CC4302 Sistemas Operativos – Control 2 – Semestre Primavera 2025
Profs.: Mateu, Arenas

Pregunta 1

Muchos perros y gatos necesitan pasar por un puente para cruzar un río. Como perros y gatos se odian, no puede haber perros y gatos simultáneamente en el puente. Para mayor eficiencia, se requiere que perros y gatos, crucen en grupos de la misma especie y por turnos alternados. Los perros solicitan cruzar invocando *cruzaPerro()* y notifican la salida del puente con *salePerro()*. Análogamente los gatos invocan *cruzaGato()* y *saleGato()*. Para hacer un uso eficiente del puente, sin introducir hambruna, Ud. debe programar la política de ingresos al puente señalada a continuación. Cada regla tiene su duplicado, cambiando la mascota por la mascota que aparece entre paréntesis.

- Si un perro (gato) solicita cruzar y no hay ningún gato (perro) esperando, el perro (gato) ingresa al puente de inmediato.
- Si un perro (gato) solicita cruzar y hay gatos (perros) cruzando, el perro (gato) debe esperar.
- Si hay gatos (perros) cruzando el puente y perros (gatos) esperando cruzar, si un nuevo gato (perro) solicita cruzar, deberá esperar hasta que crucen los perros (gatos) en espera.
- Si el último gato (perro) sale del puente y hay perros (gatos) esperando, se autoriza el ingreso de todos los perros (gatos) en espera.

Observe que si no hay perros esperando cuando sale el último gato, es imposible que hayan gatos en espera. **Programe** estas 4 funciones con los siguientes requerimientos: (i) Para la sincronización Ud. debe usar un solo mutex y múltiples condiciones de pthreads. (ii) Para evitar cambios de contexto inútiles debe usar el patrón request. (iii) Use 2 colas fifo (tipo Queue), una para los perros en espera y otra para los gatos en espera. Declare una función adicional para inicializar las variables globales que requiera.

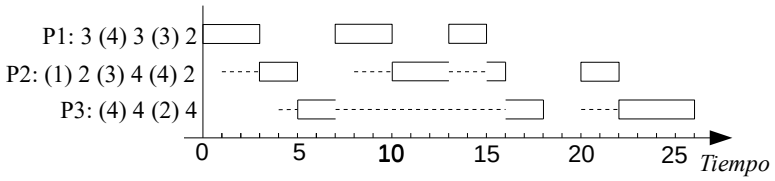
Pregunta 2

Programe las funciones del cuadro de la derecha con la misma funcionalidad y política de ingresos de las funciones de la pregunta 1, como herramientas de sincronización nativas 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 use colas del tipo Queue o NthQueue y como estado de espera WAIT_PUENTE. Declare una función adicional para inicializar las variables globales que requiera.

```
void nCruzaPerro();
void nSalePerro();
void nCruzaGato();
void nSaleGato();
```

Pregunta 3

I. (3 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? Rehaga el diagrama completo considerando ahora la estrategia *first come first served*.

II. (1,5 puntos) Explique cuál es la principal razón por la que un núcleo clásico de un sistema operativo no funciona en máquinas multi-core. ¿Qué cambio minimal hay que hacer al núcleo clásico para hacer que sí funcione correctamente en máquinas multi-core, logrando que los cores ejecuten procesos en paralelo?

III. (1,5 puntos) 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* el sistema.