Sistemas Operativos II - LCC
Práctica 2 - 2014
Sincronización de hilos en Nachos
Entrega: 6 de Mayo.

NOTA: Debe utilizar el SVN de la materia creando un subdirectorio por alumno/grupo en:
`https://dcc.fceia.unr.edu.ar/svn-no-anon/lcc/R-412/Alumnos/2014/`

1. Concurrencia en NACHOS

Nota: Al resolver los ejercicios recordar que el código bien sincronizado debe funcionar sin importar qué orden elige el scheduler para ejecutar los threads listos. Más explícitamente: deberíamos poder poner una llamada a Thread::Yield en cualquier parte del código donde las interrupciones están habilitadas sin afectar la corrección del código. Se recomienda usar la opción -rs de nachos.

1. Implementar locks y variables de condición, usando semáforos como base, esto quiere decir que tanto los locks como variables de condición **NO** deben apagar las interrupciones ni dormir hilos, sino hacerlo con semáforos como base. En synch.h está la interfase pública. Se deben definir los datos privados e implementar la interfase. Debe implementar también la función `Lock::isHeldByCurrentThread` y utilizarla para comprobar (mediante ASSERT) que el hilo que realiza un `Acquire` **no posee** el lock y que el hilo que hace `Release` **sí lo posee**.

2. Implementar paso de mensajes entre hilos a través de la clase "Puerto", que permite que los emisores se sincronicen con los receptores. Puerto::Send(int mensaje) espera atómicamente hasta que se llama a Puerto::Receive(int *mensaje), y luego copia el mensaje int en el buffer de Receive. Una vez hecha la copia, ambos pueden retornar. La llamada Receive también es bloqueante (espera a que se ejecute un Send, si no había ningún emisor esperando). **Nota:** La solución debe funcionar incluso si hay múltiples emisores y receptores para el mismo puerto.

3. Implementar un método Thread::Join que bloquea al llamante hasta que el hilo en cuestión termine.

```
Thread *t = new Thread("Hijo");
t->Fork(func,0);
t->Join(); // Aquí el hilo que ejecuta se bloquea
   // hasta que t termine.
```

Agregar un argumento al constructor de threads que indica si se llamará a un Join sobre este thread. La solución deberá borrar adecuadamente el thread control block, tanto si se hará Join como si no y aunque el thread (hijo) termine antes de la llamada a Join.

4. El scheduler de nachos implementa una política de round-robin. Implementar multicolas con prioridad. Establecer prioridades fijas para cada thread (positivas, 0 menor prioridad). El scheduler debe elegir siempre el thread listo con mayor prioridad. Modificar la implementación para solucionar o evitar en el caso de los locks y variables de condición el problema de inversión de prioridades. Explique (en un archivo de texto) porqué no puede hacerse lo mismo con los semáforos.