13. Thread 동기화

Mutex

- 상호배제 (Mutual exclusion) race condition 을 막기위함.

- Mutex 변수를 통해 공유 데이터를 보호 (lock의 개념)
  오직 한 스레드만 lock 할 수 있음.
  다쓰면 unlock 해야함.

- Create, init
  - pthread_mutex_init (*mutex, attr)
    또는
  - pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
    둘 중 하나.
    init 한 것을 또 init 한 것은 어떻게 될지 모름. 꼭 한번만 하기.

- destroy
  - pthread_mutex_destroy (*mutex)
    destroy 후 참조하면 당연히 오류.
    다른 thread가 lock을 해서 사용하고 있는데 destroy 하면 당연히 안됨.

- locking, unlocking
  - pthread_mutex_lock (*초기화 된 mutex)
    mutex가 사용가능할 때까지 block 된다.

  - pthread_mutex_trylock (*mutex)

즉시 반환. (non-blocking)

- pthread_mutex_unlock (*mutex)
  다 쓴 후 unlock하기.
  lock과 unlock 사이가 critical section이된다.

- At Most Once execution
  딱 한번만 실행되어야 하는 함수가 있을 때
  - pthread_once (초기화된 pthread_once_t =PTHREAD_ONCE_INIT, void 함수(void))
    ↑ 최초의 한번만 실행됨.
  - Static 변수로 비슷하게 구현 가능.
    static int done
    static mutex
    ↑ 두번째 호출에서는 건너뛰어짐.
) 로 선언하면 프로그램이 종료되지 않는 이상 다시 초기화되지 않는 점을 이용.

↕

- At least Once execution (단, mutex 내부은 exactly once 이다)
  한번은 꼭 실행되어야 할 함수가 있을 때
  init에 필요한 경우 있음.

Condition Variable

critical section 안에서 또다른 동기화가 필요할 때. (mutex와 같이 사용)

↓

이 안에서 필요한 조건을 확인 ← 이때 공유 변수 값을 기준으로 대기, 깨우기 등

- create

- pthread_cond_init (*cond, attr)

  또는

- pthread_cond_t cond = PTHREAD_COND_INITIALIZER

- Destroy

- pthread_cond_destroy (*cond)

- Wait

- pthread_cond_wait (cond, mutex)

- pthread_cond_timedwait (cond, mutex, timespec) → 이 시간이 지나거나 signal로 깨어나면 반환함.

※ mutex를 lock해서 가진 스레드가 호출해야함.

- Signal

- Pthread_cond_signal (cond) → cond에서 대기중인 것중 최소하나를 깨움.

- Pthread_cond_broadcast (cond) → cond인 모든 대기중인 것을 깨움.

Signal delivery in threads

- pthread_kill (thread, signal) 지정한 thread에 signal을 전달

- pthread_sigmask (how, sigset, old_set) 사용법 동일

Reader, Writer

오직 reading 만 요청해도 다 가능. 하나라도 writing을 요청하면 Mutex처럼 동작.

- init
  - pthread_rwlock_init (rwlock, attr)

- destroy
  - pthread_rwlock_destroy (rwlock)

- lock, unlock
  - _rdlock
  - _tryrdlock
  - _wrlock
  - _trywrlock
  - _unlock

※ rdlock을 요청하고 wrlock을 요청하면 교체발생.
(내가 나를 기다리게 되는 상황)