Sistemas Operativos - Práctica Nº3 - 2007

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.

   En synch.h está la interfase pública. Se deben definir los datos privados e implementar la interfase.

2. Implementar send y receive, usando variables de condición.

   Los mensajes se envían a ``puertos'', que permiten que los emisores se sincronicen con los receptores. Send(puerto, int mensaje) espera atómicamente hasta que se llama a Receive(puerto, int *mensaje) en el mismo puerto, 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).

   La solución debe funcionar incluso si hay múltiples emisores y receptores para el mismo puerto.

3. Implementar Thread::Join. 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) finalice antes de la llamada a Join.

4. Implementar multicolas con prioridad.

   1. Establecer prioridades fijas para cada thread (positivas, 0 $\Rightarrow$ menor prioridad). El scheduler debe elegir siempre el thread listo con mayor prioridad.
   2. Modificar la implementación para solucionar o evitar en el caso de los mutexs y variables de condición el problema de inversión de prioridades. ¿Porqué no puede hacerse lo mismo con los semáforos?