SISTEMAS CONCURRENTES Y DISTRIBUIDOS

Álvaro Fernández García

-Documentación Productor-consumidor con buffer acotado en MPI-

-Cambios realizados:

1º – Se han eliminado las macros correspondientes al Productor y al Consumidor ; se ha modificado el valor de la macro del Buffer, la cual ahora toma el valor de 5.

2º – A la función “productor” se le ha añadido como parámetro el rank del proceso correspondiente, sirve fundamentalmente para saber que productor se encuentra ejecutándose, además se ha ajustado el número de iteraciones, repartiéndolas entre los 5 productores.

3º – A la función “consumidor” se le ha añadido como parámetro el rank del proceso correspondiente, sirve fundamentalmente para saber que consumidor se encuentra ejecutándose, además se ha ajustado el número de iteraciones, repartiéndolas entre los 4 consumidores.

4º – La función buffer también indica a que proceso le está mandando el mensaje.

5º – Ahora se selecciona al productor (tanto en buffer como en el main) si el valor del rank (o MPI_Source) está comprendido entre 0 y 4.

6º – En la función Buffer, en los MPI_Ssend y MPI_Recv programados dentro del switch, se ha modificado el valor de los procesos fuente/destino de los mensajes por una variable llamada “fuente” la cual contiene el MPI_Source del proceso que solicita el buffer, previamente obtenido de la estructura MPI_Status “status” mediante la llamada a MPI_Probe en cada iteración. La finalidad de esto es conocer qué productor o qué consumidor de los varios existentes está solicitando el servicio para emparejar correctamente los mensajes.

7º – Puesto que se necesita conocer el valor de la fuente para que el buffer empareje correctamente los mensajes, es necesario obtenerla también en los casos pos==0 y pos==TAM, por tanto en ambas ramas se ha programado un MPI_Probe.

8º – Se han sustituido las etiquetas (tag) de 0 que tenían todos los mensajes a una etiqueta específica para cada función:

- 0: mensajes del consumidor.
- 1: mensajes del productor.
- 2: mensajes del buffer.

Estas etiquetas sirven para evitar errores en los MPI_Probe programados dentro del buffer en la rama pos==0 y pos==TAM, para identificar correctamente cuando un mensaje proviene de un consumidor y cuando de un productor, dado que si dejáramos el any tag junto con el any source tal y como aparece en la otra rama (el else) podría darse el caso de que el probe recoja un mensaje erróneo, (por ejemplo que se esté esperando el mensaje de un productor porque el consumidor no puede consumir y se reciba el mensaje de un consumidor, lo cual provocaría un error en la fuente y en la rama que debe ejecutar el buffer).

-Salida del programa:

```
Productor 2 produce valor 0
Productor 0 produce valor 0
Productor 1 produce valor 0
Productor 3 produce valor 0
Productor 4 produce valor 0
Buffer recibe 0 de Productor 2
Productor 2 produce valor 1
Buffer envía 0 a Consumidor 6
Consumidor 6 recibe valor 0 de Buffer
Productor 1 produce valor 1
Buffer recibe 0 de Productor 1
Productor 0 produce valor 1
Consumidor 7 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 7
Buffer recibe 0 de Productor 0
Consumidor 8 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 8
Buffer recibe 0 de Productor 3
Productor 3 produce valor 1
Buffer envía 0 a Consumidor 9
Consumidor 9 recibe valor 0 de Buffer
Productor 4 produce valor 1
Buffer recibe 0 de Productor 4
Consumidor 6 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 6
```

-Documentación Cena de los Filósofos en MPI-

-Descripción de la solución:

Para solucionar este problema se ha optado por lo siguiente:

