

    public void setStop(boolean stop){
        this.stop = stop;
    }

    public void run(){
        while(!stop){
            if(flag){
                System.out.println("ThreadA is working..");
            }
            else{
                Thread.yield();
            }
        }
    }
}
```

flag가 false가 되면 다른 스레드에게 실행 양보

Thread.yield

특정 스레드가 다른 스레드들과 비교하면서 CPU 시간을 충분히 얻지 못하는 프로그램을 고치기 위해 사용할 수 있다.

- Thread.yield는 테스트할 수단도 없다.
- JVM 버전 혹은, JVM 종류에 따라서 성능이 달라지는 것은 올바르지 않다.
- 차라리 애플리케이션 구조를 바꿔 동시에 실행 가능한 스레드 수가 적어지도록 조치하자.

Thread Priority

스레드 우선순위는 자바에서 이식성이 가장 나쁜 특성에 속하며,
심각한 응답 불가 문제를 스레드 우선순위로 해결하려는 시도는 절대 합리적이지 않다.
(진짜 원인을 찾아 수정하자.)