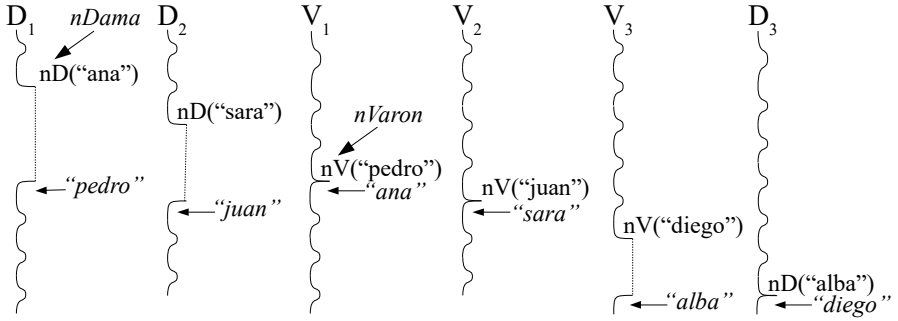
## Pregunta 1

Las tareas de nSystem representan damas y varones que buscan una pareja para bailar en una discoteca invocando *nDama* y *nVaron*. La función *nDama* recibe como parámetro el nombre de la dama. Si hay varones esperando pareja, *nDama* retorna de inmediato el nombre del varón que llegó primero. Es decir que la asignación de parejas se hace por orden de llegada. Si no hay varones en espera, *nDama* espera por una invocación de *nVaron*. Análogamente *nVaron* retorna al instante el nombre de la primera dama que todavía busca su pareja, o espera una invocación de *nDama*. El siguiente es un diagrama de tareas que muestra un ejemplo de ejecución.




El diagrama muestra que la pareja de ana ($D_1$) es pedro ($V_1$) y que por lo tanto *nDama* en $D_1$ retorna "pedro" y *nVaron* en $V_1$ retorna "ana".

Implemente las funciones *nDama* y *nVaron* 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. Declare las variables globales que necesite y especifique los campos que agregará al descriptor de tarea. Los encabezados de las funciones pedidas son:

*char *nDama(char *nombre);*
*char *nVaron(char *nombre);*

## Pregunta 2

Considere una máquina multi-core en la que no existe un núcleo de sistema operativo y por lo tanto no hay un scheduler de procesos. La siguiente es una implementación incompleta del mismo problema de la pregunta 1. En esta pregunta la asignación de parejas se hace en cualquier orden. Programe la función *varon* sin alterar el resto de la implementación. ¡Cuidado! No hay simetría con la función *dama*. No necesitará variables globales adicionales.

```
int qvarones= OPEN;
int qdamas= CLOSED;
int listo= CLOSED;
char *nom_varon;
char **ppareja_varon;

char *varon(char *nom) {
  …
}
```

```
char *dama(char *nom) {
  spinLock(&qdamas);
  char *pareja_dama= nom_varon;
  *ppareja_varon= nom;
  spinUnlock(&listo);
  return pareja_dama;
}
```

*Restricción*: La única forma de busy-waiting admitida es usando un spin-lock.

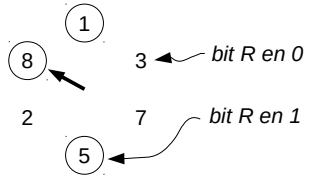
## Pregunta 3

**A.** 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.

**B.** Todos los procesadores tienen una instrucción que desactiva las interrupciones. Explique si sería posible que un programador use esa instrucción para suprimir el mecanismo de *time-slicing* y así impedir que el sistema operativo le quite la CPU a su proceso.

**C.** Compare las dos estrategias de paginamiento en demanda vistas en el curso desde el punto de vista de (i) sobrecosto en tiempo de ejecución cuando la memoria física sobra, (ii) page-faults cuando hay penuria de memoria pero hay un solo proceso en ejecución, (iii) page-faults cuando hay penuria de memoria y hay muchos procesos en ejecución.

**D.** Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La figura 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: 5, 7, 4, 7, 2, 6.