Entrega: Práctica 3 – SCD

Alumno: Fernando Cuesta Bueno – 2º Ing. Informática A1

Ejercicio 1: Múltiples productores y consumidores – Buffer FIFO

Los cambios más reseñables respecto a la implementación dada en “prodcons2.cpp” son:

La introducción de varios procesos productores y varios procesos consumidores.

```
26  const int
27     NUM_PRODUCTORES = 4,    // el programa debe ser correcto aun usando
28     NUM_CONSUMIDORES = 5;   // otros valores
29
```

La aparición de las etiquetas según si se trata de un proceso productor, consumidor o el buffer.

```
30  const int
31     etiq_productor           = 0 ,
32     etiq_consumidor          = 1 ,
33     etiq_buffer              = 2 ,
```

Un vector para saber cuántos elementos ha producido cada proceso.

```
41  // Vector con el número de items producidos por cada hebra productora
42  int items_producidos[NUM_PRODUCTORES] = {0};
43
```

Las funciones de los procesos productores y consumidores ya no trabajan hasta “num_items”, ahora trabajan hasta “np” (num_items / NUM_PRODUCTORES) y “nc” (num_items/ NUM_CONSUMIDORES).

La función del buffer recibe el mensaje según la etiqueta aceptable, en vez de según el id aceptable.

```
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_TAG ;     // $~~~$ cualquiera
```

El resto del código se encuentra en el fichero “prodcons2-mu.cpp”.

Un listado parcial de la salida del programa es el siguiente:

```
mpirun -oversubscribe  -np  3 ./prodcons2_mpi_exe
Buffer ha recibido valor 1
Buffer va a enviar valor 1
Consumidor ha recibido valor 1
Productor ha producido valor 1
Productor va a enviar valor 1
Buffer ha recibido valor 2
Productor ha producido valor 2
Productor va a enviar valor 2
Buffer ha recibido valor 3
Productor ha producido valor 3
Productor va a enviar valor 3
Buffer ha recibido valor 4
Productor ha producido valor 4
Productor va a enviar valor 4
Buffer ha recibido valor 5
Productor ha producido valor 5
Productor va a enviar valor 5
Buffer va a enviar valor 2
Consumidor ha consumido valor 1
Consumidor ha recibido valor 2
Productor ha producido valor 6
Productor va a enviar valor 6
Buffer ha recibido valor 6
Buffer ha recibido valor 7
Productor ha producido valor 7
Productor va a enviar valor 7
Buffer ha recibido valor 8
Productor ha producido valor 8
Productor va a enviar valor 8
Buffer va a enviar valor 3
Consumidor ha consumido valor 2
Consumidor ha recibido valor 3
Productor ha producido valor 9
Productor va a enviar valor 9
Buffer ha recibido valor 9
Productor ha producido valor 10
Productor va a enviar valor 10
Buffer ha recibido valor 10
Productor ha producido valor 11
Productor va a enviar valor 11
Buffer ha recibido valor 11
Consumidor ha consumido valor 3
Consumidor ha recibido valor 4
Buffer va a enviar valor 4
Productor ha producido valor 12
Productor va a enviar valor 12
Buffer ha recibido valor 12
Productor ha producido valor 13
Productor va a enviar valor 13
```

Ejercicio 2 y 3: Cena de los filósofos con y sin interbloqueo

Respecto a la plantilla suministrada en “filosofos-plantilla.cpp” hemos introducido los mensajes de solicitud de los tenedores, así como soltarlos tras haber comido.

```
53    while ( true )
54    {
55      // solicitar tenedor izquierdo
56      cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
57      MPI_Ssend( &peticion, 1, MPI_INT, id_ten_izq, etiq_tenedor, MPI_COMM_WORLD );
58
59      // solicitar tenedor derecho
60      cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
61      MPI_Ssend( &peticion, 1, MPI_INT, id_ten_der, etiq_tenedor, MPI_COMM_WORLD );
62
63      cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;
64      sleep_for( milliseconds( aleatorio<10,100>() ) );
65
66      // soltar el tenedor izquierdo
67      cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id_ten_izq <<endl;
68      MPI_Ssend( &peticion, 1, MPI_INT, id_ten_izq, etiq_tenedor, MPI_COMM_WORLD );
69
70      // soltar el tenedor derecho
71      cout<< "Filósofo " <<id <<" suelta ten. der. " <<id_ten_der <<endl;
72      MPI_Ssend( &peticion, 1, MPI_INT, id_ten_der, etiq_tenedor, MPI_COMM_WORLD );
73
74      cout << "Filosofo " << id << " comienza a pensar" << endl;
75      sleep_for( milliseconds( aleatorio<10,100>() ) );
76    }
77  }
```

