OS01 4주차 과제 - 201702081

- MUTEX

1. counter가 원래는 0으로 끝나야 하나, 이상한 값에서 끝나는 것을 알 수 있다.

   이는 메인과 스레드가 각각 counter를 변화시키는 과정에서, load-change-store 인스트럭션을 수행하는 중 간섭을 받았기 때문이다.

   결과적으로 정확하게 계산되지 않게 된다.

```
u201702081@os:~/Desktop$ ./raw
main[1885546304] : begin (counter = 0)
thread[1885542144] : begin
thread[1885542144] : done
main[1885546304] : done (counter = 9783830) (ma1.val = 200000000)
```

2. pthread_join을 없애고 실행을 여러번 반복하다 보면, thread가 완전히 끝나지 않았음에도 프로그램이 종료되는 것을 확인할 수 있다. 이는 thread를 따로 기다리지 않기 때문이다. 따라서 ma1.val이 다 증가되지 않은 채로 끝나게 된다.

```
u201702081@os:~/Desktop$ ./raw
main[1705420608] : begin (counter = 0)
thread[1705416448] : begin
main[1705420608] : done (counter = 40849256) (ma1.val = 193822110)
```

3. 스레드의 서브루틴 내 counter를 감소시키는 부분과 (14번)

   메인 루틴 내 counter를 증가시키는 부분(28번) 이 Critical Section에 해당된다.

   이는 공유 변수인 counter에 접근하기 때문이다.

```
13      for (int i = 0; i < end; i++) {
14          --counter;
15          ma->val++;
16      }
```

```
27      for (int i = 0; i < end; i++)
28          counter++;
```

4. 위의 Critical Section 주위에 lock을 걸었더니, counter 값이 0으로 잘 나오는 것을 확인할 수 있었다.

   counter는 main에서 1억 증가, thread에서 1억 감소하여 0으로 돌아오고,

   ma1.val은 초기값 1억에서 thread를 통해 1억이 증가하여 2억이 된다.

```
u201702081@os:~/Desktop$ ./mutex
main[3428902720] : begin (counter = 0)
thread[3428898560] : begin
thread[3428898560] : done
main[3428902720] : done (counter = 0) (ma1.val = 200000000)
```

5. 스레드를 하나 더 만들고 실행하면, counter 값이 스레드 루틴 내에서 1억이 한번 더 감소하여 -1억이 된다.

```
u201702081@os:~/Desktop$ ./doublemutex
main[2461968192] : begin (counter = 0)
thread[2461964032] : begin
thread2[2453571328] : begin
thread[2461964032] : done
thread2[2453571328] : done
main[2461968192] : done (counter = -100000000) (ma1.val = 200000000)
```

- 느낀 점

  쓰레드 내부의 서브루틴 중 직접적으로 변수 증감 부분에 lock 을 걸면 race 가 발생하지 않는데,
  main 내의 쓰레드를 생성하는 pthread_create 주위에 lock / unlock 을 수행하면 race 가 발생하였다.
  범위에 차이가 있지만 결국 둘 다 공유변수의 접근을 막는 것이라고 생각하였는데, 왜 이 경우에는 race 가 발생하게 될까?

  ➔ 호출하기 전 스레드에서 lock 걸면 그 스레드에 종속되기 때문에 새로운 스레드에는 적용되지 않음