<Project 3. Synchronization> 2015310884 박소현

1. Making files

- Make synch.h and synch.c files in the xv6-public folder.

- Add synch.o in Makefile

추가해 주지 않으면, undefined reference 'function' 에러가 뜬다.

- user.h와defs.h에 struct mutex\_t;와 struct cond\_t;를 정의해준다.

syscall.c와 sysproc.c에 #include "synch.h" 로 헤더 파일 추가한다.

2. synch.h

- struct mutex\_t와 struct cond\_t 의 멤버 중struct spinlock lock;을

struct spinlock \*lk;로 수정

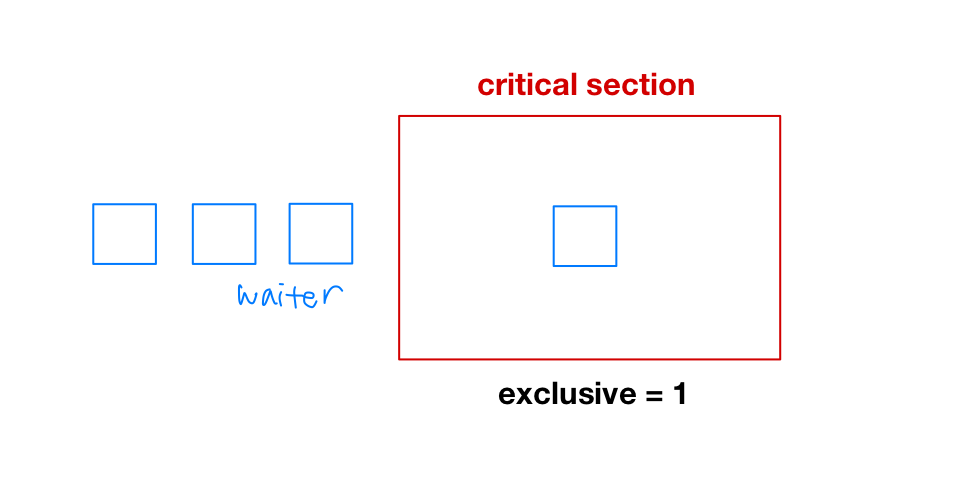
수정해주지 않는다면, field 'lock' has incomplete type 에러가 뜨며 컴파일 실패

3. synch.c

(1) mutex 구현

- 함수의 정확한 의미를 파악하자면, mutex\_lock은 뮤텍스 잠금을 요청한다는 의미이고, mutex\_unlock은 뮤텍스 잠금을 되돌려준다는 의미이다.

-mutex 원리



① int mutex\_init(struct mutext\_t \*mutex)

- mutex가 already initialized mutex라면 return -2;

- mutex->lk를 initlock 시켜준다.

- 구조체를 초기화한다.

mutex->valid = 1;

mutex->current = 0;

- 위 과정 성공시 return 0;

② int mutex\_lock(struct mutex\_t \*mutex)

- acquire(mutex->lk);로 lk에 접근 권한을 얻는다.

- mutex가 not initialized 돼 있을 경우,

release(mutex->lk); return -2;

- calling thread가 이미 mutex를 가지고 있다면,

release(mutex->lk); return -3;

- mutex lock을 요청한 thread가 요청했다고 바로 lock을 가질 수 있는 것은 아니 다.

-thread를 queue에 넣는다.

-mutex->qsize +=1;

- thread는 sleep 상태로 잠들게 한다.

여기서 if문 대신에 while문을 쓴 것을 볼 수 있다. '운영체제 아주 쉬운 세 가지

이야기' 라는 책의 내용에 따르면, 멀티 쓰레드 프로그램에서 조건을 검사할 때에는 while문을 사용하는 것이 바람직하다고 한다. 시그널 전달의 의미에 따라 if 문을 사용하는 것은 맞을 수도 있을 뿐이라 한다. while문을 사용하면 거짓으로 깨운 경우(spurious wakeup)에 대처할 수 있도록 한다. 어떤 경우에는 구현상의 문제로 하나의 시그널에 의해 두 개의 쓰레드가 깨어나는 경우가 있기도 하므로, while을 사용하여 조건을 재검사해야하는 이유이기도 하다.

- mutex->valid = 0; 으로 바꾸어주고,

mutex->current = myproc(); 현재 프로세스로 넣어준다.

- release(mutex->lk); 로 접근 권한 해제한다.

- 위 과정 성공시, return 0;

③ int mutex\_unlock(struct mutex\_t \*mutex)

- acquire(mutex->lk);로 lk에 접근 권한을 얻는다.

- mutex가 not initialized 돼 있을 경우,

release(mutex->lk); return -2;

- calling thread가 mutex를 가지고 있지 않다면,

release(mutex->lk); return -3;

- 잠들어있던 mutex->queue[0]를 깨운다.

- thread를 깨웠으니, queue의 thread들을 한 칸씩 앞으로 이동시킨다.

i번째 부분은 0으로 채운다.