Dentro de la función de los tenedores recibimos la petición del filósofo para ser cogidos (línea 88), así como obtenemos el id del filósofo a partir del campo MPI_SOURCE de estado (línea 91). Finalmente, recibimos la petición para liberar el tenedor (línea 96).

```
78  // ---------------------------------------------------------------------
79
80  void funcion_tenedores( int id )
81  {
82      int valor, id_filosofo ;  // valor recibido, identificador del filósofo
83      MPI_Status estado ;       // metadatos de las dos recepciones
84
85      while( true )
86      {
87          // recibir petición de cualquier filósofo
88          MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, etiq_tenedor, MPI_COMM_WORLD, &estado );
89
90          // guardar en 'id_filosofo' el id. del emisor
91          id_filosofo = estado.MPI_SOURCE;
92          cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;
93
94          // recibir liberación de filósofo 'id_filosofo'
95          MPI_Recv( &valor, 1, MPI_INT, id_filosofo, etiq_tenedor, MPI_COMM_WORLD, &estado );
96          cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id_filosofo <<endl ;
97      }
98  }
```

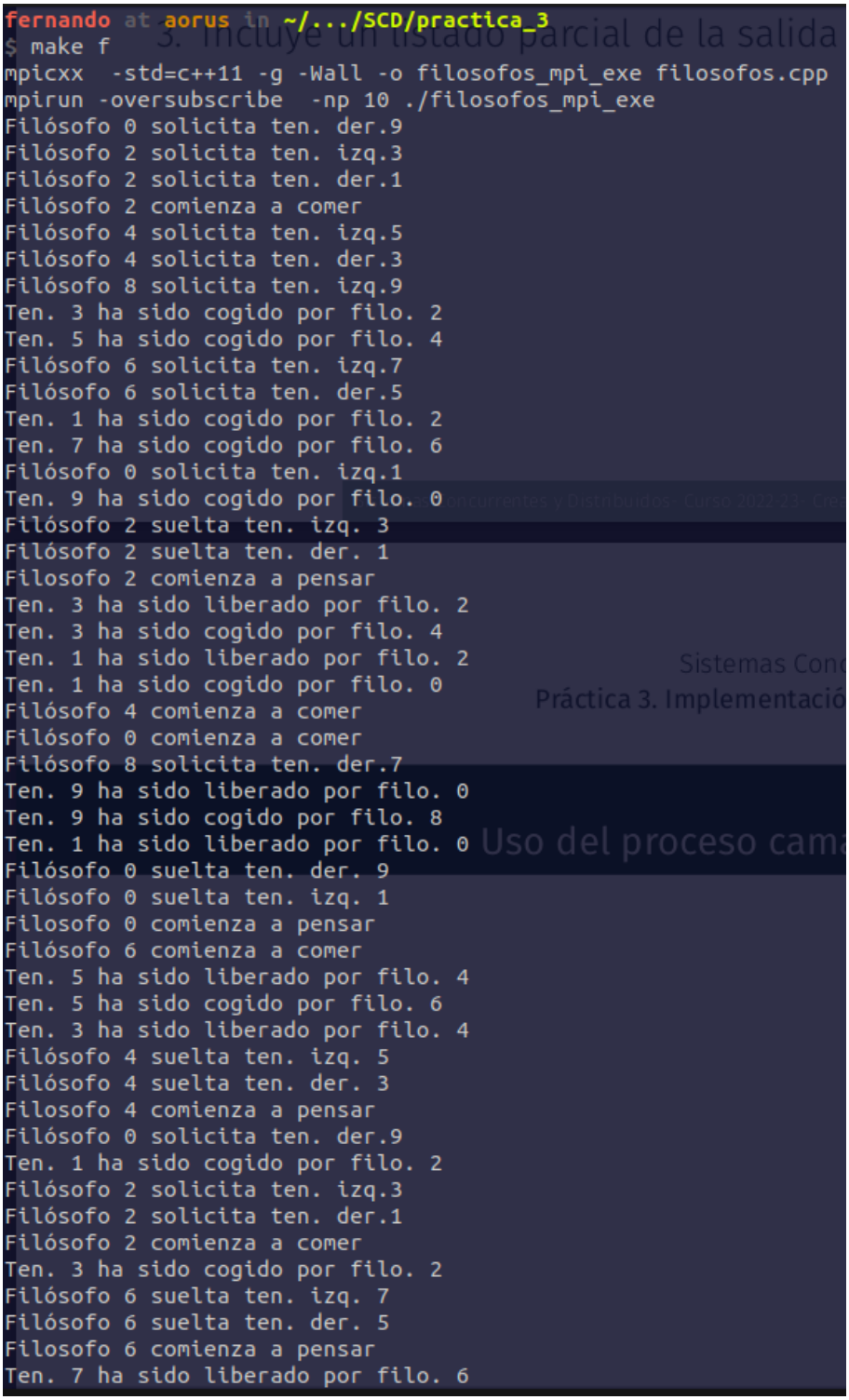
En el fichero “filosofos-interb.cpp” se encuentra la implementación que puede dar lugar a un interbloqueo y en “filosofos.cpp” la solución donde evitamos este interbloqueo.

