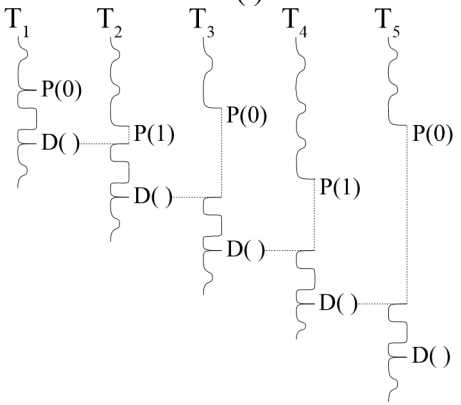
Se dispone de un recurso único compartido entre varios threads. Los threads se agrupan en categoría 0 y categoría 1. Se dice que 0 es la categoría opuesta de 1, y 1 la categoría opuesta de 0. Un thread de categoría *cat* solicita el recurso invocando la función *nPedir(cat, -1)* y devuelve el recurso llamando a la función *nDevolver( )*, sin parámetros. Los encabezados para ambas funciones son:

```
int nPedir(int cat, int timeout);
void nDevolver();
```

Programe las funciones *nPedir* y *nDevolver* garantizando la exclusión mutua al acceder al recurso compartido. Se requiere una política de asignación alternada primero y luego por orden de llegada. Esto significa que cuando un thread de categoría *cat* devuelve el recurso, si hay algún thread en espera de la categoría opuesta a *cat*, el recurso se asigna inmediatamente al thread de categoría opuesta que lleva más tiempo esperando. Si no hay threads en espera de la categoría opuesta pero sí de la misma categoría *cat*, el recurso se asigna al thread que lleva más tiempo en espera. Si no hay ningún thread en espera, el recurso queda disponible y se asignará en el futuro al primer thread que lo solicite, cualquiera sea su categoría. El siguiente diagrama muestra un ejemplo de asignación del recurso. La invocación de *nPedir* se abrevió como P(…) y la de *nDevolver* como D( ).



Inicialice las variables globales en la función *nth_iniciar* y libere los recursos solicitados en *nth_terminar*.

En esta tarea el parámetro *timeout* de *nPedir* será siempre -1 y no importa lo que retorne *nPedir*. En la tarea 5, este parámetro indica el tiempo máximo de espera. 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.

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*).

Instrucciones

Baje *t4.zip* de U-cursos y descomprímalo. El directorio *T4* contiene los archivos *test-pedir.c*, *Makefile*, *pedir.h* (con los encabezados de las funciones requeridas) y otros archivos. Ejecute el comando *make* sin parámetros en el directorio T4 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.

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.