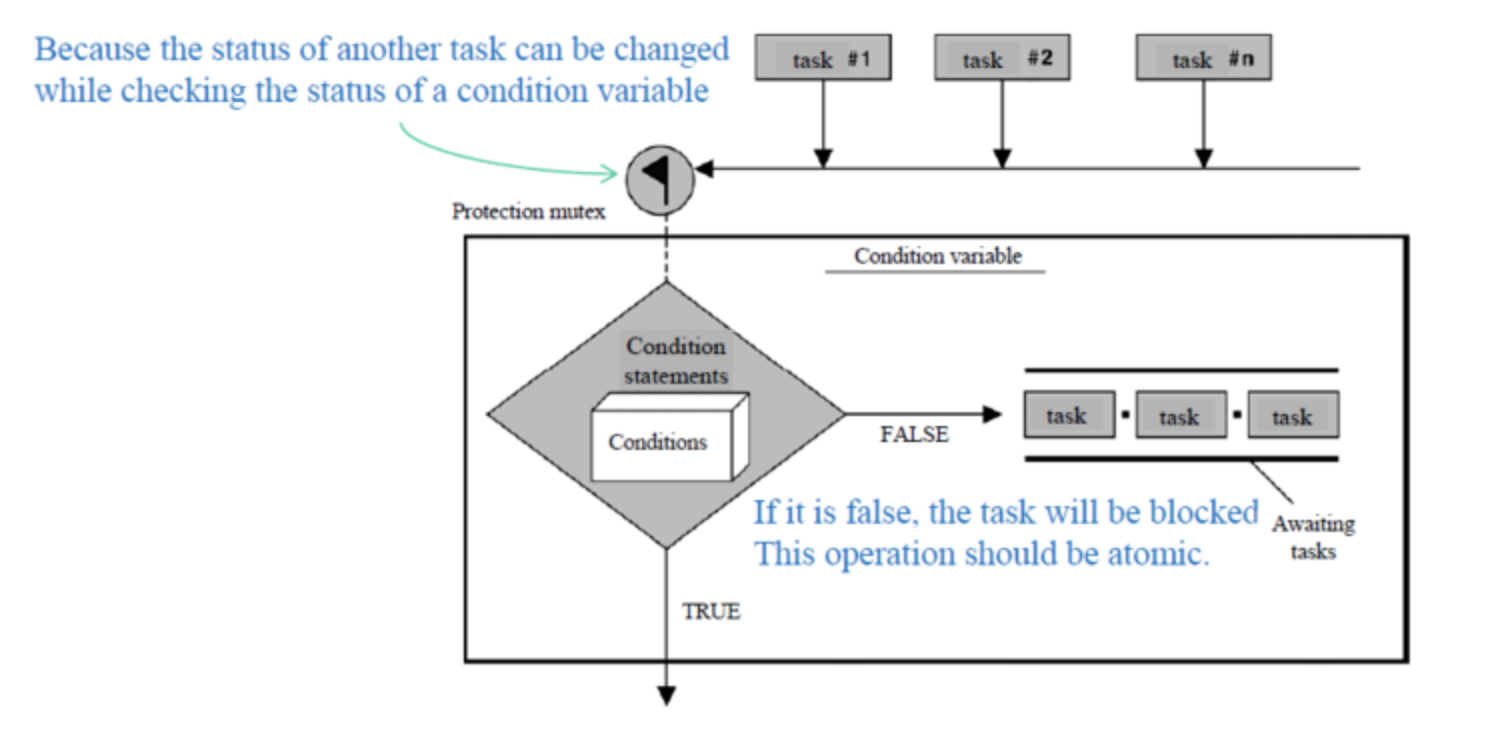
- cond->qsize -= 1;

- mutex->valid = 1; mutex->current = 0; 로 뮤텍스 잠금을 되돌려준다.

- release(mutex->lk); 로 접근 권한 해제한다.

- 위 과정 성공시, return 0;

(2) condition variable 구현

-condition variable 원리

① int cond\_init(struct cond\_t \*cond)

- 이미 initialized cv라면, return -2;

- cond->lk를 initlock 시켜준다.

- 구조체를 초기화

cond->active = 1;

cond->qsize = 1;

- 위 과정 성공시, return 0;

② int cond\_wait(struct cond\_t \*cond, struct mutex\_t \*mutex)

- acquire(cond->lk);로 lk에 접근 권한을 얻는다.

- cv나 mutex가 not initialized 됐을 경우,

release(cond->lk); return -2;

- calling thread가 mutex를 가지고 있지 않다면,

release(cond->lk); return -3;

- mutex\_unlock 해준다.

- 현재 thread를 cond->queue에 넣는다.

- cond->qsize += 1;

- thread는 sleep상태로 잠든다.

- release(cond->lk); 로 접근 권한 해제한다.

- unlock해 준 mutex를 다시 lock한다.

mutex\_lock(mutex);

- 위 과정 성공시, return 0;

③ int cond\_signal(struct cond\_t \*cond)

- acquire(cond->lk);로 lk에 접근 권한을 얻는다.

- cv가 invalid할 경우,

release(cond->lk); return -2;

- queue에서 첫번째로 들어가있는 thread를 깨운다.

- 하나의 thread를 깨웠으니, queue의 thread들을 한 칸씩 앞으로 이동시킨다.

i번째 부분은 0으로 채운다.

- cond->qsize -= 1;

- release(cond->lk); 로 접근 권한 해제한다.

- 위 과정 성공시, return 0;