Finalmente, mostramos la salida por pantalla del programa:

```
fernando at aorus in ~/.../SCD/practica_3
$ make fi
mpicxx  -std=c++11 -g -Wall -o filosofos-interb_mpi_exe filosofos-interb.cpp
mpirun -oversubscribe  -np 10 ./filosofos-interb_mpi_exe
Filósofo 6 solicita ten. izq.7
Filósofo 4 solicita ten. izq.5
Filósofo 6 solicita ten. der.5
Ten. 7 ha sido cogido por filo. 6
Filósofo 8 solicita ten. izq.9
Filósofo 0 solicita ten. izq.1
Filósofo 4 solicita ten. der.3
Ten. 5 ha sido cogido por filo. 4
Filósofo 4 comienza a comer
Filósofo 2 solicita ten. izq.3
Ten. 3 ha sido cogido por filo. 4
Filósofo 0 solicita ten. der.9
Filósofo 8 solicita ten. der.7
Ten. 9 ha sido cogido por filo. 8
Ten. 1 ha sido cogido por filo. 0
Ten. 3 ha sido liberado por filo. 4
Ten. 3 ha sido cogido por filo. 2
Ten. 5 ha sido liberado por filo. 4
Ten. 5 ha sido cogido por filo. 6
Filósofo 6 comienza a comer
Filósofo 4 suelta ten. izq. 5
Filósofo 4 suelta ten. der. 3
Filosofo 4 comienza a pensar
Filósofo 2 solicita ten. der.1
Ten. 5 ha sido liberado por filo. 6
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. izq. 7
Filósofo 6 suelta ten. der. 5
Filosofo 6 comienza a pensar
Filósofo 4 solicita ten. izq.5
Filósofo 4 solicita ten. der.3
Ten. 5 ha sido cogido por filo. 4
Filósofo 6 solicita ten. izq.7
Filósofo 8 suelta ten. izq. 9
Filósofo 8 suelta ten. der. 7
Filosofo 8 comienza a pensar
Ten. 9 ha sido liberado por filo. 8
Ten. 9 ha sido cogido por filo. 0
Filósofo 6 solicita ten. der.5
Ten. 7 ha sido liberado por filo. 8
Ten. 7 ha sido cogido por filo. 6
Filósofo 0 comienza a comer
Filósofo 8 solicita ten. izq.9
Filósofo 8 solicita ten. der.7
Filósofo 0 suelta ten. izq. 1
Filósofo 0 suelta ten. der. 9
```

En la solución sin interbloqueo hemos implementado un ligero cambio: uno de los filósofos empezará cogiendo los tenedores en orden inverso, es decir, primero cogerá el tenedor derecho y luego el izquierdo.

