SISTEMAS CONCURRENTES Y DISTRIBUIDOS.

Pablo Manresa Nebot. 2ºA2

Índice

Problema Productor-consumidor 1
Problema filósofos con interbloqueo y solución 5
Problema filósofos con camarero 6

Problema del productor- consumidor para múltiples productores y consumidores.

Uno de los principales cambios es añadir 5 consumidores y 4 productores.

Además, el número de procesos esperados es igual al número de consumidores por el número de productores más 1, siendo éste último para el proceso buffer.

También se ha añadido una etiqueta para los consumidores y otra para los productores, y de este modo diferenciar los mensajes por categorías.

```
const int np = 4,
                    nc = 5;

const int etiq_productores = 1,
                    etiq_consumidores = 2;

const int
  id_buffer           = 4 ,
  num_procesos_esperado = np+nc+1 ,
  num_items           = np*nc,
  tam_vector          = 10;
```

Otro cambio es el número de iteraciones en las funciones productor y consumidor siendo el num_items entre el número de procesos destinados para dicha función.

```
void funcion_consumidor(int orden){
        ……..
        for( unsigned int i=0 ; i < num_items/nc; i++ )
        ……..
}

void funcion_productor(int orden){
        ……..
        for( unsigned int i=0 ; i < num_items/np; i++ )
        ……..
}
```

En las funciones consumidor y productor, se han añadido en cada Ssend sus respectivas etiquetas para que se envíe a las categorias a las que pertenecen.

```
void funcion_productor(int orden){
……
MPI_Ssend( &valor_prod, 1, MPI_INT, id_buffer, etiq_productores, MPI_COMM_WORLD );
}

void funcion_consumidor(int orden)
……
MPI_Ssend( &peticion,  1, MPI_INT, id_buffer, etiq_consumidores, MPI_COMM_WORLD);
……
}
```

En cuanto a la función buffer, se ha añadido:

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

Para seleccionar si se va a escoger la categoría de consumidores, productores o ambas.

El siguiente cambio consiste en:

```
MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, id_emisor_aceptable, MPI_COMM_WORLD, &estado );
```

Que consta de añadir MPI_ANY_SOURCE para recibir de cualquier fuente y id_emisor_aceptable para que reciba de la etiqueta seleccionada anteriormente.

El siguiente cambio:

```
switch( estado.MPI_TAG ) // leer emisor del mensaje en metadatos
```

Para filtrar por etiqueta seleccionada:

```
case etiq_productores: // si ha sido el productor: insertar en buffer
          buffer[primera_libre] = valor ;
          primera_libre = (primera_libre+1) % tam_vector ;
          num_celdas_ocupadas++ ;
          cout << "Buffer ha recibido valor " << valor << endl ;
          break;

        case etiq_consumidores: // si ha sido el consumidor: extraer y enviarle
          valor = buffer[primera_ocupada] ;
          primera_ocupada = (primera_ocupada+1) % tam_vector ;
          num_celdas_ocupadas-- ;
          cout << "Buffer va a enviar valor " << valor << endl ;
          MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, 0, MPI_COMM_WORLD);
          break;
```

y, MPI_Ssend(&valor, 1, MPI_INT, estado.MPI_SOURCE, 0, MPI_COMM_WORLD);
estado.MPI_SOURCE para seleccionar al emisor del mensaje.

Finalmente, el main quedaría así:

```
if ( ( id_propio >=0 ) && ( id_propio <= np-1 ) )
      funcion_productor(id_propio%np+1);
    else if ( id_propio == id_buffer )
      funcion_buffer();
    else
      funcion_consumidor(id_propio%nc+1);
```

Si el valor está entre 0 y np-1 es un proceso productor, si es igual valor del identificador buffer es el proceso buffer, en otro caso, será para el proceso consumidor.

Listado parcial de la salida del programa sería:

Productor ha producido valor 6
Productor va a enviar valor 6
Buffer ha recibido valor 6
Buffer va a enviar valor 6
Consumidor ha recibido valor 6
Productor ha producido valor 11
Productor va a enviar valor 11
Buffer ha recibido valor 11
Buffer va a enviar valor 11
Consumidor ha recibido valor 11
Productor ha producido valor 16
Productor va a enviar valor 16
Buffer ha recibido valor 16
Buffer va a enviar valor 16
Consumidor ha recibido valor 16
Productor ha producido valor 1
Productor va a enviar valor 1
Buffer ha recibido valor 1
Buffer va a enviar valor 1
Consumidor ha recibido valor 1
Productor ha producido valor 12
Productor va a enviar valor 12
Buffer ha recibido valor 12
Buffer va a enviar valor 12
Consumidor ha recibido valor 12
Productor ha producido valor 7
Productor va a enviar valor 7
Buffer ha recibido valor 7
Productor ha producido valor 8
Productor va a enviar valor 8
Buffer ha recibido valor 8
Productor ha producido valor 9
Productor va a enviar valor 9
Buffer ha recibido valor 9
Productor ha producido valor 17
Productor va a enviar valor 17
Buffer ha recibido valor 17
Productor ha producido valor 2
Productor va a enviar valor 2
Buffer ha recibido valor 2
Productor ha producido valor 13
Productor va a enviar valor 13
Buffer ha recibido valor 13
Consumidor ha consumido valor 6
Consumidor ha recibido valor 7
Buffer va a enviar valor 7
Productor ha producido valor 18
Productor va a enviar valor 18
Buffer ha recibido valor 18
Productor ha producido valor 3
Productor va a enviar valor 3
Buffer ha recibido valor 3
Productor ha producido valor 14
Productor va a enviar valor 14
Buffer ha recibido valor 14
Consumidor ha consumido valor 1
Consumidor ha recibido valor 8

