tid\_t

thread\_create (const char \*name, int priority,

thread\_func \*function, void \*aux)

{

struct thread \*t;

struct kernel\_thread\_frame \*kf;

struct switch\_entry\_frame \*ef;

struct switch\_threads\_frame \*sf;

tid\_t tid;

ASSERT (function != NULL);

/\* Allocate thread. \*/

t = palloc\_get\_page (PAL\_ZERO);

if (t == NULL)

return TID\_ERROR;

/\* Initialize thread. \*/

init\_thread (t, name, priority);

tid = t->tid = allocate\_tid ();

/\* Stack frame for kernel\_thread(). \*/

kf = alloc\_frame (t, sizeof \*kf);

kf->eip = NULL;

kf->function = function;

kf->aux = aux;

/\* Stack frame for switch\_entry(). \*/

ef = alloc\_frame (t, sizeof \*ef);

ef->eip = (void (\*) (void)) kernel\_thread;

/\* Stack frame for switch\_threads(). \*/

sf = alloc\_frame (t, sizeof \*sf);

sf->eip = switch\_entry;

sf->ebp = 0;

/\* Add to run queue. \*/

thread\_unblock (t);

/\* KNU-COMP312 HINT (Priority Scheduling):

If the newly created thread has a higher priority than the currently running thread,

the current thread should yield the CPU immediately to allow the higher-priority thread to run.

Consider calling thread\_yield() after unblocking the new thread.

Similarly, you may want to apply the same priority check and preemption logic

in thread\_set\_priority() and sema\_up() in synch.c,

to ensure that a higher-priority thread can take over the CPU when appropriate. \*/

thread\_tests\_preemption();

return tid;

}

참고 ))

void

thread\_tests\_preemption (void)

{

if (!list\_empty (&ready\_list) &&

thread\_current()->priority < list\_entry( list\_front ( &ready\_list), struct thread, elem)->priority)

thread\_yield();

}

1. thread\_unblock() 을 list\_insert\_ordered() 로 수정
2. thread\_yield() 도 list\_insert\_ordered()로 수정
3. thread\_create()에 unblock 후 thread\_test\_preempbtion() 추가
4. thread\_set\_priority() 에 thread\_test\_preemption()추가
5. sema\_down() list\_insert\_ordered()로 수정
6. sema\_up() list\_sort() 후 처음 것 unblock()
7. sema\_up() 에 sema-value ++ 후에 thread\_test\_preemption()추가
8. lock\_acquire(), lock\_release() 는 수정 필요 없음
9. cond\_wait() list\_insert\_ordered( semaphore priority )
10. cond\_singal() list\_sort( semaphore priority)