La traza de esta implementación es la siguiente:



Como podemos observar, el primer filósofo, correspondiente al proceso 0, empieza pidiendo el tenedor derecho y luego el izquierdo. El resto de procesos actuan como en la implementación anterior.

Respecto al código hemos añadido una nueva función que únicamente ejecuta el primer proceso:

```
144     if( num_procesos == num_procesos_actual )
145     {
146       // ejecutar la función correspondiente a 'id_propio'
147       if( id_propio == 0 )                          // si es el primer proceso
148         funcion_filosofo_invertido( id_propio );    //   es el filosofo que coge los cubiertos e
149       else if ( id_propio % 2 == 0 && id_propio != 0)   // si es par y no es el primer proceso
150         funcion_filosofos( id_propio );             //   es un filósofo normal
151       else                                          // si es impar
152         funcion_tenedores( id_propio );             //   es un tenedor
153     }
```

Ejercicio 4: Cena de los filósofos sin interbloqueo y con camarero

Para impremantar el camarero central hemos implementado una nueva función:

```
115   void funcion_camarero(  )
116   {
117       int valor,                  // valor recibido
118           cantidad = 0,           // número de filósofos entados en cada momento
119           etiq_emisor_aceptable;  // etiqueta del emisor del que se puede rescibir un mensaje
120
121       MPI_Status estado ;         // metadatos de las dos recepciones
122
123       while( true )
124       {
125           if( cantidad == 4 )                         // si ya hay 4 filósofos sentados
126               etiq_emisor_aceptable = etiq_levantarse;    //  solo se pueden levantar
127           else                                        // si hay menos de 4 filósofos
128               etiq_emisor_aceptable = MPI_ANY_TAG;        //  se pueden levantar o sentar
129
130           MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, etiq_emisor_aceptable, MPI_COMM_WORLD, &estado );
131
132           if( estado.MPI_TAG == etiq_sentarse )
133           {
134               cantidad++;
135               cout << "Camarero: se sienta otro filósofo a comer. Ahora hay " << cantidad << endl;
136           }
137           else
138           {
139               cantidad--;
140               cout << "Camarero: se levanta un filósofo de la mesa. Ahora hay " << cantidad << endl;
141           }
142       }
143   }
```

Esta es ejecutada por un único proceso:

```
155     if( num_procesos == num_procesos_actual )
156     {
157       // ejecutar la función correspondiente a 'id_propio'
158       if( id_propio == id_camarero )    // si es el último proceso
159         funcion_camarero(  );           //   es el camarero
160       else if ( id_propio % 2 == 0 )    // si es par
161         funcion_filosofos( id_propio ); //   es un filósofo
162       else                              // si es impar
163         funcion_tenedores( id_propio ); //   es un tenedor
164     }
```