1º – Los procesos Filósofo se dedican únicamente a mandar mensajes a los Tenedores con la orden MPI_Ssend. Primero mandan un mensaje al proceso que se corresponde con el tenedor a su izquierda (es decir el valor del proceso destino será el que tiene la ID correspondiente al tenedor izquierdo) , a continuación mandan un mensaje al proceso que se encuentra a su derecha, (el proceso destino es el que tiene la ID del tenedor a la derecha). Una vez que los filósofos han comido, se vuelve a repetir la misma secuencia de mensajes mencionada anteriormente, para emular en este caso, que el filósofo está dejando los tenedores. Como es necesario pasar un valor se ha optado por pasar un entero almacenado en una variable llamada "peticion" con el valor 1, no obstante esto no es relevante para la solución. La etiqueta tampoco es relevante así que todos los mensajes se mandan con etiqueta 0. Por otra parte como la función Ssend es bloqueante, esto nos asegura la sincronización y que un filósofo no podrá coger un tenedor hasta que el mismo no reciba el mensaje, (y no lo recibirá si ya está siendo ocupado por otro filósofo).

2º – Por otra parte los procesos tenedores tienen programados en primer lugar un MSI_Probe con cualquier proceso fuente (y también con cualquier etiqueta aunque se podría haber puesto 0 en este caso) puesto que puede recibir una petición por parte del filósofo que está a su izquierda o derecha, a continuación se obtiene a partir de la estructura status la fuente del filósofo que ha realizado la petición, almacenándola en la variable "Filo". Una vez que se conoce cual ha sido el filósofo en pedir el tenedor se puede proceder a programar los MPI_Recv con la información correspondiente, (proceso fuente = Filo, tag = 0). El primer receive sirve para recibir el mensaje correspondiente a la solicitud del tenedor y el segundo para recibir el mensaje correspondiente a soltar el tenedor. Por tanto ambos receives tienen que tener el mismo valor de Filo para que el programa sea correcto. El valor del mensaje recibido se almacena en buf (no es relevante) y nuevamente como MPI_Recv bloquea, se asegura la sincronización.

-Situación de interbloqueo-

Puede producirse una situación de interbloqueo en la siguiente interfoliación de instrucciones:
-Todos los procesos filósofos solicitan el tenedor a su izquierda antes de que ningún otro haya solicitado el tenedor de la derecha.

-Nos encontraríamos en una situación en la que todos los filósofos están esperando el tenedor de la derecha, lo cual es imposible puesto que no quedan más tenedores libres.

Una posible solución consiste en hacer que un filósofo, (en mi caso el proceso 0) solicite los tenedores en orden inverso al resto de los filósofos, es decir primero el izquierdo y luego el derecho.

-Salida del programa-

Filosofo 6 solicita tenedor izquierdo 7
Filosofo 8 solicita tenedor izquierdo 9
Tenedor 7 recibe petición de 6
Filosofo 0 solicita tenedor derecho 9
Filosofo 2 solicita tenedor izquierdo 3
Tenedor 3 recibe petición de 2
Filosofo 4 solicita tenedor izquierdo 5
Filosofo 2 solicita tenedor derecho 1
Tenedor 1 recibe petición de 2
Filosofo 2 COMIENDO
Filosofo 6 solicita tenedor derecho 5
Filosofo 4 solicita tenedor derecho 3
Tenedor 5 recibe petición de 4
Filosofo 8 solicita tenedor derecho 7
Tenedor 9 recibe petición de 8
Filosofo 2 suelta tenedor izquierdo 3
Tenedor 3 recibe liberación de 2
Tenedor 3 recibe petición de 4
Filosofo 2 suelta tenedor derecho 1
Tenedor 1 recibe liberación de 2
Filosofo 4 COMIENDO
Filosofo 2 PENSANDO
Filosofo 2 solicita tenedor izquierdo 3
Filosofo 4 suelta tenedor izquierdo 5
Filosofo 4 suelta tenedor derecho 3
Filosofo 4 PENSANDO

-Documentación Cena de los filósofos con camarero en MPI-

-Descripción de la solución:

Para solucionar este problema se ha optado por lo siguiente:

1º – La función tenedor permanece sin modificaciones con respecto al programa de los filósofos sin camarero.

