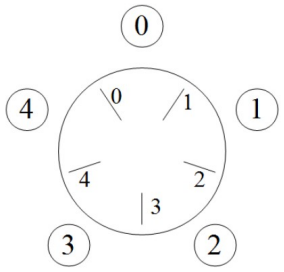
## Pregunta 1

```
void pedir(int id);
void devolver(int id);
void init( );
void filosofo(int id) {
  for (;;) {
    pedir(id);
    comer(id,(id+1)%5);
    devolver(id);
    pensar();
  }
}
```

***Parte i.-*** A la derecha se muestra una solución parcial de la cena de filósofos, en donde 5 filósofos numerados de 0 a 4 cenan juntos en una mesa redonda en donde hay 5 palitos, numerados de 0 a 4. Cada filósofo alterna entre comer y pensar. Para poder comer el filósofo *id* necesita los palitos *id* y *(id+1)%5*. No necesita ningún palito para pensar. Los filósofos deben comer en paralelo cuando sea posible pero no deben comer con el mismo palito simultáneamente, y tampoco pueden sufrir hambruna.




Ud. debe programar las funciones *pedir* y *devolver*. Programe también *init* para inicializar las variables globales que necesite. Para evitar la hambruna se exige que los filósofos coman por orden de llegada. Por ejemplo si el filósofo 1 está comiendo y 0 llama a pedir, 0 deberá esperar porque el palito 1 está ocupado. Si a continuación 3 llama a pedir también deberá esperar porque llegó después que 0. Finalmente, el filósofo 1 termina de comer e invoca *devolver*. Entonces 0 y 3 deberán comer en paralelo. Observe que la función *devolver* puede despertar 0, 1 o 2 filósofos. Use la cola *Queue* para hacer esperar a los filósofos. Recuerde que la función *peek* le permite obtener el primer elemento de la cola, sin extraerlo.

***Restricciones***: Ud. debe usar el patrón *request* para evitar cambios de contexto innecesarios. Para la sincronización debe usar un mutex y múltiples condiciones.

***Parte ii.-*** Programe las mismas funciones de la parte *i*, pero usando spin-locks como herramienta de sincronización. También deberá usar el patrón *request* y la cola *Queue*.

## Pregunta 2

```
void nPedir(int id);
void nDevolver(int id);
void nth_init( );
```

Programe las funciones *nPedir* y *nDevolver* con los mismos parámetros y funcionalidad de las funciones *pedir* y *devolver* de la pregunta 1, pero como herramientas de sincronización nativas de nThreads, es decir usando operaciones como *setReady*, *suspend*, *schedule*, *START_CRITICAL*, etc. Ud. ***no*** puede implementar la API solicitada en términos de otras herramientas de sincronización pre-existentes en nThreads. Debe evitar cambios de contexto inútiles. Programe además *nth_init* para inicializar variables globales.

*Ayuda*: Use la cola *Queue* para encolar los threads en espera usando el estado WAIT_PALITOS. Use el campo *ptr* del descriptor de thread para guardar la dirección del parámetro *id*.

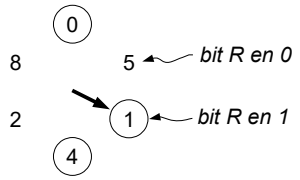
## Pregunta 3

**I.** El proceso A posee código en el intervalo de direcciones virtuales [80 KB, 88 KB[, datos en [100 KB, 112 KB] y pila en [508 KB, 512 KB[. El proceso B posee código en [60 KB, 72 KB[, datos en [96 KB, 112KB[ y pila en [504 KB, 512 KB[. (a) Confeccione las tablas de páginas de ambos procesos, indicando número de página real y atributos V y W. La máquina posee 256 KB de memoria real organizada en páginas de 4 KB. Decida Ud. la asignación de páginas reales. (b) El proceso de B invoca fork; haga la tabla de páginas del hijo justo después de que escribe en la dirección 101 KB considerando que el núcleo implementa la estrategia *copy on write*.

**II.** Suponga que un usuario es capaz de cargar un módulo de Linux. Discuta si esto puede comprometer la seguridad de otros usuarios de la misma máquina.

**III.** Se tiene un archivo de 271 KB en una partición Unix con bloques de 1 KB. Haga un diagrama mostrando inodos, bloques de datos y de indirección. Conteste además: ¿Cuanto espacio en disco se ocupa realmente para almacenar este archivo?

**IV.** Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La figura de la derecha indica el estado inicial de la memoria, mostrando las páginas residentes en memoria, la posición del cursor y el valor del bit R. Dibuje los estados por los que pasa la memoria para la siguiente traza de accesos a páginas virtuales: 8 6 0 1 7 8.




**V.** Ud. debe elegir una herramienta para garantizar la exclusión mutua en una sección crítica de un módulo del núcleo de Linux. Considere una máquina multicore. ¿Bajo qué condiciones sería más eficiente usar un spin-lock y cuando sería más eficiente un semáforo?