Telecomunicaciones y Sistemas Distribuidos/Redes y Telecomunicaciones

Proyecto de aprobación

2013

Resumen

Este proyecto pretende que el estudiante aplique algunos de los algoritmos distribuidos analizados en el curso. Además requiere que se utilicen los conceptos sobre comunicaciones, diseño e implementación de protocolos de comunicaciones.

1. El problema

Se require desarrollar un sistema distribuido en el que se simule un conjunto de procesos ejecutándose en forma concurrente o paralela en un sistema de computación con un mecanismo de pasaje de mensajes asíncrono.

Cada proceso contiene un conjunto de recursos locales que pueden ser compartidos con los demás procesos y cada proceso requerirá de un subconjunto de recursos de manera exclusiva temporariamente, luego usarlos y liberarlos.

Cada conjunto de recursos a los cuales se accede está abstraido en una región crítica. Dos regiones críticas diferentes acceden a conjuntos de recursos disjuntos.

El sistema distribuido consta de n procesos y k regiones críticas.

Cada proceso tiene el siguiente comportamiento:

```
...
while (! finish() ) {
  // ... do something
```

```
  critical_region r = get_critical_region();
  enter(r);
  ...
  release(r);
}
exit();
```

donde la función `get_critical_region()` genera[1] la región crítica a acceder en el ciclo corriente.

2. Requisitos

Implementar el sistema distribuido descripto sin usar una coordinación central. Podrá usar las herramientas de desarrollo de su elección. Concretamente deberá:

1. Implementar un conjunto de primitivas de comunicación de alto nivel para implementar los algoritmos requeridos. En particular, se recomienda la siguiente API:

   ```
   processes(); // number of processes
   my_pid();
   send_to(pid, msg);
   msg=recv_from(pid);
   ```

   donde `pid` en `sent_to()` y `recv_from()` es el identificador de proceso destino o `ANY` (para soportar broadcasting). Las demás funciones tendrán la semántica obvia. Puede simular la ejecución del código dentro de una región crítica con una simple demora (al igual con `do something`).

2. Definir un protocolo (algoritmo) distribuido para que cada proceso acceda a su conjunto de recursos en forma exclusiva cada vez que los solicite.

3. Cada proceso deberá poder ejecutarse por la línea de comandos, dándole un identificador de proceso.

[1] Por ejemplo, de manera aleatoria.

4. Los procesos pueden estar concurrentemente en diferentes regiones críticas simultáneamente.

5. Implementar un mecanismo de terminación automática (cuando todos los nodos estén ociosos y no haya mensajes en tránsito).

3. Evaluación

Cada grupo deberá exponer su implementación y cada uno de sus integrantes deberán poder responder satisfactoriamente a las preguntas realizadas por el equipo docente y los demás alumnos.

Los elementos de evaluación serán:

- Correctitud de la implementación.
- Facilidad de uso.
- Generalidad.
- Algoritmos seleccionados para las dos soluciones pedidas.
- Análisis de rendimiento (en términos de números de mensajes requeridos) de cada solución.

4. Algunas recomendaciones

Para poder simular cada nodo, pueden utilizarse dos enfoques:

1. Uso de dos threads (o procesos) en cada nodo: Un thread simularía la actividad del proceso (*main*) y otro thread (*listener*) sería en encargado de recibir los mensajes de los demás procesos y notificar al *main*.

2. Es posible realizar la simulación en un único thread usando I/O asíncrona (non-blocking I/O):

   En los sistemas tipo UNIX, es posible hacer I/O asíncrona usando `fcntl(s,F_SETFL,O_NONBLOCK)`.

Luego de hacer esto, la llamada *n = read(s,buf,n);* no se bloqueará. Si efectivamente hay algo en el canal, `n` será mayor que cero.

Cada vez que ocurra una actividad de I/O en `s`, el sistema generará una señal SIGIO. Si la aplicación instala un *signal handler* para `SIGIO`, ésta función se disparará automáticamente ante el arribo de un mensaje.

En el sitio web del curso encontrará un ejemplo con programas cliente y servidor utilizando I/O asincrónica con UDP.

En ambos enfoques conviene utilizar canales sin conexión (ej: UDP).

Además deberá tener en cuenta que ambos enfoques conllevan a ejecución asincrónica o concurrente de código, por lo cual deberá tener en cuenta las posibles condiciones de carrera que puedan surgir sobre los datos o estado del proceso.