EXAMEN PRÁCTICA 3 – SCD – (12/1/15) – CARLOS UREÑA
PASO DE MENSAJES MPI (MODELO B)

El tiempo de realización del examen es de 60 minutos. La primera parte se responderá contestando en el propio folio del examen. La segunda parte se realiza partiendo de la solución proporcionada por el alumno a la práctica. Se debe subir el código generado a Tutor antes del plazo que el profesor indique.

1. Corrección de errores (6 puntos)

Accede a Tutor y descarga el archivo que el profesor te ha proporcionado (A3A-3b-ebs.cpp). El código que contiene intenta plantear una solución al problema de la cena de los filósofos con camarero. Sin embargo, hay diversos errores en el código proporcionado que deberás solucionar.

1a.1. Intenta compilar el código proporcionado. Comprueba que hay diversos errores básicos de sintaxis que impiden que el código genere un ejecutable. Encuentra dichos errores y arréglalos hasta que el código compile correctamente. Indica en el folio del examen el número de línea y cómo debería quedar ésta una vez solventado el problema.

1a.2. Ejecuta el programa y observa la salida que genera. El algoritmo presenta un error en su diseño. Explica brevemente en el folio del examen por qué la salida proporcionada no es correcta.

1a.3. Observa el código del programa ejecutado. Encuentra y corrige el error en el diseño del algoritmo que impide que el programa muestre una salida coherente. Señala en el folio del examen en qué línea está el problema y cómo debería quedar ésta para solucionar el error.

1a.4. Se desea modificar el algoritmo dado del problema de los filósofos con camarero para dar servicio a 7 filósofos y 7 tenedores en lugar de a 5. También se desea cambiar el identificador de los procesos para que los filósofos sean los procesos con rank 0, 1, 2, …, 6 y los tenedores sean los procesos 7, 8, …, 13. Describe brevemente qué cambios debes realizar en el código proporcionado y en qué líneas.

2. Problema de múltiples Productores-Consumidores con Paso de Mensajes (4 puntos)

Partiendo del algoritmo desarrollado durante la Práctica 3 para resolver el problema de múltiples Productores-Consumidores usando paso de mensajes MPI, se pide realizar los siguientes cambios. El valor que los procesos productores envían al buffer deberá ser un valor aleatorio entre 0 y 9 generado mediante la función rand() en lugar de uno secuencial como hasta ahora. Así mismo, el productor deberá enviar un mensaje al buffer pidiendo permiso para enviar el valor, y el buffer deberá responder al productor autorizando a enviar el mensaje antes de hacerlo (al igual que ya hace el consumidor).

