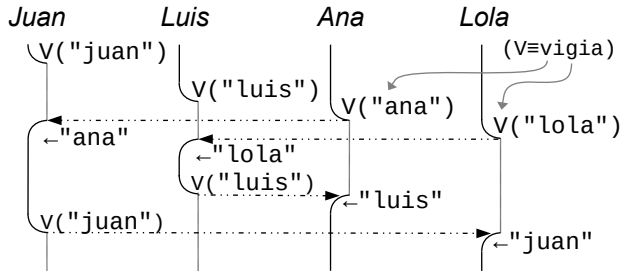
CC4302 Sistemas Operativos – Control 2 – Semestre Otoño 2026
Profs.: Mateu, Torrealba

Pregunta 1

```
char *vigia(char *nom);
```

A) (4 puntos) En el Titanic se necesita que siempre hayan 2 vigías en busca de icebergs. Los vigías son representados por threads que llaman cada cierto tiempo a la función *vigia*, con el encabezado del cuadro de la derecha. El parámetro *nom* es el nombre del vigía. Cuando un thread *T* invoca *vigia*, *T* se suspende hasta que sea relevado luego de que otros 2 threads llamen a *vigia*. Mientras tanto *T* actúa como vigía junto al thread que había llamado previamente a *vigia* y más tarde junto al próximo thread que llame a *vigia*. Esta función debe retornar el nombre del thread que lo relevó. El diagrama de threads de la derecha muestra un ejemplo de ejecución. Observe que el primer thread que invoca *vigia* (Juan) es relevado por Ana, Luis por Lola, Ana por Luis y Lola por Juan.



Programe la función *vigia* usando un solo mutex y múltiples condiciones de pthreads. Debe evitar cambios de contexto inútiles y por lo tanto necesitará usar el patrón request. Declare una función adicional para inicializar las variables globales que requiera.

Ayuda: Agregue un campo adicional a la estructura request para depositar ahí el nombre del thread que releva al thread suspendido. Use una cola global de tipo *Queue* para hacer esperar a los 2 threads que actúan como vigías. La función *queueLength* entrega el largo de la cola.

B) (1 punto) ¿Cuantos timers se necesitan en una máquina octa-core para implementar el scheduler round-robin del núcleo de un sistema operativo? Explique por qué.

C) (1 punto) Un programador propone hackear un sistema Unix de la siguiente manera. El sabe en qué dirección de la memoria está el vector de interrupciones. Entonces él propone cambiar la función que ejecuta las llamadas al sistema por su propia función. Explique si puede o no lograr hackear la máquina.

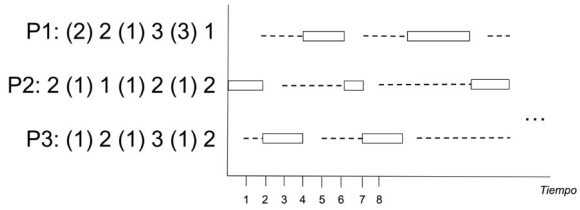
Pregunta 2

```
char *nVigia(char *nom);
```

I) (4 puntos) Programe la función del cuadro de la derecha con la misma funcionalidad de la función de la parte A de la pregunta 1, como herramienta de sincronización nativa de nThreads, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. No puede usar otras herramientas de sincronización preexistentes como mutex, condiciones o semáforos. Para los threads en espera debe usar una cola del tipo *NthQueue* y como estado de espera WAIT_VIGIA. Declare una función adicional para inicializar las variables globales que requiera.

Ayuda: Use el campo *ptr* en el descriptor de thread para almacenar el nombre del vigía que releva a otro.

II) (2 puntos) El diagrama de abajo 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? ¿Es *preemptive*? Justifique su respuesta. Rehaga el diagrama completo considerando ahora la estrategia de prioridades *non preemptive*, donde P1 posee mejor prioridad que P2 y P2 mejor prioridad que P3.