CC41B: Sistemas Operativos
Tarea 2 - Semestre Primavera'2006
Plazo de entrega: Viernes 27 de Octubre
Prof.: Luis Mateu

Scheduling con prioridades en nSystem

nSystem2007 beta viene con una estrategia de scheduling basada en prioridades. Las tareas pueden definir su propia prioridad invocando por ejemplo `nSetPriority(5)`. La prioridad es un entero entre 0 y 3, en donde 0 es la mejor prioridad y 3 la peor. El invariante del scheduler consiste en que, en cualquier instante, el procesador no puede estar asignado a una tarea cuya prioridad es peor que la prioridad de alguna de las tareas que se encuentran en la cola de tareas listas para ejecutarse. Entre tareas de la misma prioridad, se sigue aplicando la estrategia *preemptive last come first served* que se vio en clases de cátedra. Cuando se crea una nueva tarea, ésta toma inicialmente la prioridad de la tarea que invocó `nEmitTask`.

La anomalía de la inversión de prioridades

Cuando se emplean semáforos para garantizar la exclusión mutua entre tareas, ocurre una anomalía que recibe el nombre de *inversión de prioridades*. Para entender esta anomalía considere la siguiente situación: Una tarea C de baja prioridad ha ganado el acceso a la zona de exclusión mutua. Pero C no avanza porque existe una tarea A de alta prioridad y otra tarea B de prioridad media que acaparan la CPU. Entonces A solicita el semáforo de exclusión mutua, pero no es otorgado porque está en manos de C. A tendrá que esperar a que B termine, o ingrese a un estado de espera, para que C pueda recibir el procesador y así liberar el semáforo. La anomalía consiste en que A debe esperar a que se ejecute la tarea B, de menor prioridad, para poder continuar.

Sincronización por medio de mutex

En esta tarea Ud. deberá mejorar nSystem2007 beta implementando una nueva herramienta de sincronización denominada *mutex*, la cual evitará la anomalía de la inversión de prioridades. Esta herramienta es equivalente a un semáforo binario y se emplea únicamente cuando dos o más tareas necesitan garantizar la exclusión mutua en alguna zona de código (de ahí su nombre). Se usa de la siguiente manera:

`nMutex mutex= nMakeMutex();`	

`Tarea 1`
`nRequest(mutex);`
`  ...` *zona de exclusión mutua* `...`
`nRelease(mutex);`
	`Tarea 2`
	`nRequest(mutex);`
	`  ...` *zona de exclusión mutua* `...`
	`nRelease(mutex);`

Un mutex posee las siguientes características:

- Es binario: puede estar libre u ocupado. Cuando está libre, cualquier tarea que lo solicite con `nRequest` lo obtendrá de inmediato dejándolo ocupado hasta que la misma tarea lo libere con `nRelease`.
- Cuando el mutex está ocupado, las tareas que invoquen `nRequest` quedarán en espera hasta que el mutex se libere. *Las tareas adquieren el mutex de acuerdo a su prioridad*. A igualdad de prioridades se respeta el orden de llegada de las solicitudes.
- Una tarea puede estar ocupando varios mutex en un instante dado. *Los mutex se liberan en el orden inverso a cómo se solicitaron*. No hacerlo constituye un error por parte del programador usuario de los mutex. Esta restricción simplificará su implementación.

Su implementación de los mutex debe evitar la anomalía de la inversión de prioridades. Para ello, cuando una tarea C de baja prioridad posee un mutex y éste es solicitado por una tarea A de mayor prioridad, C adquiere temporariamente la prioridad de A hasta liberar el mutex. Esto se logra manteniendo una *prioridad formal* que corresponde a la que se especifica usando `nSetPriority`, y una *prioridad efectiva* que es la que finalmente empleará el scheduler. La prioridad efectiva de una tarea T se calcula como la mejor prioridad entre (i) la prioridad formal de la tarea T, y (ii) las prioridades efectivas de las tareas que esperan adquirir uno de los mutex que T está ocupando. La anomalía se evita porque aún cuando una tarea B de prioridad media esté lista para ejecutarse, el scheduler pasará el procesador a una tarea C de menor prioridad, cuando ésta posee un mutex que fue solicitada por una tarea A de mayor prioridad.

Observe que de todas formas la tarea A tendrá que esperar a que C libere el mutex, teniendo A mejor prioridad formal que C. Pero esto es inevitable porque una vez que una tarea obtiene un mutex, sólo ella lo puede liberar.

Indicaciones

Considere cuidadosamente la siguientes observaciones para facilitar el desarrollo de su tarea:

- Estudie los siguientes programas en el directorio `nSystem2007/ex-pri/`: `testpri.c` sirve para verificar que las prioridades están fucionando correctamente. La anomalía se muestra en acción en `pri-inversion.c`. La versión con mutex está en `pri-invmutex.c`.
- Haga modificaciones menores a `nSystem2007/include/nSystem.h`: agregue la declaración del tipo nMutex

y los procedimientos `nRequest`, `nRelease`, `nMakeMutex` y `nDestroyMutex`.

- Haga modificaciones menores a `nSystem2007/src/nSysimp.h`: Agregue un campo en el descriptor de procesos para almacenar la prioridad formal de una tarea y otro campo para almacenar una lista enlazada de los mutex que ocupa una tarea. Esta lista le servirá para calcular la prioridad efectiva. Recuerde que los mutex se liberan en el orden inverso a cómo se solicitan. Agregue el estado WAIT_MUTEX.
- Haga modificaciones menores a `nSystem2007/src/nProcess.c`: `nSetPriority` debe cambiar la prioridad formal de la tarea en ejecución y recalcular su prioridad efectiva. Tenga cuidado porque si se baja la prioridad, la tarea quizás deba ceder el procesador. Además vea en donde se inicializa el campo `pri` del descriptor de proceso para saber en donde debe iniciliazar la prioridad formal.
- Agregue el archivo `nSystem2007/src/nMutex.c`: Implemente acá los nuevos procedimientos para el manejo de los mutex. Agregue otros procedimientos internos si lo necesita. Para que el comando `make` considere este archivo, agregue `nMutex.o` a la variable NSYSTEM definida en el archivo `nSystem2007/src/Makefile`.
- Recalcule la prioridad efectiva de una tarea T cuando (i) otra tarea solicite un mutex que posee actualmente T, (ii) cuando T libere un mutex, y (iii) cuando se invoque `nSetPriority`. Le será de utilidad el procedimiento `BestWaitingPriority(Q)`, implementado en `nQueue.c`, que calcula la mejor prioridad de las tareas que esperan en la cola Q.
- Verifique que su tarea funciona con `nSystem2007/ex-pri/testmutex.c`.

Recursos

Baje nSystem2007 beta de `http://www.dcc.uchile.cl/~lmateu/CC41B`.

Plazo de entrega

La tarea se entrega en U-cursos. Para ello entregue un archivo tar comprimido con gzip que incluya todos los archivos de nSystem que Ud. modificó. No incluya archivos binarios. El plazo de entrega vence el Viernes 27 de Octubre. Se descontará medio punto por día hábil de atraso. Se recomienta vivamente resolver esta tarea antes del control 2.