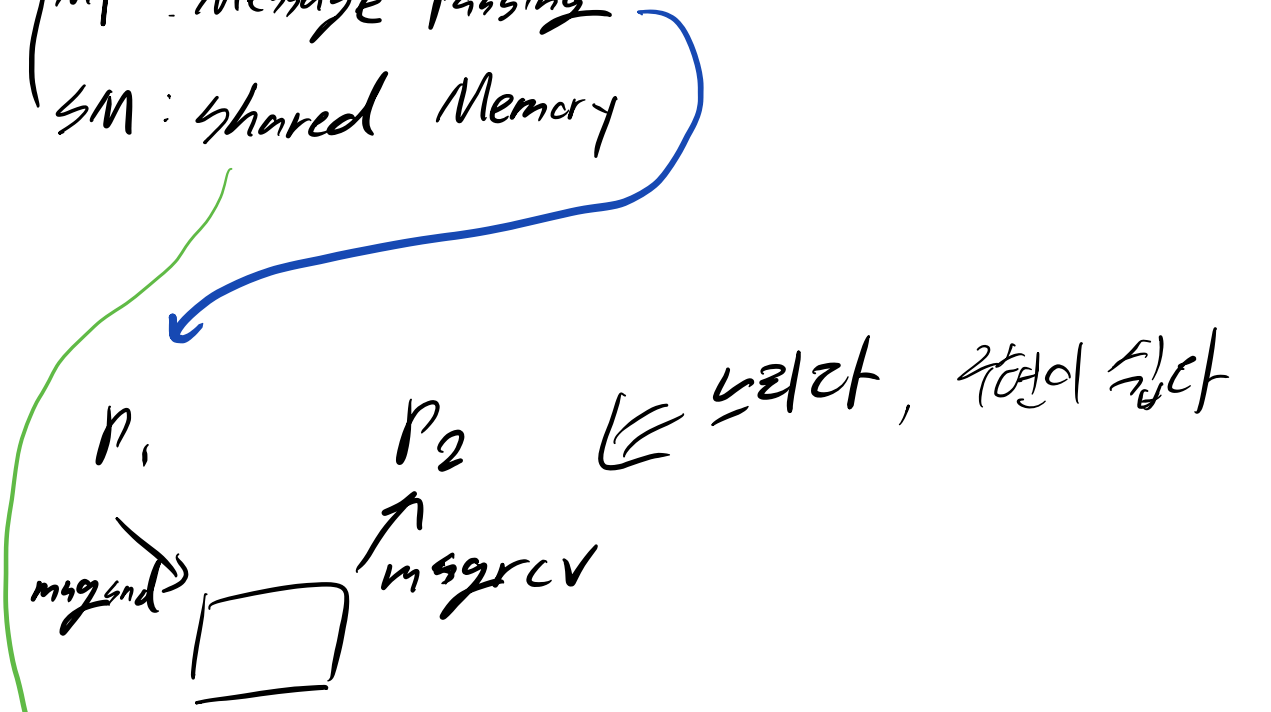
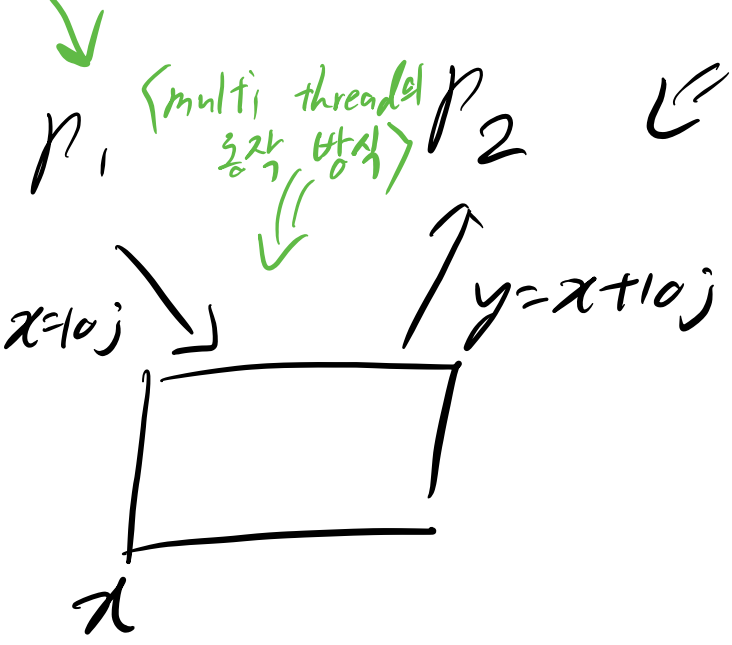
IPC (Inter Process Communication)

