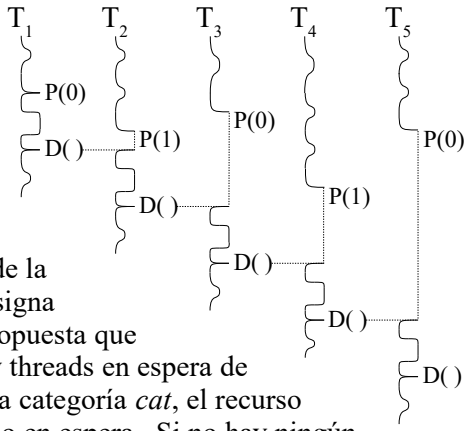
Pregunta 1

Parte a.- Se dispone de un recurso único compartido entre varios threads de nSystem. 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 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 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 diagrama de arriba muestra un ejemplo de asignación del recurso. La invocación de *pedir* se abrevió como P(…) y la de *devolver* como D( ).

La siguiente implementación *incorrecta* usa semáforos para implementar pedir y devolver:

```
nSem s[2]; // = nMakeSem(0) x 2
int cnt[2]= {0, 0};
// cnt[cat] es el n° de threads de
// categoría cat en espera.
int ocup= -1; //-1: libre
// 0 o 1: ocupado por thread de cat 0 o 1
void pedir(int cat) {
  if (ocup==-1)
    ocup= cat;
  else {
    cnt[cat]++;
    nWaitSem(s[cat]);
  }
}
```

```
void devolver() {
  if (cnt[!ocup]>0) {
    cnt[!ocup]--;
    nSignalSem(s[!ocup]);
    ocup= !ocup; //¡Atención!
  }
  else if (cnt[ocup]>0) {
    cnt[ocup]--;
    nSignalSem(s[ocup]);
    // ¡ocup se mantiene!
  }
  else
    ocup= -1;
}
```

Observe que los tickets de los semáforos de nSystem son otorgados por orden de llegada. Además !0 es 1 y !1 es 0. La solución funciona el 99.9% de los casos pero posee dataraces debido a 2 errores. El primero es fácil. El segundo error se debe a la línea con el llamado de atención. Ejemplifique el segundo error con un diagrama de threads que mueste que si hace un rato que un thread de categoría 0 ocupa el recurso y esperan 2 threads de categorías 0 y 2 threads de categoría 1, cuando se libera el recurso, el orden de atención de los 5 threads podría ser incorrectamente 0, 1, 1, 0, 0.

Parte b.- Corrija los 2 errores de la solución usando exactamente 3 semáforos. No puede usar otras herramientas de sincronización, ni *fifoqueues*.

Parte c.- Reprograme la solución reemplazando los semáforos por los monitores y condiciones de nSystem. Debido a que *nSignalCondition* no respeta el orden de llegada, debe crear una condición por cada thread en espera. Necesitará usar variables globales y *fifoqueues*. ¿Cómo se inicializan?

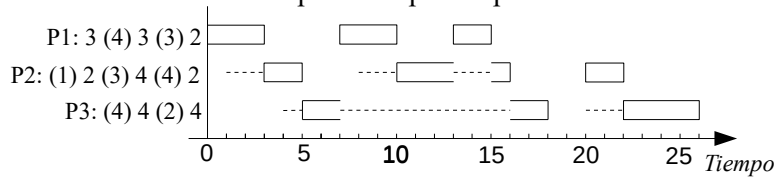
Pregunta 2

I. (4 puntos) Se desea agregar las operaciones *nPedir* y *nDevolver* como herramienta de sincronización nativa de nSystem con los mismos requerimientos de las funciones *pedir* y *devolver* de la pregunta 1. Los encabezados de las funciones son:

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

Programe estas funciones usando los procedimientos de bajo nivel de nSystem (*START_CRITICAL*, *Resume*, *PutTask*, etc.). Ud. no puede usar otros mecanismos de sincronización ya disponibles en nSystem como semáforos, monitores, mensajes, etc. Necesitará usar variables globales. Señale cómo se inicializan. No puede usar *fifoqueues*. Debe usar las colas de threads de nSystem (tipo *Queue*, el mismo de la *ready_queue*).

II. (2 puntos) El diagrama muestra las decisiones de scheduling para 3 procesos. A la izquierda se indica para cada proceso las duraciones de sus ráfagas de CPU y entre paréntesis las duraciones de sus estados de espera. 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.



Rehaga el diagrama considerando ahora la estrategia *shortest job first non preemptive*. El predictor para la duración de la ráfaga es la duración de la última ráfaga. Lo que ocurre antes de tiempo=6 es lo mismo que aparece en el diagrama original.