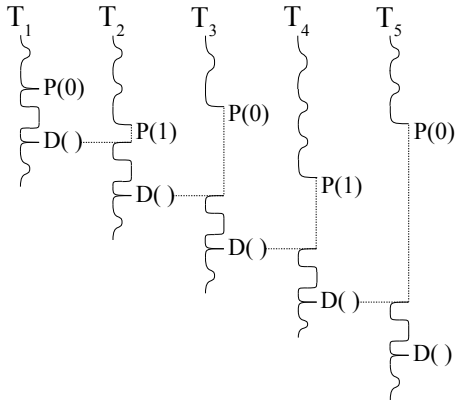
CC4302 Sistemas Operativos – Tarea 3 – Semestre Primavera 2024 – Profs.: Mateu, Torrealba, Arenas

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 *pedir(cat)* y devuelve el recurso llamando a la función *devolver( )*, sin parámetros. Los encabezados para ambas funciones son:

```
void pedir(int cat);
void devolver(int id);
```

Programe 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 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 *pedir* se abrevió como P(…) y la de *devolver* como D( ).



Ud. debe resolver este problema de sincronización usando un mutex y múltiples condiciones de pthreads. Debe evitar cambios de contexto inútiles y por lo tanto deberá usar el patrón *request*. Necesitará usar variables globales y 2 colas fifo (tipo Queue), una para los threads de categoría 0 y la otra para la categoría 1. Inicialice las variables globales en la función *iniciar* y libere los recursos solicitados en *terminar* (como las colas de threads en espera).

Instrucciones

Baje *t3.zip* de U-cursos y descomprímalo. El directorio *T3* contiene los archivos *test-pedir.c*, *Makefile*, *pedir.h* (con los encabezados requeridos) y otros archivos. Ud. debe programar en el archivo *pedir.c* las funciones solicitadas. Defina otras funciones si las necesita.

Descargue *t3.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros en el directorio *T3* para recibir instrucciones acerca del archivo en donde debe programar su solución (*pedir.c*), cómo compilar, probar y depurar su solución, los requisitos que debe cumplir para aprobar la tarea y cómo entregar su tarea por U-cursos.

Pruebe su tarea bajo Debian 12. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo por aprobar este modo de ejecución.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún incidente en el manejo de memoria.
- *make run-thr debe felicitarlo y no reportar ningún datarace.*

Cuando pruebe su tarea con *make run* en su computador asegúrese de que esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo.

Invoque el comando *make zip* para ejecutar todos los tests y generar un archivo *pedir.zip* que contiene *pedir.c*, con su solución, y *resultados.txt*, con la salida de *make run*, *make run-g*, make *run-san* y *make run-thr*.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *pedir.zip* generado por *make zip*. Este incluye *pedir.c* y *resultados.txt*. 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.