(MP : Message Passing

SM : Shared Memory

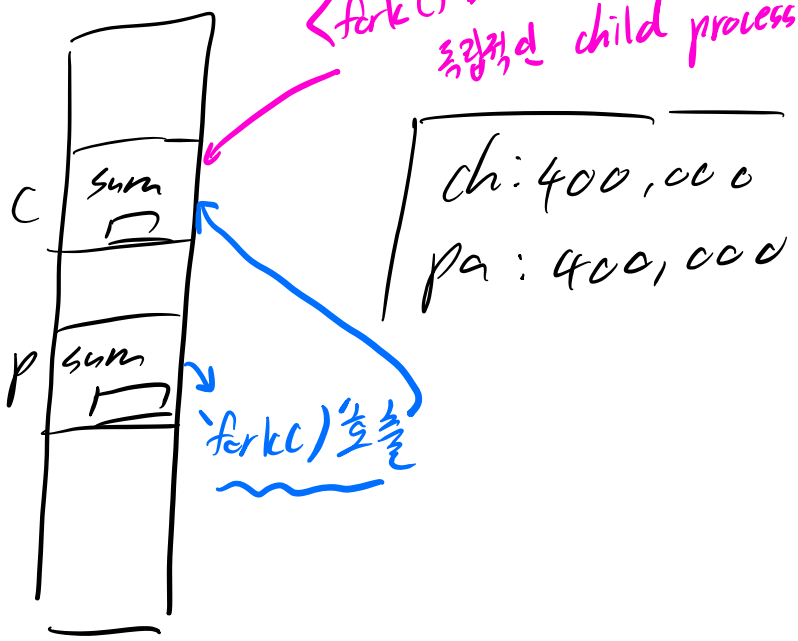


느리다, 구현이 쉽다

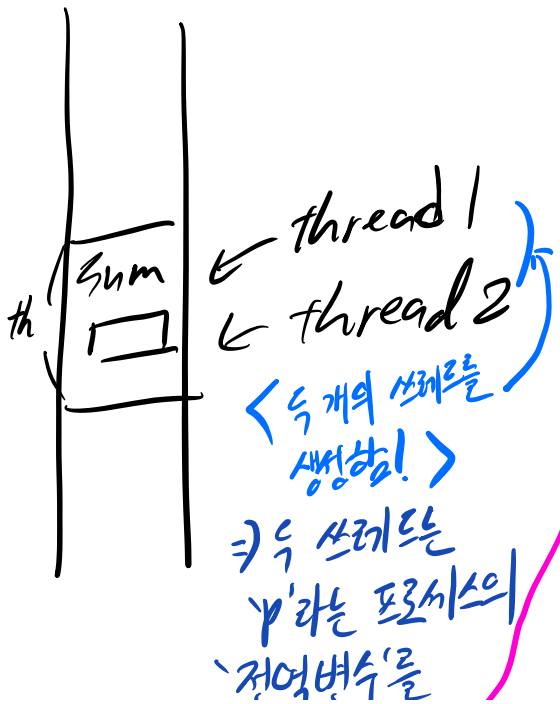


빠르다, 구현이 어렵다.

<ex1.c>



<th.c>



'thread1' (=th1) runs up to <sum=2omil>

↓

steps ('thread1'의 cpu 점유시간이 끝남.)

↓

'thread2' (=th2) runs up to <sum=50mil>

↓

stop ('thread2'의 cpu 점유시간이 끝남.)

↓

'thread1' finishes. <sum=70mil>

↓

공유한다!

