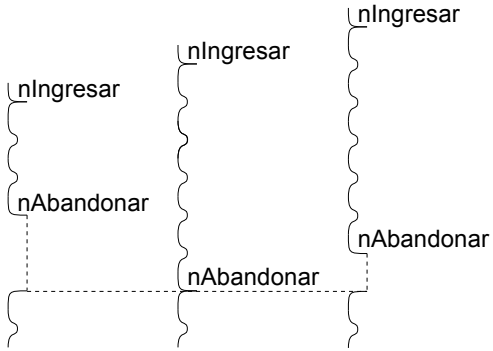
Pregunta 1

Parte a.- (4 puntos) Se requiere implementar en nSystem un sistema de reuniones. Las funciones son las siguientes:

- *void nIngresar( )*: notifica el ingreso a la reunión.
- *void nAbandonar( )*: solicita finalizar la reunión. Se bloquea en espera hasta que todas las tareas que hayan ingresado hayan solicitado finalizar la reunión. Esto incluye aquellas tareas que ingresen después de esta invocación de *nAbandonar*.

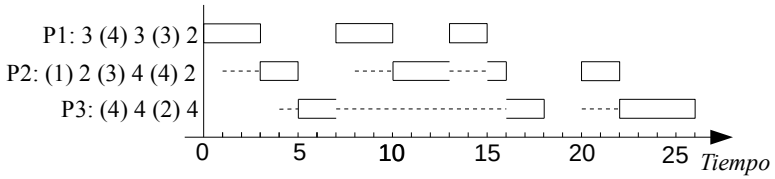
La siguiente figura muestra con un diagrama de threads un ejemplo de uso.



Observe que cuando se invoca el último *nAbandonar* todas la tareas pasan a estado READY.

Implemente esta API 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. Ud. podría necesitar agregar nuevos campos al descriptor de tarea de nSystem, nuevos estados, nuevas variables globales, etc.

Parte b.- (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.

¿De qué estrategia de scheduling se trata?

Rehaga el diagrama completo considerando ahora la estrategia *round robin* con tajadas de 5 unidades de tiempo, marcando en el diagrama los instantes en que un proceso agota su tajada. Considere que cuando un proceso pasa a estado de espera se le descuenta de su tajada el tiempo que ocupó de CPU. Al pasar a estado READY recupera la CPU. El proceso al que se le quitó la CPU queda en primer lugar en la cola. Cuando un proceso agota su tajada se va al final de la cola y reinicia su tajada en 5 unidades de tiempo. Los 3 procesos tienen una tajada inicial de 5 unidades de tiempo.

Pregunta 2

I. (4 puntos) Resuelva el mismo problema de la pregunta 1 considerando ahora una máquina *octa-core* en la que no existe un núcleo de sistema operativo y por lo tanto la única herramienta de sincronización preexistente es el *spin-lock*. Las funciones que Ud. debe implementar son las siguientes:

```
void ingresar();
void abandonar();
```

Para programar su solución Ud. dispone de *spin-locks* y la función *coreId()* que entrega el número del core que la invoca. Necesitará declarar variables globales.

Restricción: Dado que no hay un núcleo de sistema operativo, la única forma válida de esperar el fin de la reunión es utilizando un spin-lock. Otras formas de *busy-waiting* no están permitidas. No hay *malloc*, *fifoqueues*, *LLMutex*, cola de procesos "ready", etc.

II. (1 punto) ¿Cuál es la ventaja número 1 de un núcleo clásico por sobre un núcleo moderno? ¿Y cuál es la desventaja número 1?

III. (1 punto) Suponga que Ud. necesita incrementar una variable global al programar código de un núcleo de Unix para una máquina multicore. Explique si será necesario asegurar la exclusión mutua y cómo lo haría de la manera más eficiente. Responda una vez considerando un núcleo clásico y una segunda vez considerando un núcleo moderno.