Otro cambio reseñable es en la función de los filósofos, estos solicitan al camarero sentarse y levantarse:

```cpp
void funcion_filosofos( int id )
{
  int id_ten_izq = (id+1)              % num_filo_ten, //id. tenedor izq.
      id_ten_der = (id+num_filo_ten-1) % num_filo_ten, //id. tenedor der.
      peticion;

  while ( true )
  {
    // 1. Sentarse
    MPI_Ssend( &peticion, 1, MPI_INT, id_camarero, etiq_sentarse, MPI_COMM_WORLD );
    cout << "Filósofo " << id << " se sienta a comer." << endl;

    // 2. Tomar tenedores
    // solicitar tenedor izquierdo
    cout << "Filósofo " << id << " solicita ten. izq." << id_ten_izq << endl;
    MPI_Ssend( &peticion, 1, MPI_INT, id_ten_izq, etiq_tenedor, MPI_COMM_WORLD );

    // solicitar tenedor derecho
    cout << "Filósofo " << id << " solicita ten. der." << id_ten_der << endl;
    MPI_Ssend( &peticion, 1, MPI_INT, id_ten_der, etiq_tenedor, MPI_COMM_WORLD );

    // 3. Comer
    cout << "Filósofo " << id << " comienza a comer" << endl ;
    sleep_for( milliseconds( aleatorio<10,100>() ) );

    // 4. Soltar tenedores
    // soltar el tenedor izquierdo
    cout << "Filósofo " << id << " suelta ten. izq. " << id_ten_izq << endl;
    MPI_Ssend( &peticion, 1, MPI_INT, id_ten_izq, etiq_tenedor, MPI_COMM_WORLD );

    // soltar el tenedor derecho
    cout << "Filósofo " << id << " suelta ten. der. " << id_ten_der << endl;
    MPI_Ssend( &peticion, 1, MPI_INT, id_ten_der, etiq_tenedor, MPI_COMM_WORLD );

    // 5. Levantarse
    MPI_Ssend( &peticion, 1, MPI_INT, id_camarero, etiq_levantarse, MPI_COMM_WORLD );
    cout << "Filósofo " << id << " se levanta de la mesa." << endl;

    // 6. Pensar
    cout << "Filosofo " << id << " comienza a pensar" << endl;
    sleep_for( milliseconds( aleatorio<10,100>() ) );
  }
}
```

En “filosofos-cam.cpp” se encuentra el resto del código del ejercicio.

Un listado parcial de la salida del programa es el siguiente:

```
fernando at aorus in ~/.../SCD/practica_3
$ make fc
mpirun -oversubscribe -np 11 ./filosofos-cam_mpi_exe
Filósofo 6 se sienta a comer.
Filósofo 6 solicita ten. izq.7
Filósofo 6 solicita ten. der.5
Camarero: se sienta otro filósofo a comer. Ahora hay 1
Camarero: se sienta otro filósofo a comer. Ahora hay 2
Camarero: se sienta otro filósofo a comer. Ahora hay 3
Camarero: se sienta otro filósofo a comer. Ahora hay 4
Ten. 7 ha sido cogido por filo. 6
Filósofo 8 se sienta a comer.
Filósofo 8 solicita ten. izq.9
Filósofo 8 solicita ten. der.7
Ten. 9 ha sido cogido por filo. 8
Filósofo 6 comienza a comer
Filósofo 0 se sienta a comer.
Filósofo 0 solicita ten. izq.1
Filósofo 0 solicita ten. der.9
Filósofo 2 se sienta a comer.
Filósofo 2 solicita ten. izq.3
Filósofo 2 solicita ten. der.1
Ten. 5 ha sido cogido por filo. 6
Ten. 1 ha sido cogido por filo. 0
Ten. 3 ha sido cogido por filo. 2
Filósofo 6 suelta ten. izq. 7
Filósofo 6 suelta ten. der. 5
Filósofo 6 se levanta de la mesa.
Filosofo 6 comienza a pensar
Camarero: se levanta un filósofo de la mesa. Ahora hay 3
Camarero: se sienta otro filósofo a comer. Ahora hay 4
Ten. 7 ha sido liberado por filo. 6
Ten. 7 ha sido cogido por filo. 8
Filósofo 8 comienza a comer
Filósofo 4 se sienta a comer.
Filósofo 4 solicita ten. izq.5
Filósofo 4 solicita ten. der.3
Ten. 5 ha sido liberado por filo. 6
Ten. 5 ha sido cogido por filo. 4
Filósofo 8 suelta ten. izq. 9
Filósofo 8 suelta ten. der. 7
Filósofo 8 se levanta de la mesa.
Filosofo 8 comienza a pensar
Ten. 7 ha sido liberado por filo. 8
Ten. 9 ha sido liberado por filo. 8
Ten. 9 ha sido cogido por filo. 0
Camarero: se levanta un filósofo de la mesa. Ahora hay 3
Filósofo 0 comienza a comer
Filósofo 0 suelta ten. izq. 1
Filósofo 0 suelta ten. der. 9
Ten. 1 ha sido liberado por filo. 0
```