8/3 ~ 8/7 5주차 - 1

쓰레드 세입에 관하여

쓰레스 세이프 (Thread-safe)하다는 것이 어떤 의지인지?

- 멀티 스레드 프로그래밍에서 일반적으로 어떤 함수나 변수, 혹은 객체가 여러 스레드로부터 동시에 접근이 이루어져도 프로그램의 실행에 문제가 없음을 뜻한다.
- 멀티 스레드 환경에서 동작해도 원래 의도한 형태로 동작하는 코드를 'Thread-safe하다'라고 말할 수 있다. 예를 들어, Thread-safe하지 않은 코드는 다음 코드가 있다.

```
int num = 1;
boolean is_even;

int inc(int n)
{
  num += n;
  if((num % 2) == 0)
    is_even = true;
  else
    is_even = false;
  return num;
}
```

위 코드를 보면, num이라는 변수에 숫자를 더해서 짝수면 true 홀수면 false

싱글 스레드 환경에서는 문제가 없는 코드다.

하지만 멀티 스레드 환경에서 고민해보자. 위 코드는 문제를 발생시킬 소지가 있다.

두개의 스레드가 위 코드를 같이 수행한다고 가정하면.

두개의 스레드는 전역변수인 num, is_even을 공유한다.

1번과 2번 스레드가 다음 함수를 실행하고

```
a = inc(1);
```

1번스레드가 먼저 다음 라인 까지 수행했다면.

```
num += n;
if ((num % 2) == 0)
```

num이 1인 상황에서 1을 더했으니 num % 2는 0이 되는 게 맞다. 그리고 is_even에 true를 설정하기 바로 직전 다른 스레드가 다음 코드를 수행했다고 하자.

```
  num += n; // 2 + 1 => 3
  ...
  else
    is_even = false;
  retrun num;
}
```

1번 스레드가 num을 2로 세팅해놓았기 때문에 num += n 을 계산하면 3이 된다.

3은 홀수이므로 is_even = false가 된다.

이때 , 다시 1번 스레드가 수행되어 is_even = true; 를 수행한다.

이런 수행 순서를 지나면, num값은 3이 되고, is_even은 true로 세팅된다.

프로그래머의 의도와 맞지 않는 결과가 멀티 스레드 환경에서 발생했기 때문에 Thread-safe 하지 않다.

Thread -safe 하게 만들기

Critical session이라는 개념을 도입해서 연산들을 Serialize(직렬화) 해줘야 한다.

```
int num;
boolean is_even;
pthread_mutext mutex_lock = PTHREAD_MUTEX_INITIALIZERR;

int inc(int n)
{
  pthread_mutex_lock (&mutex_lock);
  // Critical Section Start
  num += n;
  if (( num % 2) == )
    is_even = true;
  else
    is_even = false;

  // Critical Section END
  pthread_mutex_unlock (&mutex_lock);

  return;
}
```

위와 같이 Critical Section을 정의해서 Critical Section은 한 번에 한 스레드만 수행되도록 만들어줘야 한다.

pthread_mutex_lock (&mutex_lock);은 mutex_lock 변수에 serialize 포인트를 둔다.

즉, pthread_mutext_lock(&mutex_lock); ~ pthread_mutex_unlock(&mutex_lock); 사이에는 하나의 스레드만 들어갈 수 있다.

(pthread_mutext_lock(&another_lock) 같이 다른 serialize 포인트는 막히지 않는다.)

이후 thread_mutex_unlock (&mutex_lock);으로 하나의 스레드가 critical section에서 나오면, 기다리고 있었던 다른 스레드가 critical section으로 들어가게 된다. 이때, 들어가는 순서는 critical section을 만드는 구현체에 따라 다르다.

wait queue를 두어 FIFO로 구현할 수도 있고, spinlock처럼 random하게 구현할 수도 있다.