Sistemas Operativos
Trabajo Práctico III

Federico Huel (329/07)
Juán Manuel Lomonaco (603/07)
Rodrigo Campos (561/06)

November 15, 2011

Detalles de implementación

Solo fue necesario cambiar el código del servidor provisto por la cátedra, ya que ésta era la única parte "sin implementar" (o con una implementación a modo de ejemplo que no era correcta). Y el envío de mensajes al que se refiere el paper "An optimal algorithm for mutual exclusion in computer networks", en el contexto de este trabajo, se refiere a envío de mensajes entre servidores.

Implementación del algoritmo usando message-passing

El paper plantea tres procesos, cada uno con distintas responsabilidades. Y si bien la separación en procesos hace que la explicación sea clara, no es así como se lo decidió implementar: se usó un único proceso que cumple las funciones de los 3 procesos planteados en el paper. De esta forma, el proceso de un servidor realiza un "MPI_Recv" de cualquier source y tag y, de acuerdo a lo que haya recibido, es de acuerdo a qué proceso del paper actúa.

También, en el servidor se decidió extender el código provisto por la cátedra. Por lo que los nombres de las variables que se seguían necesitando no se modificaron. Sin embargo, para un claro mapeo del pseudocódigo al código, para las nuevas variables se decidió utilizar, cuando fue posible, el mismo nombre que poseen en el paper.

No hay nada más que creamos necesario destacar, el código está debidamente comentado y se mapea, a nuestro entender, fácilmente con el paper.

Criterio de parada del sistema

El criterio de parada del sistema es que todos los servidores siguen corriendo, e intercambiando mensajes, hasta que no quede ningún cliente. Cuando no queda ningún cliente, todos los servidores paran y, así, terminan todos los procesos.

Para lograr esto, todos los servidores tienen una cuenta de cuántos clientes hay corriendo. Cada vez que un cliente deja de correr, el servidor asociado decrementa su contador y le avisa a todo el resto de los servidores para que decrementen sus contadores. Si no queda ningún cliente corriendo, el servidor termina.

Limitaciones

- Cuando uno o más clientes dejan de correr, los servidores asociados siguen corriendo y es necesario, para el correcto funcionamiento del sistema, que se le envíen los mensajes y esperar su respuesta. Esto podría ser evitado, ya que el cliente asociado a ese servidor no está corriendo y no necesita entonces ser notificado cuando otro proceso accede a la sección crítica.

- Se usa la funcion "MPI_Send" que **puede** bloquear hasta que el receptor del mensaje haga el correspondiente "MPI_Recv". En la práctica esto puede pasar, por ejemplo, cuando todos los cientes de todos los servidores piden acceso a la sección crítica simultáneamente. Al hacer esto, se le

manda un mensaje a todo el resto de los servidores usando "MPI_Send". Y si todos los servidores se encuentran ejecutando el "MPI_Send", entonces ninguno se encuentra ejecutando el "MPI_Recv", y esto podría bloquearse indefinidamente. Otro ejemplo que podría causar problema con "MPI_Send" en la práctica es cuando dos clientes, A y B, piden acceso a la sección crítica y el servidor asociado a A, al ejecutar "MPI_Send", le quiere mandar un mensaje al servidor asociado a B mientras que el servidor asociado a B, al ejecutar "MPI_Send", le quiere mandar un mensaje al servidor asociado a A. Es decir, si un servidor C está haciendo un "MPI_Send" a un servidor D mientras que D está haciendo un "MPI_Send" a C.

Igualmente, se preguntó si eran aceptables estas limitaciones a la cátedra, quien dijo que sí. También, vale la pena mencionar que la segunda limitación nunca la pudimos chocar en la práctica con hasta 60 procesos. Y la implementación se probó tanto con mpich2 1.4.1 como con openmpi 1.4.3. Se supone que quizás esta segunda limitación probablemente no ocurra en la práctica porque "MPI_Send" se la usa para mandar buffers vacíos (donde solo el tag interesa) y buffers de a lo sumo un entero.