CC4302 Sistemas Operativos

Control 2 – Semestre Primavera 2012

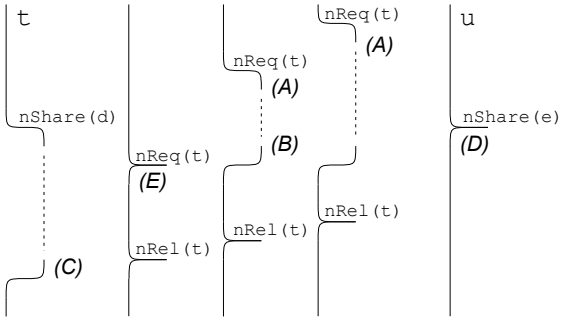
Prof.: Luis Mateu

Pregunta 1

Se le pide a Ud. agregar una nueva primitiva de sincronización para compartir datos en nSystem. La API de este mecanismo es la siguiente:

Procedimiento	*Significado*
`void nShare(char* data);`	Ofrece datos para compartir
`char *nRequest(nTask t);`	Solicita datos provenientes de la tarea t
`void nRelease(nTask t);`	Notifica que los datos ya no se usarán

La figura muestra que nRequest debe esperar que t invoque nShare (ver *A*). Cuando t invoca nShare, se desbloquean todas las tareas que esperaban un nShare(data) de t (ver *B*), retornando todas ellas el puntero data. nShare se bloquea hasta que todas las tareas con un nRequest pendiente de t notifiquen mediante nRelease que data de t no se usará más (ver *C*). Si t invoca nShare y no encuentra tareas solicitando datos de t, nShare retorna de inmediato (ver *D*). Si se invoca nRequest(t) mientras nShare está activo en t, nRequest retorna de inmediato los datos que actualmente comparte t (ver *E*).



Implemente esta API 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. Ud. sí puede introducir modificaciones en el descriptor de tarea (nTask), agregar estados, etc.

Pregunta 2

Parte a.- (1 punto)

Un programador plantea eliminar el busy-waiting de los spin-locks de la siguiente manera: cuando se pida el spin-lock y éste está cerrado, entonces suspender el thread que lo pide y retomar otro thread. ¿Cómo le respondería Ud.?

Parte b.- (1 punto)

Considere una máquina multi-core con un núcleo clásico de Unix. (i) ¿Cómo se garantiza la exclusión mutua en el núcleo? (ii) ¿Se puede usar una variable global current_process? (iii) ¿Cuál es la principal limitante desde el punto de vista del desempeño?

Parte c.- (4 puntos)

Se tiene una máquina *multi-core* en la que no existe un núcleo de sistema operativo y por lo tanto solo se pueden usar los *spin-locks* para sincronizar los cores. Se le pide implementar un semáforo que garantice un *orden de atención FIFO*. Para ello Ud. debe definir el tipo Sem y programar las siguientes funciones:

`void iniSem(Sem *sem, int ini);`	Inicializa un semáforo con `ini` tickets disponibles inicialmente
`void waitSem(Sem *sem);`	Solicita un ticket
`void signalSem(Sem *sem);`	Deposita un ticket

Hint: Use la metáfora de la Isapre.

Dado que no hay núcleo de sistema operativo:

- no existe un scheduler de procesos: no puede llamar a `Resume`
- no existe manejo de colas: no puede usar `PutTask`, `GetTask`
- no hay herramientas de sincronización avanzadas como monitores, semáforos o mensajes
- no existe manejo de memoria como malloc/free, etc.

Ud. debe resolver el problema usando únicamente los spin-locks, i.e. las funciones `spinLock(int *sl)` y `spinUnlock(int *sl)`. Por lo tanto cuando waitSem debe esperar por un ticket, no tiene otra alternativa que hacer *busy-waiting*.