Esta tarea es una continuación de la tarea 4. Ahora deberá implementar el parámetro *timeout* para nPedir.

Esto implica que el nThread solicita el recurso compartido y lo esperara por *timeout* milisegundos antes de retornar.

Si *nPedir* recibe el recurso antes del *timeout*, *nPedir* debe retornar 1. Si se excede el *timeout*, *nPedir* retorna 0 de inmediato, sin recibir el recurso.

Instrucciones

Baje *t5.zip* de U-cursos y descomprímalo. Copie el archivo *pedir.c* de su solución de la Tarea4 al directorio *T5* contiene los archivos *test-pedir.c*, *Makefile*, *pedir.h* (con los encabezados de las funciones requeridas) y otros archivos.

Modifique *pedir.c* para que nPedir y nDevolver manejen de forma adecuada el parámetro timeout.

Ejecute el comando *make* sin parámetros en el directorio T5 para recibir instrucciones adicionales sobre cómo compilar y probar su solución, los requisitos que debe cumplir para aprobar la tarea y cómo entregar su tarea por U-cursos.

Requerimientos

1) Ud. debe programar las funciones pedidas en el archivo *pedir.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).

2) Ud. **debe usar 2 colas del tipo NthQueue** para encolar los nthreads en espera. Los encabezados de las funciones que puede usar están en *nKernel/nthread.h* y *nKernel/nthread-impl.h*. Si necesita guardar información asociada a los nthreads en espera, pida memoria con *malloc* y asígnela al campo *ptr* (de tipo void*) en el descriptor del nthreads.

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*).

Ayuda

Necesita distinguir entre nThreads que invocaron *nPedir* con *timeout* (*timeout*>0) y los que lo hicieron sin *timeout* (*timeout*<0). Para los primeros use el estado WAIT_REQUEST_TIMEOUT. Para los segundos use el estado WAIT_REQUEST. Es importante porque cuando se invoca *nDevolver* un nThread en espera del recurso compartido con *timeout* está configurado para despertarse con *nth_programTimer*. Si su estado continúa en WAIT_REQUEST_TIMEOUT su *timeout* debe cancelarse con *nth_cancelThread* (vea la clase auxiliar de mensajes con timeout).

Observe que *nth_programTimer* recibe el *timeout* en nanosegundos, mientras que *nPedir* recibe el *timeout* en milisegundos. Para convertir timeout en milisegundos a nanosegundos, use la expresión *timeout**1000000**LL** para que el resultado sea de tipo long long, de otro modo el tipo sería *int* y habría desborde. ¡Use el sufico **LL**!

Cuando el *timeout* expira para un nThread que invocó *nPedir*, ese nThread todavía está en alguna cola con los nThreads en espera , tendrá que eliminarlo de la o las colas que correspondan.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *pedir.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, sábados, domingos o festivos.