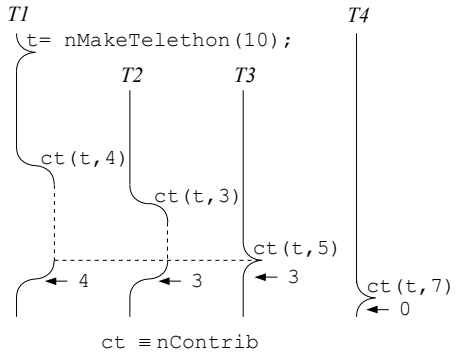
CC4302 Sistemas Operativos – Control 2 – Semestre Otoño 2016 – Prof.: Luis Mateu

Pregunta 1

Parte I. (4,5 puntos) Se desea agregar un sistema de Teletón de manera nativa a nSystem. La API para este sistema es la siguiente:

```
nTelethon *nMakeTelethon(double goal);
double nContrib(nTelethon *t, double x);
```

La función *nMakeTelethon* crea y entrega una nueva Teletón para juntar exactamente la meta de *goal* dólares. La función *nContrib* es invocada por múltiples tareas de nSystem para aportar *x* dólares. Esta función debe esperar hasta que la suma de los aportes alcance la meta, retornando el monto efectivo aportado. Por ejemplo si faltan 3 dólares para alcanzar la meta y una tarea aporta 5, entonces *nContrib* retorna 3 en esa tarea. Si una tarea invoca *nContrib* cuando la meta ya fue alcanzada, entonces se retorna 0 de inmediato. La siguiente figura muestra un ejemplo de ejecución con múltiples tareas.



Defina el tipo nTelethon y programe las funciones *nMakeTelethon* y *nContrib* usando las funciones de bajo nivel de nSystem (*START_CRITICAL*, *Resume*, *PutTask*, etc.). Sea eficiente: evite cambios de contexto innecesarios.

Parte II. (1,5 puntos) Suponga que Ud. implementa un núcleo moderno de Unix para una máquina multicore. Las contrapartes de los procesos en el núcleo (los *peers*) necesitan (a) consultar y modificar un diccionario implementado mediante una tabla de hashing, y (b) leer o escribir en un disco, considerando que cada lectura o escritura toma unos 10 milisegundos. Discuta qué herramienta de sincronización usaría para garantizar la exclusión mutua para los casos (a) y (b) eligiendo entre un *spin-lock* o un semáforo como el que se implementó en clases de cátedra para un núcleo moderno de una máquina multicore. Justifique sus 2 decisiones en términos de la eficiencia en tiempo de CPU y el mejor aprovechamiento del recurso CPU.

Pregunta 2

A) (4 puntos) Programe el mismo sistema de Teletón de la pregunta 1 para una máquina con 8 cores físicos, sin un núcleo de sistema operativo y por lo tanto no hay scheduler de procesos. Los 8 cores ejecutan código en paralelo y comparten la memoria. Los *spin-locks* son la única herramienta disponible para sincronizar los distintos cores. Ud. dispone de la función *coreId( )* que entrega el número del core que la invoca (entre 0 y 7). La API de este sistema es:

```
void iniTelethon(Telethon *t, double goal);
double contrib(Telethon *t, double x);
```

Observe que no hay threads: los que invocan estas funciones son los distintos cores. Dado que no hay una cola de procesos *ready*, para esperar no le queda otra que hacerlo mediante un *spin-lock*. No puede recurrir a otra forma de *busy-waiting*. No hay *malloc* ni existen colecciones como las colas FIFO.

B) (1 punto) Una ráfaga de un proceso se ejecuta en 3 tajadas (*slices*). ¿De qué tipo de scheduling se trata? ¿Preemptive o non-preemptive? Explique.

C) (1 punto) ¿Cuantos spin-locks se necesitan en un núcleo clásico de Unix para una máquina multi-core? ¿Y cuantos spin-locks se necesitan en un núcleo moderno para una máquina mono-core?