En estas 2 tareas Ud. deberá implementar un controlador para lectores/escritores para nThreads. Los encabezados de la funciones pedidas son:

*nRWLock *nMakeRWLock( );*	Crea un controlador
*void nDestroyRWCtl(nRWLock *rwl);*	Destruye el controlador *rwl*
*int nEnterRead(nRWLock *rwl, int timeout);*	Solicitud de ingreso de un lector
*void nExitRead(nRWLock *rwl);*	Notificación de salida de un lector
*int nEnterWrite(nRWLock *rwl, int timeout);*	Solicitud de ingreso de un escritor
*void nExitWrite(nRWLock *rwl);*	Notificación de salida de un escritor

En la tarea 4, las funciones *nEnterRead* y *nEnterWrite* retornan siempre 1 y el parámetro *timeout* será siempre -1. La tarea 5 tiene 2 semanas más de plazo que la tarea 4. En la tarea 5, si *timeout* es -1, se espera indefinidamente. Si no, se espera a los más *timeout* milisegundos. Si se adquiere exitosamente el *rwlock* se retorna 1. Si el *timeout* expira, se retorna 0.

Para evitar hambruna se requiere programar la siguiente política para la aceptación de las solicitudes de ingreso de lectores y escritores. El principio es que los lectores y escritores ingresan alternadamente, excepto cuando los únicos threads pendientes pertenecen a solo una de estas 2 categorías.

I. Una solicitud de ingreso de un escritor (función *nEnterWrite*) se acepta de inmediato si no hay lectores o un escritor trabajando. De otro modo la solicitud queda pendiente.

II. Una solicitud de ingreso de un lector (función *nEnterRead)* se acepta de inmediato si no hay un escritor trabajando y *no hay solicitudes de escritores pendientes*. De lo contrario la solicitud queda pendiente.

III. Cuando termina de trabajar un escritor (función *nExitWrite*) y hay solicitudes de lectores pendientes, se acepta el ingreso de todos los lectores pendientes. Si no hay lectores pendientes, pero sí hay solicitudes de escritores pendientes, se acepta el ingreso del escritor que lleva más tiempo esperando.

IV. Cuando termina de trabajar un lector (función *nExitRead)* y ya no quedan otros lectores trabajando, si hay solicitudes de escritores pendientes, se acepta el ingreso del escritor que lleva más tiempo esperando.

Restricciones

Ud. debe programar las funciones pedidas en el archivo *rwlock.c* **como herramientas de sincronización nativas de nThreads**, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. Ud. no puede implementar la API solicitada en términos de otras herramientas de sincronización pre-existentes en nThreads (como semáforos, mutex o condiciones).

Ud. **debe usar 2 colas de tipo *NthQueue***: una para los escritores pendientes y la otra para los lectores pendientes. Para los estados de espera, use WAIT_RWLOCK (y WAIT_RWLOCK_TIMEOUT en la tarea 5). Ya están definidos en *nKernel/nthread-impl.h*.

Ejemplos de la solución que se espera de Ud. son la implementación de los semáforos, mutex, condiciones y mensajes de nThreads (en los archivos *nKernel/sem.c*, *nKernel/mutex-cond.c* y *nKernel/nmsgs.c*).

Instrucciones

Descargue *t4.zip* de U-cursos y descomprímalo. Los archivos adjuntos para la tarea 5 se publicarán más tarde. Ejecute el comando *make* sin parámetros en el directorio *T4* para recibir instrucciones acerca del archivo en donde debe programar su solución (*T4/rwlock.c*), cómo compilar y probar su solución, los requisitos que debe cumplir para aprobar la tarea y cómo entregar su tarea por U-cursos. Además se explica cómo puede probar sus tareas 1, 2 y 3 usando nThreads como implementación de pthreads.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *rwlock.zip* generado por *make zip*. Recuerde descargar el archivo que subió, descargar nuevamente los archivos adjuntos y volver a probar la tarea tal cual como la subió a U-cursos. Solo así estará seguro de no haber entregado archivos incorrectos. Se descuenta medio punto por día de atraso. No se consideran los días de receso, sábado, domingo o festivos.