Práctica 2: Sincronización de hilos

2024 – Sistemas Operativos 2

Licenciatura en Ciencias de la Computación

29 de marzo de 2024

1. Introducción

Al resolver los ejercicios, recuerde que el código bien sincronizado debe funcionar sin importar qué orden elija el planificador para ejecutar los hilos listos. Más explícitamente: se debería poder poner una llamada a `Thread::Yield` en cualquier parte del código donde las interrupciones estén habilitadas sin afectar la corrección del código. Se recomienda usar la opción `-rs` de Nachos.

2. Ejercicios

1. Implemente "locks" (candados) y variables de condición. Use semáforos como base; esto implica que tanto los "locks" como las variables de condición **no** deben apagar las interrupciones ni dormir hilos, sino proveer su funcionalidad por medio de semáforos.

   En `threads/lock.hh` y `threads/condition.hh` están las interfaces públicas- Se deben definir los datos privados e implementar la interfaz. Debe implementar también la función `Lock::IsHeldByCurrentThread` y utilizarla para comprobar (mediante `ASSERT`) que el hilo que realice un `Acquire` no posea el "lock" y que el hilo que haga `Release` sí lo posea. Agregue además otras llamadas a `ASSERT` y `DEBUG` que considere convenientes. Utilice la bandera de debug `s`.

2. Nachos provee una interfaz para el test del problema del productor y consumidor. Agregue una implementación para el mismo donde utilice las estructuras implementadas en el ejercicio anterior. Suponga que los items producidos son enteros y se almacenan en un buffer acotado de tamaño 3. Sólo hay un productor y un consumidor y se produce un total de 1000 items (el primer item es 1, el segundo 2, etc...). El productor debe incluir las siguientes líneas:

   ```
   printf("Productor esperando (buffer lleno)\n");
   printf("Productor produce: %d en %d\n", i, pos);
   ```

   y el consumidor debe incluir estas:

   ```
   printf("Consumidor esperando (buffer vacio)\n");
   printf("Consumidor consume: %d en %d\n", i, pos);
   ```

   Opcional: implemente otros casos de test para "locks" y variables de condición.

3. Implemente paso de mensajes entre hilos a través de canales, que permitan que los emisores se sincronicen con los receptores. Haga una nueva clase `Channel` con los siguientes métodos:

   ```
   void Channel::Send(int message);
   void Channel::Receive(int *message);
   ```

   `Send` espera atómicamente hasta que se llame a `Receive` y luego copia el mensaje en el búfer de `Receive`. Una vez hecha la copia, ambos pueden retornar. La llamada a `Receive` también es bloqueante: en caso de que no haya ningún emisor esperando, espera a que llegue uno (es decir, que se ejecute un `Send`).

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

   Opcional: implemente casos de test para canales.

4. Implemente un método `Thread::Join` que bloquee al llamante hasta que el hilo en cuestión termine.

   ```
   Thread *t = new Thread("Hijo");
   t->Fork(func, nullptr);
   t->Join();  // El hilo en ejecución se bloquea hasta que 't' termine.
   ```

   Agregue un argumento al constructor de `Thread` que indique si se llamará a `Join` sobre este hilo.

   La solución debe borrar adecuadamente el bloque de control del hilo ("thread control block" o TCB), tanto si se hará `Join` como si no y aunque el hilo hijo termine antes de la llamada a `Join`.

   Opcional: implemente casos de test para `Join`.

5. *a*) El planificador de Nachos implementa una política de "round robin". Implemente multicolas con prioridad.

   Establezca prioridades fijas para cada hilo (enteros entre 0 como la mínima prioridad y 9). El planificador debe elegir siempre el hilo listo con mayor prioridad. Agregue un nuevo constructor a Thread que tome como argumento adicional la prioridad del nuevo hilo y modifique el constructor ya provisto para que la prioridad por defecto sea 4.

   b) Modifique la implementación para solucionar o evitar en el caso de los "locks" y variables de condición el problema de inversión de prioridades.

   Explique (en su implementación para los locks) por qué no puede hacerse lo mismo con los semáforos.