CC4302 Sistemas Operativos – Examen – Semestre Primavera 2016 – Prof.: Luis Mateu

Pregunta 1

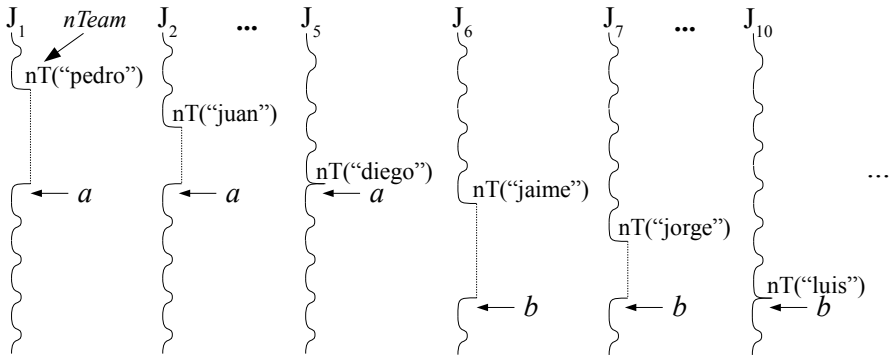
Se necesita formar múltiples equipos de 5 tareas cada uno. Para ello las tareas invocan la función *nTeam* indicando su nombre como argumento. Esta función espera hasta que 5 tareas hayan invocado *nTeam* retornando un arreglo de 5 strings con los nombres del equipo completo. Este es un ejemplo del código de un 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. 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".

Pregunta 2

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);
```

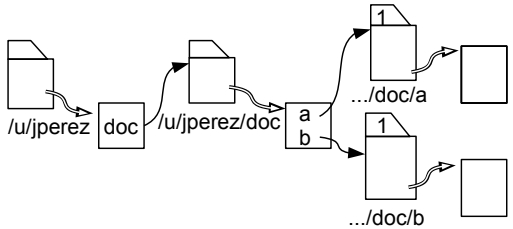
Para programar su solución Ud. dispone de spin-locks y la función *coreId( )*. Necesitará usar variables globales y *malloc*.

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*.

Pregunta 3

a.- Considere que Ud. programa una componente del núcleo de un sistema operativo para una máquina multi-core. Un thread debe esperar hasta que ocurra un cierto evento que gatillará otro thread. Explique cuándo es más eficiente esperar usando *busy-waiting* en vez de retomar otro proceso y cuándo no. Repita su explicación considerando ahora una máquina mono-core.

b.- La figura muesta a la izquierda varios archivos y directorios de la partición /u. El número que aparece en los inodos de los archivos *a* y *b* es el contador de links duros. A la derecha se muestran los comandos que ejecuta el usuario *jperez*:




```
$ cd /u/jperez
$ ln doc/b x
$ mkdir tmp
$ cp x tmp/c
$ ln -s doc/a y
```

Rehaga la figura de la izquierda de acuerdo a los cambios realizados.

c.- 5 procesos se encuentra en estado de espera porque hicieron peticiones para leer sectores del disco en el siguiente orden: 100, 900, 200, 400, 300. El último sector leído fue el 700 y el penúltimo el 800. Indique en qué orden se harán las lecturas de estos 5 procesos cuando la estrategia de scheduling de disco es: (i) *first come first served*, (ii) *shortest seek first*, (iii) método del ascensor (o *look*).

d.- Compare ventajas y desventajas de usar un disco tradicional para paginamiento versus un *solid state drive* (SSD). En su comparación considere número de page faults atendidos por segundo, velocidad (en MB/seg.) para cargar en memoria un proceso que fue llevado a disco (estado *swap*) y tiempo de vida del dispositivo de paginamiento.

e.- ¿Cuál es el tamaño máximo de un archivo que usa un solo bloque de indirección simple en una partición con bloques de 1 KB? ¿Y si la partición posee bloques de 4 KB?