Productor-consumidor con múltiples prod. /cons.

Se han creado dos nuevas constantes enteras, que serán las etiquetas correspondientes de los productores y consumidores: etiq_prod = 0 y etiq_cons = 1; otras dos constantes que definen el número de productores y consumidores, la variable que guarda el número de procesos esperados es ahora la suma del número de productores más el de consumidores más uno (el buffer), el id del buffer pasa a ser 4, ya que tenemos 4 cuatro productores (sus id van de 0 a 3); por último, se ha creado una variable, k, que define el número de elementos que debe producir cada productor.

En la función producir, se ha añadido un nuevo parámetro, orden, que es un entero que define el número de orden del proceso que la llama. Dentro de esta función, la variable contador, será el producto de la variable k por el número de orden.

En la función funcion_productor, se ha añadido un nuevo parámetro, num_orden, con el mismo propósito que en la función anterior. En este caso, el bucle, tendrá de límite el valor de k y a la hora de usar la función MPI_Ssend, se usará como etiqueta, etiq_prod, para indicar que el mensaje viene de un productor.

En la función funcion_consumidor, se realizan los mismos cambios que en la funcion_productor, solo que en este caso usaremos como etiqueta, etiqu_cons

En la función funcion_buffer, se tendrá en cuenta como emisor aceptable las etiquetas de los productores y consumidores.

Problema de los filósofos con interbloqueo y posible solución

En este caso, se ha creado un programa que resuelve el problema de los filósofos, pero aún así, se puede dar el caso de que hay interbloqueo. Ya que todos los filósofos, inicialmente cogerán el tenedor a su izquierda y luego el de su derecha, el interbloqueo se producirá cuando todos ellos cojan a la vez (o casi a la misma vez) el tenedor de la izquierda. Como el tenedor situado a la izquierda de un filosofo es a la vez el de la derecha del siguiente filosofo, si ningún filosofo suelta el tenedor de la izquierda es imposible que los procesos continúen su ejecución.

Para solucionar el problema del interbloqueo, podemos hacer que un proceso cualquiera comience cogiendo el tenedor situado a su derecha. Por ejemplo, supongamos que el filósofo 0, comienza cogiendo el tenedor a su derecha (o al menos lo intenta). En el hipotético caso de que el filósofo 0 sea más rápido que el filosofo 4 a la hora de coger el tenedor, el filósofo 4 deberá esperar a que ese tenedor (que correspondería al de su izquierda) este libre para coger el de la derecha. Por el otro lado, si el filósofo 4 es más rápido en coger su tenedor, el filósofo 0, deberá esperar, dejando libre el tenedor a su derecha para ser utilizado por el filósofo 1.

De esta forma, siempre nos aseguraremos de que hay un tenedor libre para ser usado por otro filósofo evitando una posible situación de interbloqueo.

Problema de los filósofos con camarero

Para este caso, se ha creado un nuevo proceso, camarero, que controla el número de filósofos que hay sentados en la mesa. Un filósofo, antes de coger ningún tenedor, deberá solicitar al camarero sentarse en una mesa. A la hora de implementarlo, el filósofo, enviara un mensaje al camarero al inicio. El camarero, por su lado, irá llevando la cuenta de cuando filósofos hay sentados en todo momento y comprobará en cada iteración, si tiene alguna solicitud pendiente (implementado con MPI_PROBE). En caso de que así sea, si el número de filósofos es menor a 4, aceptará cualquier tipo de solicitud, ya sea de sentarse o levantarse, en caso contrario, solo aceptará solicitudes para levantarse. Por el otro lado, el filósofo, deberá esperar por la confirmación del camarero para poder sentarse.