Buffer va a enviar valor 8
Productor ha producido valor 10
Productor va a enviar valor 10
Buffer ha recibido valor 10
Consumidor ha consumido valor 11
Consumidor ha recibido valor 9
Buffer va a enviar valor 9
Productor ha producido valor 4
Productor va a enviar valor 4
Buffer ha recibido valor 4
Consumidor ha consumido valor 16
Consumidor ha recibido valor 17
Buffer va a enviar valor 17
Productor ha producido valor 15
Productor va a enviar valor 15

Problema de los filósofos con interbloqueo y su solución.

Puesto que, cada filósofo intenta coger primero el tenedor de la izquierda, puede ocurrir que, al inicio, cada filósofo coja primero dicho tenedor. En consecuencia, todos los filósofos quedarán bloqueado, debido a que, ninguno podrá empezar a comer con un sólo tenedor, por lo que, nunca se liberarán.

La **solución propuesta** es la siguiente consiste en hacer que uno de los filósofos, solicite los tenedores al contrario, es decir, si el id del primero es igual a 0, entonces solicita primero el de la derecha, en otro caso, solicitará el de la izquierda.

```
if(id == 0){
                cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
                // ... solicitar tenedor derecho (completar)
                MPI_Ssend(NULL, 0, MPI_INT, id_ten_der, etiq_solicitar_tenedor, MPI_COMM_WORLD);

                cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
                // ... solicitar tenedor izquierdo (completar)
                MPI_Ssend(NULL, 0, MPI_INT, id_ten_izq, etiq_solicitar_tenedor, MPI_COMM_WORLD);
        }
        else    {
                cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
                // ... solicitar tenedor izquierdo (completar)
                MPI_Ssend(NULL, 0, MPI_INT, id_ten_izq, etiq_solicitar_tenedor, MPI_COMM_WORLD);

                cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
                // ... solicitar tenedor derecho (completar)
                MPI_Ssend(NULL, 0, MPI_INT, id_ten_der, etiq_solicitar_tenedor, MPI_COMM_WORLD);
}
```

Listado parcial de la salida del programa sería:

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

Ten. 7 ha sido cogido por filo. 6
Filósofo 8 solicita ten. izq.9
Filósofo 6 comienza a comer
Ten. 5 ha sido cogido por filo. 6
Filósofo 2 solicita ten. izq.3
Filósofo 0 comienza a comer
Ten. 1 ha sido cogido por filo. 0
Filósofo 2 solicita ten. der.1
Ten. 3 ha sido cogido por filo. 2
Ten. 5 ha sido liberado por filo. 6
Ten. 5 ha sido cogido por filo. 4
Filósofo 6 suelta ten. izq. 7
Filósofo 6 suelta ten. der. 5
Filosofo 6 comienza a pensar
Filósofo 4 solicita ten. der.3
Ten. 7 ha sido liberado por filo. 6
Filósofo 0 suelta ten. izq. 1
Filósofo 0 suelta ten. der. 9
Filosofo 0 comienza a pensar
Filósofo 2 comienza a comer
Ten. 1 ha sido liberado por filo. 0
Ten. 1 ha sido cogido por filo. 2
Ten. 9 ha sido liberado por filo. 0
Ten. 9 ha sido cogido por filo. 8
Filósofo 8 solicita ten. der.7
Filósofo 8 comienza a comer
Ten. 7 ha sido cogido por filo. 8
Filósofo 6 solicita ten. izq.7
Filósofo 0 solicita ten. der.9
Ten. 3 ha sido liberado por filo. 2
Ten. 3 ha sido cogido por filo. 4

Problema de los filósofos con un camarero.

Para resolver el problema de sentarse y levantarse, se ha utilizado lo siguiente:

a) Para solicitar sentarse
```
MPI_Ssend(NULL, 0, MPI_INT, id_camarero, etiq_sentarse, MPI_COMM_WORLD);
   Para sentarse
MPI_Recv(NULL, 0, MPI_INT, id_camarero, etiq_sentarse, MPI_COMM_WORLD, &estado);
```

b) Para solicitar levantarse
```
MPI_Ssend(NULL, 0, MPI_INT, id_camarero, etiq_levantarse, MPI_COMM_WORLD);
```

Como se ha podido observar, se han creado dos etiquetas más, una para los mensajes de la categoria “sentarse” y otra pa los de categoria “levantarse”.