2º – En la función filósofo se han añadido dos mensajes más con MPI_Ssend. El primero está situado antes de coger los tenedores y simula la petición que hace el filósofo al camarero para sentarse. El segundo esta situado después de soltar ambos tenedores, y recrea la petición al camarero para levantarse. Para que el camarero pueda distinguir cuando un filósofo quiere levantarse o sentarse se han utilizado dos etiquetas distintas:

- 1 : Levantarse.
- 2 : Sentarse.

El resto de la función permanece sin modificaciones con respecto al apartado anterior (salvo la eliminación del filósofo que coge primero el tenedor derecho).

3º – La función camarero presenta las siguientes características:
En primer lugar esta función posee una variable entera nombrada como "num_filo" (inicialmente a 0) la cual lleva la cuenta del número de filósofos que hay sentados en la mesa en dicho instante, por tanto se incrementa en uno cuando el camarero recibe una petición de sentarse y se decrementa ante las peticiones de levantarse.
La función comienza comprobando el estado de la mesa, si ya hay un total de 4 filósofos sentados, no se le debe permitir a ninguno más sentarse, (para evitar bloqueos) por tanto el camarero solo atenderá las peticiones de levantarse, esto está programado con MPI_Probe, de tal forma que admite mensajes de cualquier proceso fuente, pero solo con etiqueta 1 (levantarse). En cualquier otro caso, el camarero atiende cualquier petición, (MPI_Probe con cualquier fuente y etiqueta).
Una vez que se ha detectado algún mensaje, se extrae cual es el Filósofo que está realizando la petición (MPI_SOURCE) y cual es la misma (MPI_TAG). (Esto se extrae a partir del source). Una vez que se conocen estos datos el camarero selecciona que MPI_Recv debe de ejecutar (el de un mensaje de sentarse o de levantarse) mediante el tag, y con la fuente correcta, para el correcto emparejamiento de los mensajes.
Nuevamente, como Ssend y Recv son bloqueantes, se nos asegura la sincronización.

4º – En el main ahora el proceso con rank 10 es el camarero y a las funciones Filósofo y Tenedor, se les pasa como número de procesos 10 (ya que si dejáramos el size (11), los tenedores no serían los correctos).

-Salida del programa-

Filosofo 8 solicita sentarse al camarero
Filosofo 8 se ha SENTADO
Filosofo 8 solicita tenedor izquierdo 9
Camarero recibe la petición de sentarse de Filósofo 8
Filosofo 8 solicita tenedor derecho 7
Tenedor 9 recibe petición de 8
Filosofo 2 solicita sentarse al camarero
Camarero recibe la petición de sentarse de Filósofo 2
Filosofo 6 solicita sentarse al camarero
Camarero recibe la petición de sentarse de Filósofo 6
Filosofo 0 solicita sentarse al camarero
Filosofo 4 solicita sentarse al camarero
Filosofo 6 se ha SENTADO
Filosofo 6 solicita tenedor izquierdo 7
Filosofo 2 se ha SENTADO
Filosofo 2 solicita tenedor izquierdo 3
Tenedor 3 recibe petición de 2
Filosofo 2 solicita tenedor derecho 1
Camarero recibe la petición de sentarse de Filósofo 0
Filosofo 0 se ha SENTADO
Filosofo 0 solicita tenedor izquierdo 1
Tenedor 7 recibe petición de 8
Filosofo 8 COMIENDO
Tenedor 1 recibe petición de 2
Filosofo 2 COMIENDO
Filosofo 8 suelta tenedor izquierdo 9
Filosofo 8 suelta tenedor derecho 7
Filosofo 8 solicita levantarse al camarero
Tenedor 7 recibe liberación de 8
Tenedor 7 recibe petición de 6
Tenedor 9 recibe liberación de 8
Filosofo 6 solicita tenedor derecho 5
Filosofo 8 se ha LEVANTADO
Filosofo 8 PENSANDO