`thread 2' finishes. <sum = 80mil>

th1 : 70mil
th2 : 80mil

<하지만, 절대로 이런 결과가 출력되지 않는다.>

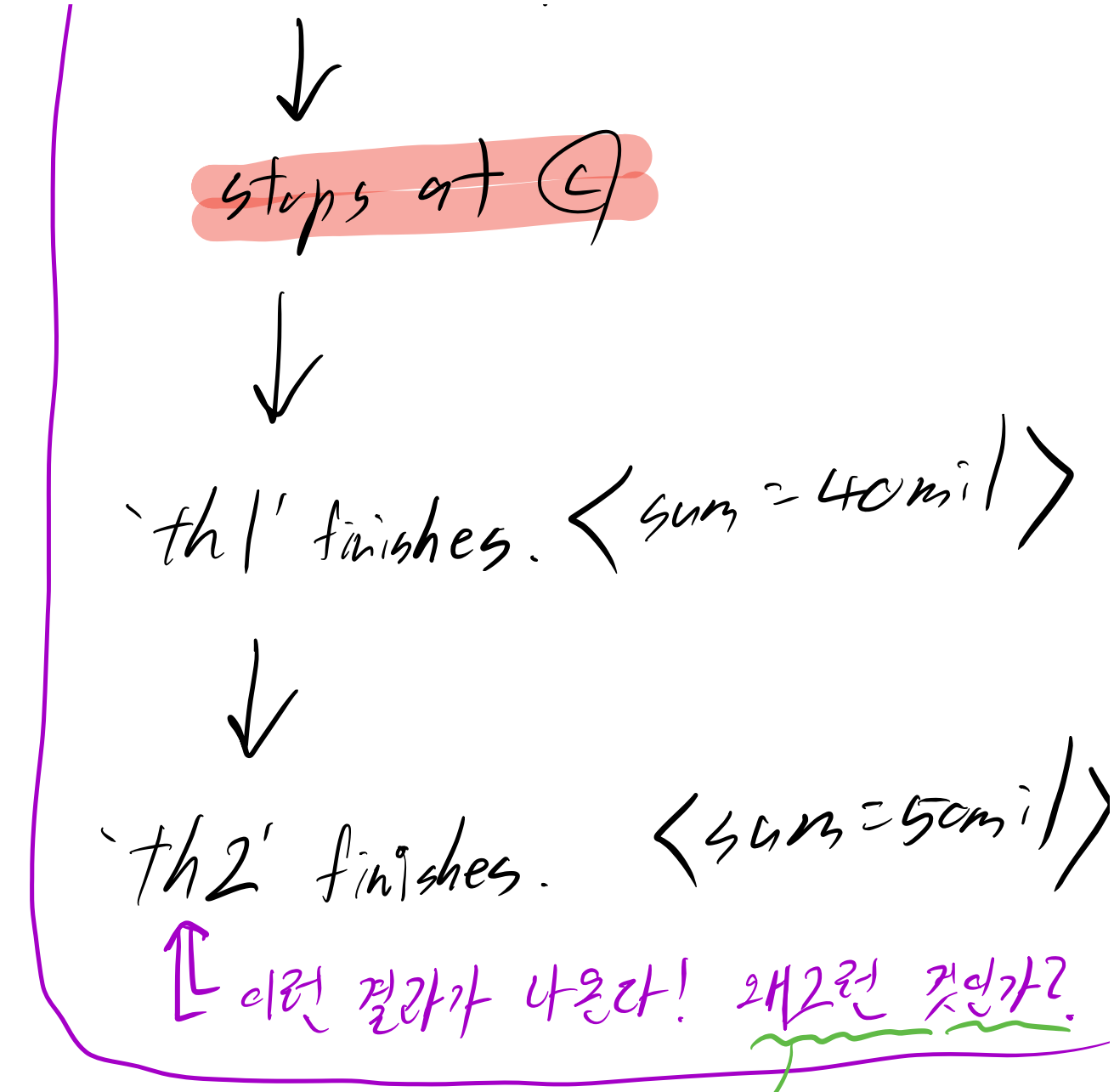
<th.c> `th1' runs up to <sum = 20mil>

↓

stops at (A) | this eax = 20mil

↓

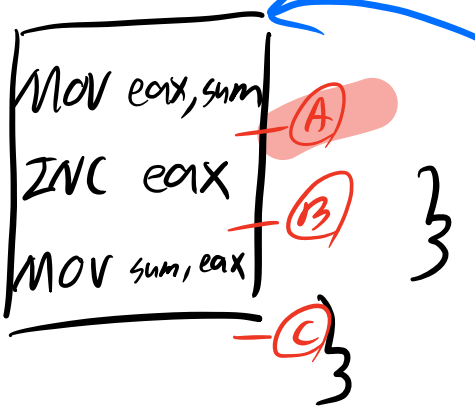
`th2' runs up to <sum = 50mil>


stops at ©
`th1' finishes. <sum = 40mil>
`th2' finishes. <sum = 50mil>
이런 결과가 나온다! 왜그런 것인가?

<th.c>

```
int sum=0;
```

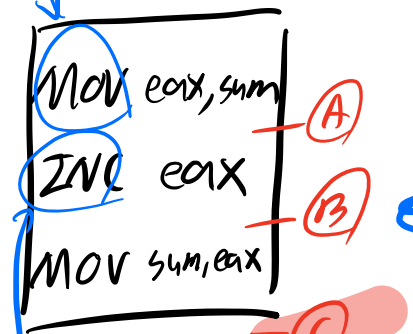
```
void *foo1(...){
    for (i=0; i<20000; i=i+1){
        for (j=0; j<20000; j=j+1){
            sum = sum + 1;
```




```
    }
}
```

```
void *foo2(...){
    for (i=0; i<20000; i=i+1){
        for (j=0; j<20000; j=j+1){
            sum = sum + 1;
```

MOV A, B: B의 값을 A에 저장해



INC A: A를 증가시켜라

```
    }
}
```

(increase)

X 의 현상을 해결하기 위한 방법.

- `semaphore(세마포)'를 통해, `mutual exclusion'을 실시한다!
- mutual exclusion : 공유변수(=전역변수)에 한 개의 쓰레드만 접근할 수 있도록 만드는 것.
  => 즉, 두 쓰레드가 동시에 특정 공유변수에 접근할 수 없다. 한 쓰레드가 특정 공유변수에 대한 접근이 끝나면, 나머지 쓰레드가 해당 변수에 접근할 수 있다.