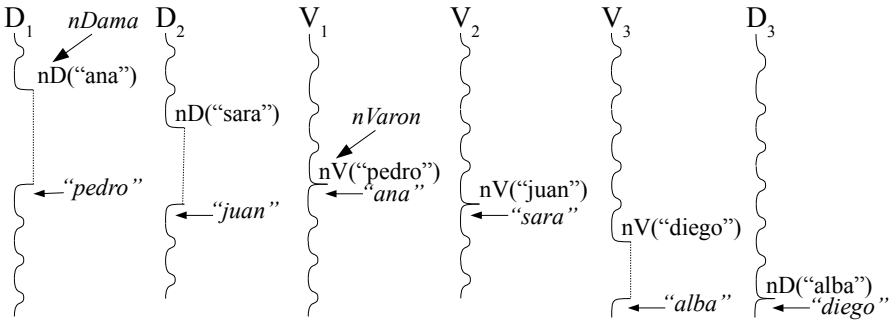
Pregunta 1

I. (4 puntos) 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.

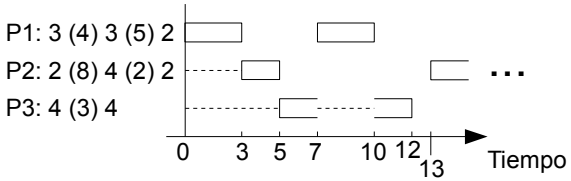


El diagrama muestra que la pareja de ana (D_1) es pedro (V_1) y que por lo tanto *nDama* en D_1 retorna "pedro" y *nVaron* en V_1 retorna "ana".

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);*

II. (2 puntos) El diagrama de arriba a la derecha muestra el scheduling de 3 procesos. En tiempo 0 los 3 procesos están READY (línea punteada). La estrategia de scheduling es en base a prioridades fijas y distintas. Junto a cada proceso se indica la duración de las ráfagas de CPU y entre paréntesis la duración de los estados de espera. Responda: **a.-** Ordene los procesos de mejor a peor prioridad (0,5 puntos) **b.-** Explique si se trata de scheduling *preemptive* o *non-preemptive* (0,5 puntos) **c.-** Complete el diagrama (1 punto).



Pregunta 2

A. (4 puntos) 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 parte I. 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.

B. (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.

C. (1 punto) Un programador plantea eliminar el busy-waiting de los spin-locks de la siguiente manera: cuando se pida un spin-lock que está cerrado, entonces el programador propone suspender el thread que lo pide y retomar otro thread. ¿Cómo le respondería Ud.?