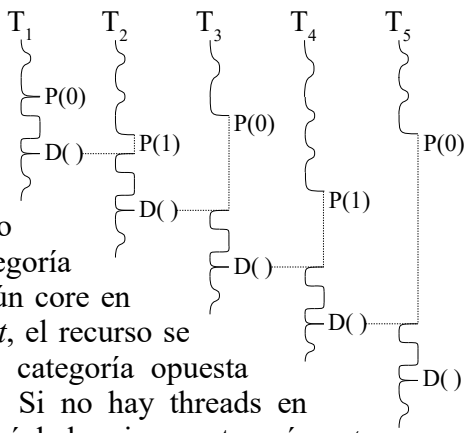
Se dispone de un recurso único compartido por los múltiples cores de una máquina *sin sistema operativo*. Los cores se agrupan en categoría 0 y categoría 1. Se dice que 0 es la categoría opuesta de 1, y 1 la categoría opuesta de 0. Un core de categoría *cat* solicita el recurso invocando la función *pedir(cat) y d*evuelve el recurso llamando a la función *devolver( )*, sin parámetros.

Se necesita programar las funciones *pedir* y *devolver* 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 core de categoría *cat* devuelve el recurso, si hay algún core en espera de la categoría opuesta a *cat*, el recurso se asigna inmediatamente al core 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 core que lleva más tiempo en espera. Si no hay ningún core en espera, el recurso queda disponible y se asignará en el futuro al primer core que lo solicite, cualquiera sea su categoría. El diagrama de arriba muestra un ejemplo de asignación del recurso. La invocación de *pedir* se abrevió como P(…) y la de *devolver* como D( ).




Programe las funciones *pedir, devolver*, *iniciar* y *terminar* con encabezados:

```
void iniciar();
void terminar();
void pedir(int cat);
void devolver();
```

Dado que la máquina no posee un sistema operativo, la única herramienta de sincronización disponible son los spin-locks. Ud. sí dispone del tipo *Queue*. También dispone de *coreId( )* para determinar la identificación del core, pero no le será útil en esta tarea. No necesita saber que la máquina tiene 8 cores.

Dado que no hay sistema operativo, no puede usar funciones como *malloc*, *pthread_mutex_lock*, *sem_wait*, etc. Puede usar *printf* para fines de depuración pero no olvide eliminarlos antes de entregar su tarea.

Inicialice las variables globales en la función *iniciar* y libere los recursos solicitados en *terminar* (como las colas de cores en espera).

La metodología que para resolver problemas de sincronización con spin-locks, como esta tarea, es la misma que se enseñó para resolver problemas de sincronización con semáforos.

## *Instrucciones*

Descargue de U-cursos el archivo *t6.zip* y descomprímalo. El directorio *T6* contiene el archivo de encabezados para las funciones pedidas (*pedir.h*), la implementación de los spin-locks (*spin-locks.h*, *spin-locks.c* y *swap.s*), la biblioteca de PSS (*pss.h* y *pss.c*) que implementa el tipo *Queue*, el programa de prueba (*test-pub.c*) y el *Makefile* para compilar el programa de prueba.

Ejecute el comando *make* sin parámetros bajo Debian 11. Le explicará en qué archivo debe resolver esta tarea, qué requisitos debe cumplir para aprobar su tarea, cuáles son las opciones de compilación y ejecución, cómo entregar su tarea y cómo borrar los archivos intermedios.

*Observación*: El programa de prueba emula 8 cores virtuales con 8 pthreads. No necesita correr el test en una máquina con 8 cores reales, pero los spin-locks son muy ineficientes en este ejemplo cuando se ejecuta en una máquina con menos cores. Si el round-robin de Linux le quita la CPU al core virtual que adquirió el spin-lock, los demás cores virtuales que compiten por el mismo spin-lock se quedaran inútilmente en el ciclo de busy-waiting hasta que Linux le devuelva la CPU al poseedor del spin-lock. Eso puede tomar bastante tiempo. Para evitar el problema, en un núcleo real se inhiben las interrupciones cuando se solicita un spin-lock y así no se pierde la CPU. El programa de prueba no puede inhibir las interrupciones de Linux. Pero reitero, *en esta tarea no hay un requisito en el número de cores reales de la máquina en donde corre el test.*

## *Entrega*

Ud. debe entregar mediante U-cursos un archivo *pedir.zip* generado con el comando *make zip* que incluye los archivos *resultados.txt* y *pedir.c* con su solución. **Recuerde descargar el archivo que subió, descomprimirlo e inspeccionar el contenido para verificar que son los archivos correctos.** Su tarea será rechazada si el comando *make zip* falla. Se descontará medio punto por día hábil de atraso.