Realiza los cambios necesarios para que, en el caso de que en el buffer se almacenen dos valores iguales consecutivos (independientemente del productor que los envíe), el buffer finalice. Antes de finalizar deberá avisar a todos los productores y consumidores para que estos finalicen enviando un valor especial en los mensajes de autorización. (Por ejemplo, un valor negativo). Sólo cuando todos los procesos hayan finalizado el buffer terminará. Implementa esta condición de forma que cuando cualquier proceso vaya a finalizar, imprima un mensaje en pantalla informando de ello.

```
#include <iostream>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <mpi.h>

#define NUM_FILOSOFOS   5
#define NUM_PROCESOS   (2*(NUM_FILOSOFOS)+1)
#define RANK_CAMARERO  ((NUM_PROCESOS)-1)

#define TAG_SENTARSE   0
#define TAG_LEVANTARSE 1


using namespace std ;

void Filosofo( int id );
void Tenedor ( int id );
void Camarero() ;

// ---------------------------------------------------------------------

void retraso_aleatorio()
{
   static bool primera = true ;

   // inicializar generador de números aleatorios (la primera vez)
   if ( primera )
   {  srand(time(NULL)) ;
      primera = false ;
   }
   // retraso aleatorio, de entre una y dos décimas de segundo
   usleep( 1000L * (100L + rand() % 100L ) );
}
// ---------------------------------------------------------------------

int main(int argc,char** argv )
{
   int rank,size;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );

   if ( size != NUM_PROCESOS )
   {
      if( rank == 0 )
         cout << "El numero de procesos debe ser " << NUM_PROCESOS << endl
<< flush;
      MPI_Finalize( );
      return 0;
   }
```

```
   if ( rank == RANK_CAMARERO )
      Camarero() ;      // el 10 es el camarero
   else if ( (rank%2) == 0 )
      Filosofo( rank ); // los pares (0,2,4,6,8) son Filosofos
   else
      Tenedor( rank );  // los impares (1,3,5,7,9) son Tenedores

   MPI_Finalize( );
   return 0;
}
// ----------------------------------------------------------------------

void Filosofo( int id )
{
   const int
      primero = (id+1)                % (NUM_PROCESOS-1) ,
      segundo = (id+(NUM_PROCESOS-2)) % (NUM_PROCESOS-1) ;

   while ( true )
   {
      // solicita sentarse
      cout << "Filosofo " << id << " solicita sentarse." << endl << flush;
      MPI_Ssend( &id, 1, MPI_INT, RANK_CAMARERO, TAG_SENTARSE,
MPI_COMM_WORLD );

      // solicita y coge primer tenedor
      cout << "Filosofo " << id << " solicita primer tenedor: " << primero
<< endl << flush;
      MPI_Ssend( &id, 1, MPI_INT, primero, primero, MPI_COMM_WORLD );
      cout << "Filosofo " << id << " coge primer tenedor: " << primero <<
endl << flush;

      // solicita coge segundo tenedor
      cout << "Filosofo " << id << " coge segundo tenedor: " << segundo <<
endl << flush;
      MPI_Ssend( &id, 1, MPI_INT,segundo,MPI_COMM_WORLD);
      cout << "Filosofo " << id << " coge segundo tenedor: " << segundo <<
endl << flush;

      // come
      cout << "Filosofo " << id << " COMIENDO" << endl << flush;
      retraso_aleatorio() ;

      // suelta el segundo tenedor
      cout << "Filosofo " << id << " suelta tenedor: " << segundo << endl
<< flush;
      MPI_Ssend( &id, 1, MPI_INT, primero, primero, MPI_COMM_WORLD );

      // suelta el primer tenedor
      cout << "Filosofo " << id << " suelta tenedor: " << primero << endl
<< flush;
      MPI_Ssend( &id, 1, MPI_INT, segundo, segundo, MPI_COMM_WORLD );
```

```
      // solicita levantarse
      cout << "Filosofo " << id << " solicita sentarse." << endl << flush;
      MPI_Ssend ( &id, 1, MPI_INT, RANK_CAMARERO, TAG_SENTARSE,
MPI_COMM_WORLD );

      // piensa
      cout << "Filosofo " << id << " PENSANDO" << endl << flush ;
      retraso_aleatorio() ;
   }
}
//----------------------------------------------------------------------

void Tenedor( int id )
{
   int valor ;
   MPI_Status status;

   while ( true )
   {
      // espera un mensaje desde cualquier filosofo vecino
      cout <<"Tenedor " << id << " espera petición." << endl << flush;
      MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, id, MPI_COMM_WORLD,
&status );
      cout << "Tenedor " << id << " recibe petición de " <<
status.MPI_SOURCE << endl << flush ;

      // espera a que el filosofo suelte el tenedor
      MPI_Recv( 1, &valor,  MPI_INT, status.MPI_SOURCE, id,
MPI_COMM_WORLD, &status );
      cout<<"Tenedor " << id << " recibe liberación de " <<
status.MPI_SOURCE << endl << flush;
  }
}

//----------------------------------------------------------------------

void Camarero()
{
   int valor, num_sentados = 0, tag_aceptada ;
   MPI_Status status;

   while ( true )
   {
      if ( num_sentados < NUM_FILOSOFOS-1 )
         tag_aceptada = MPI_ANY_TAG ;
      else
         tag_aceptada = TAG_LEVANTARSE ;

      // espera un mensaje desde cualquier filosofo con etiqueta aceptable
      cout << "Camarero espera petición de sentarse/levantarse (sentados
== " << num_sentados << ")" << endl << flush;
```

```
      MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, tag_aceptada,
MPI_COMM_WORLD, &status);
      cout << "Camarero recibe petición de filosofo " << status.MPI_SOURCE
<< endl << flush ;

      if ( status.MPI_TAG == TAG_LEVANTARSE )
         num_sentados -- ;
      else
         num_sentados ++ ;
  }
}
```

SOLUCIÓN (ERRORES COMPILACIÓN)

1. Línea 84. Falta un argumento en la función MPI_Ssend. No está puesto el tag del mensaje (debería ser "segundo").

   MPI_Ssend(&id, 1, MPI_INT,segundo,segundo,MPI_COMM_WORLD);

2. Línea 123. La función MPI_Recv está recibiendo los dos primeros argumentos al revés. Primero debe enviarse el valor y después el tamaño del envío.

   MPI_Recv(&valor, 1, MPI_INT, status.MPI_SOURCE, id, MPI_COMM_WORLD, &status);

SOLUCIÓN (ERROR DISEÑO)

El error en la salida está en que ningún filósofo puede levantarse de la mesa cuando termina de comer. Debe haber algún fallo al enviar el mensaje al camarero para solicitar levantarse.

Línea 101. En el paso del mensaje al camarero para solicitar levantarse, el tag del mensaje es el de Sentarse (TAG_SENTARSE) en lugar de el de levantarse (TAG_LEVANTARSE).

MPI_Ssend (&id, 1, MPI_INT, RANK_CAMARERO, TAG_LEVANTARSE,MPI_COMM_WORLD);