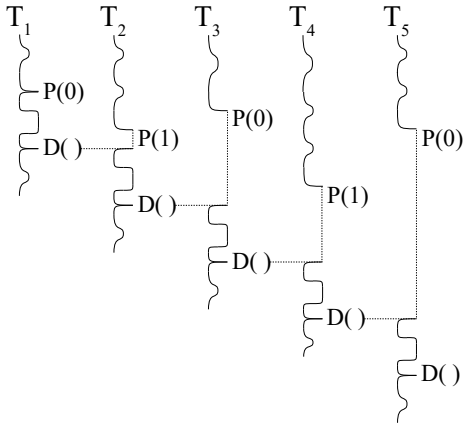
CC4302 Sistemas Operativos – Control 2 – Semestre Primavera 2023 – Prof.: Luis Mateu

Pregunta 1

Parte a.- (4 puntos) 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:

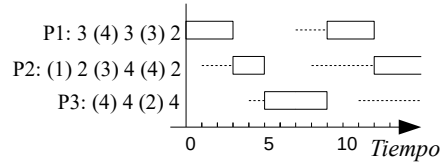
```
void pedir(int cat);
void devolver(int id);
```

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.

Parte b.- (2 puntos)



El diagrama parcial muestra las decisiones de scheduling para 3 procesos. Para cada proceso se indica la duración de la ráfaga y la duración de su estado de espera entre paréntesis. En el diagrama la línea punteada indica que el estado del proceso es READY. Si el proceso está en estado de espera el espacio aparece en blanco. ¿De qué estrategia de scheduling se trata? Rehaga el diagrama completo considerando ahora la estrategia *shortest job first* non preemptive. Considere que el predictor de duración para la próxima ráfaga es la duración de la última ráfaga.

Pregunta 2

Programe las funciones *nPedir* y *nDevolver* como herramienta de sincronización nativa de nThreads con los mismos requerimientos de las funciones *pedir* y *devolver* de la parte *a* de la pregunta 1. Los encabezados de las funciones son:

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

Ud. debe resolver este problema de sincronización usando los procedimientos de bajo nivel de nThreads (*START_CRITICAL*, *END_CRITICAL*, *schedule*, *setReady*, *nth_putBack*, etc.). Ud. no puede usar otros mecanismos de sincronización ya disponibles en nThreads como semáforos, mutex, mensajes, etc. Necesitará usar variables globales.