CC41B : Sistemas Operativos
Tarea 2 – Semestre 2002/2
Plazo de entrega : Viernes 25 de Octubre
Prof.: Luis Mateu.

La principal novedad de nSystem2002 es que 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 99, en donde 0 es la mejor prioridad y 99 la peor. El invariante que verifica el scheduler es 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 se crea con 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 nSystem2002 implementando una nueva herramienta de sincronización denominada *mutex*, la cual evita la anomalía de la inversión de prioridades. Esta herramienta se emplea únicamente cuando dos o más tareas necesitan garantizar la exclusión mutua en alguna zona de código (de ahí su nombre) de la siguiente manera :

```
                  nMutex mutex= nMakeMutex();

    Tarea 1             |     Tarea 2
    -------             |     -------
  nRequest(mutex);      |   nRequest(mutex);
    ... zona de         |     ... zona de
    exclusion mutua ... |     exclusion mutua ...
  nRelease(mutex);      |   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.

Para evitar la anomalía de la inversión de prioridades Ud. debe manejar 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. Con esto se evita la anomalía porque cuando la tarea A, de alta prioridad, solicita el mutex que está siendo ocupado por la tarea C, de baja prioridad, esta última adquiere la prioridad de A hasta que libere el mutex. De esta forma, C tendrá temporalmente mejor prioridad que la tarea B.

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 `nSystem2002/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 `testmutex.c`.
- Haga modificaciones menores a `nSystem2002/include/nSystem.h`: agregue la declaración del tipo `nMutex` y los procedimientos `nRequest`, `nRelease`, `nMakeMutex` y `nDestroyMutex`.
- Haga modificaciones menores a `nSystem2002/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 `nSystem2002/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 `nSystem2002/src/nMutex.c`: Implemente acá los nuevos procedimientos para el manejo de 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 `nSystem2002/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.
- Se entregará un programa de prueba para que verifique que su tarea funciona. Su tarea se probará con este programa más algún otro ejemplo. Pruebe su tarea mientras tanto con `testmutex.c`.

Entrega de la tarea

Esta tarea es individual y se entrega mediante el cursomático de CC41B. Entregue *todos y solo* los archivos que modificó en los fuentes originales de nSystem. *No entregue* el programa de prueba. El cursomático está en:

`http://www.dcc.uchile.cl/~cc41b/cursomatico`

El plazo de entrega vence el día Viernes 25 de Octubre. Se descontarán 5 décimas por cada día de atraso. La tarea se debe entregar funcionando. Las tareas que no funcionen con el programa de prueba suministrado se calificarán con nota 1.0.

Observación: En este curso no habrá notas I. Si Ud. tiene nota inferior a 4.0 en tareas, queda reprobado.