Esta tarea es una continuación de la tarea 4. Ahora deberá implementar el parámetro *timeout* para *nRequestDisk*. Se modifica el valor retornado por *nRequestDisk*. El nuevo encabezado es:

int *nRequestDisk(int track, int timeout);*

Esta función solicita acceso al disco indicando la pista. Si *timeout* es menor que 0, *nRequestDisk* espera indefinidamente hasta obtener el disco, retornando el valor 0. En caso contrario espera hasta un máximo de *timeout* milisegundos. Si después de este tiempo el disco sigue ocupado, esta función retorna de inmediato el valor 1. Si el disco es otorgado antes de ese tiempo, la función retorna 0.

Instrucciones

Descargue *t5.zip* de U-cursos y descomprímalo. Copie su archivo *disk.c* con la solución de su tarea 4 a este mismo directorio. Modifique *disk.c* implementando el parámetro *timeout* de *nRequestDisk*. Ejecute el comando *make* sin parámetros en el directorio *T5* para recibir instrucciones sobre cómo compilar y probar su solución, los requisitos que debe cumplir para aprobar la tarea y cómo entregar su tarea por U-cursos.

Ayuda

Como en la tarea 4, use 2 colas de prioridad de tipo *PriQueue*, definida en *nKernel/pss.h*.

Necesita distinguir entre nthreads que invocaron *nRequestDisk* 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 *nReleaseDisk* un nthread en espera en un *nRequestDisk* 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 *nRequestDisk* 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 *nRequestDisk*, ese nthread todavía está en alguna cola de prioridad con los nthreads en espera. Tendrá que eliminarlo de ambas. Para lograrlo, siga cuidadosamente las siguientes instrucciones. Necesitará leerlas varias veces. Un error le costará mucho tiempo en depuración. En *nRequestDisk* coloque la dirección de *track* en el campo *ptr* del descriptor de ese nthread (servirá solo como una marca, puede ser cualquier dirección no nula). Al programar el timeout con *nth_programTimer*, suministre en el segundo parámetro una función *f* que hará la eliminación. Esa función *f* recibe como parámetro el descriptor del nthread cuyo timeout expiró. En el campo *ptr* estará el puntero a *track*. Elimine el nthread de ambas colas con la función *delPri* que elimina un objeto sin importar su prioridad. Esta función está declarada en nKernel/pss.h. **Además deje el campo *ptr* en NULL**. Si después de llamar a *schedule*, el campo *ptr* es NULL, quiere decir que el timeout expiró y por lo tanto *nRequestDisk* debe retornar 1. Si no es NULL, *nRequestDisk* debe retornar 0 porque sí obtuvo el recurso compartido.

Tenga cuidado porque la función *f* se invoca dentro de una rutina de atención de señales. Haga cosas simples en esa función.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *disk.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.