Portafolio práctica 4 SCD

Prod-Cons 2 multiple

Cambios que he realizado

- Ahora el que un proceso sea productor/consumidor se decide en el main por si es par o impar el id del proceso, los productores son pares mientras que los consumidores son impares. Además el id del buffer está predefinida a 8 con una variable global y se han modificado las cabeceras de las funciones para que reciban como argumento su ID correspondiente.

```
if ( id_propio == id_buffer)
    funcion_buffer();
else if ( id_propio % 2 )     // Si es impar es consumidor
    funcion_consumidor(id_propio);
else                          // Si es par y no es el 10, se productor
    funcion_productor(id_propio);
```

  Asi como se han creado tres constantes globales llamdas:

```
valores_por_productor = num_items / NUM_PRODUCTORES ,
NUM_PRODUCTORES       = 4 ,
NUM_CONSUMIDORES      = 5 ,
etiq_consumidor       = 0 ,
etiq_productor        = 1 ,
```

- La función producir dato ahora tiene el contador inicializado a:

```
static int contador = (id_productor / 2) * valores_por_productor ;
```

  Y el for interno de la funcion producir se ejecuta ahora hasta `valores_por_productor`.

- En la función buffer, ahora siempre se recibe de `MPI_ANY_SOURCE` pero varía la etiqueta en función del estado del buffer. Ej:

```
MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE,
etiq_consumidor,MPI_COMM_WORLD);
```

  La etiqueta de la que se recibe se decide por el siguiente condicional:

```

```

```
if ( num_celdas_ocupadas == 0 )               // si buffer vacío
    etiq_emisor_aceptable = etiq_productor ;       // $~~~$ solo prod.
else if ( num_celdas_ocupadas == tam_vector ) // si buffer lleno
    etiq_emisor_aceptable = etiq_consumidor ;      // $~~~$ solo cons.
else                                          // si no vacío ni lleno
    etiq_emisor_aceptable = MPI_ANY_SOURCE ;     // $~~~$ cualquiera
```

Del mismo modo, las comunicaciones con el buffer ahora se hacen únicamente a través de la etiqueta correspondiente: Ej:

```
MPI_Ssend( &valor_prod, 1, MPI_INT, id_buffer, etiq_productor;
MPI_COMM_WORLD);
```

Listado parcial de la salida del programa

```
Productor: 0 ha producido valor 5
Productor: 0 va a enviar valor 5
PRODUCTOR: 0 FINALIZÓ
Buffer ha recibido valor 5
Consumidor ha consumido valor 17
Buffer va a enviar valor 18
Consumidor: 3 ha recibido valor 18
Consumidor ha consumido valor 9
Buffer va a enviar valor 13
Consumidor: 4 ha recibido valor 13
Consumidor ha consumido valor 8
Buffer va a enviar valor 19
Consumidor: 2 ha recibido valor 19
Consumidor ha consumido valor 12
Buffer va a enviar valor 20
Consumidor: 1 ha recibido valor 20
Consumidor ha consumido valor 3
Buffer va a enviar valor 10
Consumidor: 0 ha recibido valor 10
Consumidor ha consumido valor 18
Buffer va a enviar valor 14
Consumidor: 3 ha recibido valor 14
Consumidor ha consumido valor 19
Buffer va a enviar valor 4
Consumidor: 2 ha recibido valor 4
Consumidor ha consumido valor 20
Buffer va a enviar valor 15
Consumidor: 1 ha recibido valor 15
```

> Consumidor ha consumido valor 10
> CONSUMIDOR: 0 FINALIZÓ

Filosofos con y sin interbloqueo

Puntos destacables de ambas implementaciones

Con interbloqueo

¿Qué provoca el interbloqueo? El hecho de que, si los 5 filosofos se sientan simultaneamente y casualmente solicitan el tenedor de la izquierda simultaneamente sin que ninguno antes haya obtenido el de la derecha, da lugar a que ninguno pueda obtener su tenedor derecho (ya que es el izquierdo de otro filosofo que a su vez está solicitando su derecho) y quedarán interbloqueados.

Solución al interbloqueo

La solución que se ha planteado es hacer que un único filosofo con una id de proceso concreta obtenga sus tenedores en orden inverso. Esto hará que nunca haya tantos filosofos como tenedores solicitando bloqueando distintos tenedores, ya que siempre habŕa uno que, al intentar obtener su primer tenedor este ya esté bloqueado y tenga que esperar a que aquel que lo bloqueó termine. Esto se consigue con el siguiente condicional:

```
    if ( id == 0 )  // Si es el primer filósofo, que coja primero el
derecho y luego el izquierdo
    {
        cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der
<<endl;
        // ... solicitar tenedor derecho primero
        MPI_Ssend(&valor, 1, MPI_INT, id_ten_der, 0, MPI_COMM_WORLD);

        cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq
<<endl;
        // ... solicitar tenedor izquierdo después
        MPI_Ssend(&valor, 1, MPI_INT, id_ten_izq, 0, MPI_COMM_WORLD);

    }
    else              // Si es cualquier otro, primero el izquierdo y luego
el derecho
    {
        // [... coge el izquierdo primero ...]
    }
```

Filosofos con caramero

En esta solución lo que se hace es recurrir a un proceso auxiliar que tiene un comportamiento similar al del buffer del primer ejercicio y que evita que se sienten más de n-1 filósofos simultaneamente a una mesa de n asientos. Esto lo hace con llamdas síncronas a los filósofos solicitantes.

Listado parcial de la salida del programa

Ejemplo de interbloqueo:

```
Filósofo 0 solicita ten. izq.1
Filósofo 6 solicita ten. izq.7
Filósofo 8 solicita ten. izq.9
Filósofo 2 solicita ten. izq.3
Filósofo 4 solicita ten. izq.5
Filósofo 8 solicita ten. der.7
Filósofo 0 solicita ten. der.9
Ten. 7 ha sido cogido por filo. 6
Filósofo 6 solicita ten. der.5
Ten. 9 ha sido cogido por filo. 8
Ten. 1 ha sido cogido por filo. 0
Filósofo 4 solicita ten. der.3
Ten. 3 ha sido cogido por filo. 2
Filósofo 2 solicita ten. der.1
Ten. 5 ha sido cogido por filo. 4
```

Ejemplo salida filosofo con camarero:

```
Filósofo 6 comienza a comer
Ten. 3 ha sido liberado por filo. 4
Filósofo 4 se levanta de la mesa
Filosofo 4 comienza a pensar
Filósofo 2 se sienta a la mesa.
Filósofo 2 solicita ten. izq.3
Ten. 3 ha sido cogido por filo. 2
Filósofo 2 solicita ten. der.1
Filósofo 6 suelta ten. izq. 7
Ten. 7 ha sido liberado por filo. 6
Ten. 7 ha sido cogido por filo. 8
Filósofo 8 comienza a comer
Filósofo 6 suelta ten. der. 5
Filósofo 6 se levanta de la mesa
Ten. 5 ha sido liberado por filo. 6
Filosofo 6 comienza a pensar
Filósofo 4 se sienta a la mesa.
```