Pregunta 1

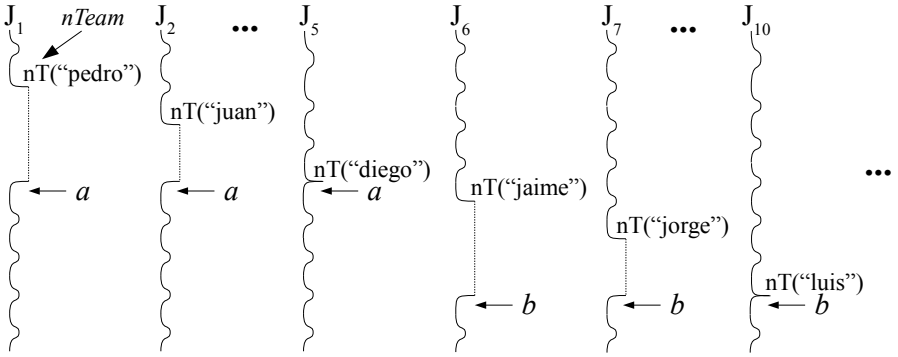
Parte i.- (4 puntos) Se necesita formar múltiples equipos de 5 tareas cada uno. Para ello las tareas de nSystem invocan la función *nTeam* indicando su nombre como argumento. Esta función espera hasta que 5 tareas hayan invocado *nTeam* retornando un arreglo de 5 strings con los nombres del equipo completo. Este es un ejemplo del código de una tarea:

```
int player(char *myname) {
  for (;;) {
    char **team= nTeam(myname);
    playBaby(team);
    drinkBeer();
} }
```

Implemente la función *nTeam* 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. Necesitará usar variables globales. Su solución debe permitir formar un número ilimitado de equipos (no solo 2). El encabezado es:

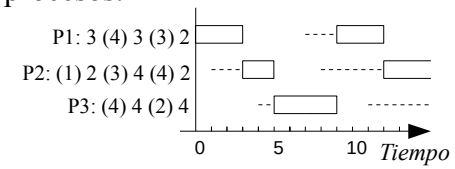
```
char **nTeam(char *name);
```

El siguiente diagrama muestra un ejemplo de uso:



Las primeras 5 tareas (J_1 a J_5) forman el equipo *a* y por lo tanto sus llamadas a *nTeam* retornan el mismo arreglo *a* con los 5 nombres del equipo: "pedro", "juan", …, "diego". Las siguientes 5 tareas (J_6 a J_{10}) forman el equipo *b* y sus llamadas a *nTeam* retornan el arreglo *b*, distinto de *a*, con los nombres "jaime", "jorge", …, "luis".

Parte ii.- (2 puntos) El diagrama parcial muestra las decisiones de scheduling para 3 procesos.



Para cada proceso se indica la duración de la ráfaga y la duración de su estado de espera entre paréntesis. 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 *shortest job first* non preemptive. Considere que el predictor de duración para la próxima ráfaga es la duración de la última ráfaga.

Pregunta 2

A) (4 puntos) Resuelva el mismo problema de la pregunta 1 considerando ahora una máquina multi-core en la que no existe un núcleo de sistema operativo y por lo tanto no hay un scheduler de procesos. Ud. debe programar la función *team* cuyo encabezado es:

```
char **team(char *name);
```

Para programar su solución Ud. dispone de spin-locks y la función *coreId( )*. Necesitará usar variables globales y *malloc*.

Restricción: Dado que no hay un núcleo de sistema operativo, la única forma válida de esperar a que se forme el equipo es utilizando un spin-lock. Otras formas de *busy-waiting* no están permitidas. No hay *fifoqueues* (¡pero sí *malloc*!).

B) (1 punto) Explique por qué se inhiben las interrupciones cuando se usan spin-locks para garantizar la exclusión mutua en un núcleo moderno para multi-core.

C) (1 punto) Explique cuál es la principal razón por la que un núcleo clásico de un sistema operativo no funciona en máquinas multi-core. ¿Cómo haría que un núcleo clásico sí funcione en máquinas multi-core haciendo un mínimo de modificaciones?