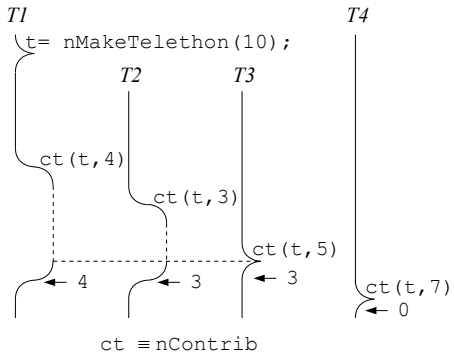
CC4302 Sistemas Operativos – Examen – Semestre Primavera 2014 – Prof.: Luis Mateu

Pregunta 1

I. (3 puntos) Se desea agregar un sistema de Teletón de manera nativa a nSystem. La API para este sistema es la siguiente:

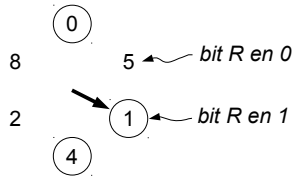
```
nTelethon *nMakeTelethon(double goal);
double nContrib(nTelethon *t, double x);
```

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 figura de la derecha 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.

II. (1 punto) Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La figura de la derecha indica el estado inicial de la memoria, mostrando las páginas residentes en memoria, la posición del cursor y el valor del bit R:



Dibuje los estados por los que pasa la memoria para la siguiente traza de accesos a páginas virtuales:

2 3 5 3 7 0

III. (1 punto) ¿Tiene sentido usar *spin-locks* en una máquina monocore? Explique. Responda entonces cómo implementaría la exclusión mutua en un núcleo moderno de Unix para una máquina monocore.

IV. (1 punto) Se acaba de leer el bloque 600 y previamente se había leído el bloque 300 de un disco tradicional. Los procesos P1, P2, P3, P4, P5 y P6 se encuentran en espera con peticiones para leer los bloques 100, 1200, 200, 1100, 700 y 400 respectivamente. En qué orden se harán las lecturas cuando la estrategia es a) *shortest seek first*, b) método del ascensor o *look*, o c) *circular-look*.

Pregunta 2

A) (3 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) Haga una tabla con 3 columnas para las estrategias de scheduling *first come first served*, *shortest job first* y *round robin*, y 4 filas para las propiedades (i) tiempo de despacho, (ii) tiempo de respuesta, (iii) simplicidad de la implementación, y (iv) número de cambios de contexto implícitos. En las celdas coloque palabras como malo, bueno, regular, pésimo, simple, compleja, 0, pocos, muchos, etc. para indicar cómo le va a una estrategia comparada con las demás.

C) (1 punto) Un sistema operativo tiene un disco dedicado exclusivamente a paginamiento. El disco es capaz de leer a 100 MB/seg. y tiene un tiempo de acceso de 5 milisegundos. Estime (i) cuantos *page-faults* puede atender por segundo, y (ii) cuanto tardaría en llevar a memoria un proceso que se encuentra en estado *swap*, suponiendo que su *working set* es de 300 MB. Complemente sus 2 estimaciones explicando por qué está usando un determinado parámetro del disco y no el otro.

D) (1 punto) Explique para qué sirve el número *major* en un *device*. ¿Por qué es importante considerarlo al momento de programar un módulo?