Finalmente, para la función camarero se ha hecho lo siguiente:

```
int filo_sentados = 0;
```

```
        MPI_Status estado ;       // metadatos de las dos recepciones
        int etiq_valida;
        int id_filosofo;

while(true){
                if(filo_sentados < 4){
                        etiq_valida = MPI_ANY_TAG;
                }       else    {
                        etiq_valida = etiq_levantarse;
                }

                MPI_Recv(NULL, 0, MPI_INT, MPI_ANY_SOURCE, etiq_valida, MPI_COMM_WORLD, &estado);

                id_filosofo = estado.MPI_SOURCE;

                if(estado.MPI_TAG == etiq_levantarse){
                        filo_sentados--;
                        cout << "El camarero levanta al filosofo " << id_filosofo << " y hay " << filo_sentados << "
filosofos sentados" << endl;
                }
                else if(estado.MPI_TAG == etiq_sentarse){
                        filo_sentados++;
                        MPI_Ssend(NULL, 0, MPI_INT, id_filosofo, etiq_sentarse, MPI_COMM_WORLD);
                        cout << "El camarero da el turno al filosofo " << id_filosofo << " y hay " << filo_sentados <<
" filosofos sentados" << endl;
                }
        }
```

Finalmente, el main se ha hecho de la siguiente manera, si es par es un filósofo, si es impar un tenedor:

```
if(id_propio == id_camarero)
                funcion_camarero();
                else{
                  // ejecutar la función correspondiente a 'id_propio'
                  if ( id_propio % 2 == 0 )          // si es par
                     funcion_filosofos( id_propio ); //   es un filósofo
                  else                              // si es impar
                     funcion_tenedores( id_propio ); //   es un tenedor
                  }
  }
```

Para id_camarero = num_procesos-1;
num_procesos = 2*num_filosofos+1,

<u>Listado parcial de la salida del programa sería:</u>

```
El filosofo 0solicita sentarse
El filosofo 8solicita sentarse
El filosofo 2solicita sentarse
El filosofo 0 puede sentarse
Filósofo 0 solicita ten. izq.1
El filosofo 6solicita sentarse
El camarero da el turno al filosofo 0 y hay 1 filosofos sentados
Filósofo 0 solicita ten. der.10
Ten. 1 ha sido cogido por filo. 0
El filosofo 8 puede sentarse
Filósofo 8 solicita ten. izq.9
```

Filósofo 8 solicita ten. der.7
Ten. 3 ha sido cogido por filo. 2
El filosofo 4solicita sentarse
El camarero da el turno al filosofo 2 y hay 2 filosofos sentados
El camarero da el turno al filosofo 6 y hay 3 filosofos sentados
El camarero da el turno al filosofo 8 y hay 4 filosofos sentados
Ten. 7 ha sido cogido por filo. 6
El filosofo 2 puede sentarse
Filósofo 2 solicita ten. izq.3
Filósofo 2 solicita ten. der.1
El filosofo 6 puede sentarse
Filósofo 6 solicita ten. izq.7
Filósofo 6 solicita ten. der.5
Filósofo 6 comienza a comer
Ten. 5 ha sido cogido por filo. 6
Ten. 9 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
El filosofo 4 puede sentarse
Filósofo 4 solicita ten. izq.5
El camarero levanta al filosofo 6 y hay 3 filosofos sentados
El camarero da el turno al filosofo 4 y hay 4 filosofos sentados
Ten. 7 ha sido liberado por filo. 6
Ten. 7 ha sido cogido por filo. 8
Ten. 5 ha sido liberado por filo. 6
Ten. 5 ha sido cogido por filo. 4
Filósofo 0 comienza a comer
Filósofo 4 solicita ten. der.3
Ten. 9 ha sido liberado por filo. 8
Filósofo 8 suelta ten. izq. 9
Filósofo 8 suelta ten. der. 7
Filosofo 8 comienza a pensar
El camarero levanta al filosofo 8 y hay 3 filosofos sentados
Ten. 7 ha sido liberado por filo. 8
Ten. 1 ha sido liberado por filo. 0
Ten. 1 ha sido cogido por filo. 2
Filósofo 0 suelta ten. izq. 1
Filósofo 0 suelta ten. der. 10
Filosofo 0 comienza a pensar
El camarero levanta al filosofo 0 y hay 2 filosofos sentados
Filósofo 2 comienza a comer
Ten. 7 ha sido cogido por filo. 6
El filosofo 6solicita sentarse
El filosofo 6 puede sentarse
Filósofo 6 solicita ten. izq.7
Filósofo 6 solicita ten. der.5
El camarero da el turno al filosofo 6 y hay 3 filosofos sentados