CC4302 Sistemas Operativos – Examen – Semestre Otoño 2015 – Prof.: Luis Mateu

Pregunta 1

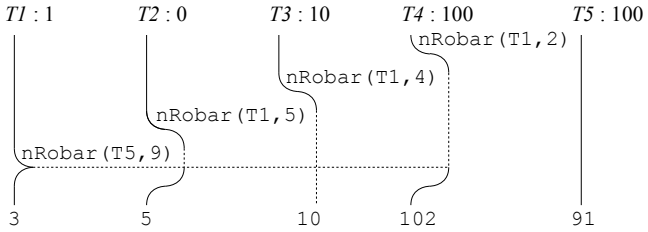
Considere que una tarea en nSystem almacena inicialmente 100 euros. Una tarea puede robar dinero desde otra tarea con la siguiente función:

void nRobar(nTask desde, int cantidad);

En tal caso se le resta *cantidad* euros a la tarea *desde* y se le suman a la tarea que invocó *nRobar*. El parámetro *cantidad* es siempre mayor que cero. Un primer invariante es que una tarea no puede almacenar una cantidad negativa de euros. Si la tarea *desde* no posee suficiente dinero para robarle *cantidad* entonces *nRobar* debe esperar hasta que la tarea *desde* sí posea al menos la cantidad requerida. El segundo invariante es que en un instante dado la tarea *T* no puede estar esperando robarle *c* euros a la tarea *U* si *U* tiene al menos *c* euros.

Observe que cuando la tarea *T* lograr robar dinero, varias otras tareas pueden estar esperando poder robarle dinero a *T*. No está especificado en qué orden deben robarle el dinero a *T*.

El siguiente diagrama de tareas es un ejemplo de uso de *nRobar*.



En el inicio del diagrama las 5 tareas T1 a T5 poseen 1, 0, 10, 100 y 100 euros respectivamente. Al final poseen 3, 5, 10, 102 y 91 euros. Las tareas T2, T3 y T4 le roban a T1 que no tiene suficiente dinero y por lo tanto deben esperar. Cuando T1 roba 9 euros, T2 y T4 logran su robo pero T3 continúa esperando. Note que es válido haber dejado esperando a T2 o T4 en vez de T3.

Programe la función *nRobar* usando las funciones de bajo nivel de nSystem (*START_CRITICAL*, *Resume*, *PutTask*, etc.). Indique qué nuevos campos necesita agregar al descriptor de tarea y su valor inicial. Ud. *no puede* usar otros mecanismos de sincronización ya disponibles en nSystem, como semáforos, monitores, mensajes, etc.

Pregunta 2

Considere una máquina con 8 cores físicos que comparten la memoria, sin un núcleo de sistema operativo y por lo tanto no hay scheduler de procesos. Los cores se enumeran con valores que van de 0 a 7. De manera similar a la pregunta 1 a cada core se le asignan inicialmente 100 euros. Un core puede robar *cantidad* euros del core número *desde* invocando la función:

void robar(int desde, int cantidad);

Programe la función *robar* y declare las variables globales que necesite señalando su valor inicial. Ud. dispone de la función *coreId( )* que entrega el número del core que la invoca (entre 0 y 7). Use *spin-locks* para sincronizar los distintos cores.

Restricciones: No hay *nMalloc* ni tampoco *fifoqueues*. Dado que no hay una cola de procesos *ready*, para esperar no le queda otra que hacerlo mediante un *spin-lock*, pero no puede recurrir a otra forma de *busy-waiting*.

Ayuda: Use una matriz *m* de 8 por 8 punteros a spin-locks. Si *m[i][j]* no es nulo quiere decir que *m[i][j]* es la dirección de un *spin-lock* en el que el core *i* espera para robarle al core *j*.

Pregunta 3

A.- ¿Cuál es la ventaja número 1 de un núcleo clásico por sobre un núcleo moderno? ¿Y cuál es la desventaja número 1?

B.- ¿En qué tipo de sistema el shell de comandos es más eficiente? ¿En un sistema que usa segmentación o en uno basado en paginamiento? Explique por qué.

C.- Explique en palabras como programaría la función *copy_from_user* usada para implementar drivers de dispositivos, e indique cuál es el cuidado más importante que Ud. debe considerar desde el punto de vista de la seguridad del núcleo.

D.- Confeccione una tabla con 3 columnas para las estrategias de *scheduling* de disco (1) *first come first served*, (2) *shortest seek first*, y (3) *método del ascensor*. Incluya 3 filas considerando la cantidad de procesos intensivos en entrada/salida en ejecución: (i) un solo proceso, (ii) 2 procesos, y (iii) muchos procesos. En cada celda señale para ese número de procesos si la estrategia considerada es la mejor en tiempo promedio de acceso al disco, la peor o la del medio, o si se comporta igual a alguna de las otras, etc. Además en cada celda indique si para ese número de procesos se puede producir hambruna o no.