Yang Chen

Portafolio Productor-Consumidor

1. **Cambios realizados y el próposito de dichos cambios:**
   Asignar el ID de buffer a 5, de manera que los procesos con un ID<5 son productores y los ID>5 son cnsumidores.
   Ahora el buffer recibe los datos del MPI_ANY_SOURCE, para diferenciar si es de productor o consumidor ponemos el tag productor/consumidor correspondiente.

2. **Codigo fuente:**

```
#include <mpi.h>
#include <iostream>
#include <math.h>
#include <time.h>          // incluye "time"
#include <unistd.h>        // incluye "usleep"
#include <stdlib.h>        // incluye "rand" y "srand"


#define Productor       0
#define Buffer          5
#define Consumidor      2
#define ITERS           20
#define TAM             5

using namespace std;

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

void productor(int rank)
{
    int value ;

    for ( unsigned int i=0; i < ITERS ; i++ )
    {
        value = i ;
        cout << "Productor " << rank << " produce valor " << value << endl <<
flush ;

        // espera bloqueado durante un intervalo de tiempo aleatorio
        // (entre una decima de segundo y un segundo)
        usleep( 1000U * (100U+(rand()%900U)) );

        // enviar 'value'
        MPI_Ssend( &value, 1, MPI_INT, Buffer, Productor, MPI_COMM_WORLD );
```

```
    }
}
// ---------------------------------------------------------------------

void buffer()
{
    int             value[TAM] ,
                    peticion ,
                    pos   =   0,
                    rama ;
    MPI_Status status ;

    for( unsigned int i=0 ; i < ITERS*2 ; i++ )
    {
        if ( pos==0 )          // el consumidor no puede consumir
            rama = 0 ;
        else if (pos==TAM) // el productor no puede producir
            rama = 1 ;
        else                        // ambas guardas son ciertas
        {
            // leer 'status' del siguiente mensaje (esperando si no hay)
            MPI_Probe(  MPI_ANY_SOURCE,  MPI_ANY_TAG,  MPI_COMM_WORLD,
&status );

            // calcular la rama en funcion del origen del mensaje
            if ( status.MPI_SOURCE == Productor )
                rama = 0 ;
            else
                rama = 1 ;
        }
        switch(rama)
        {
            case 0:
                MPI_Recv(  &value[pos],  1,  MPI_INT,MPI_ANY_SOURCE,  Productor,
MPI_COMM_WORLD, &status );
                cout << "Buffer recibe " << value[pos] << " de Productor " <<
status.MPI_SOURCE << endl << flush;
                pos++;
                break;
            case 1:
                MPI_Recv(  &peticion,  1,  MPI_INT,MPI_ANY_SOURCE,  Consumidor,
MPI_COMM_WORLD, &status );
                MPI_Ssend(  &value[pos-1],  1,  MPI_INT,  status.MPI_SOURCE,  0,
MPI_COMM_WORLD);
```

```
                    cout << "Buffer envia " << value[pos-1] << " a Consumidor "
<<status.MPI_SOURCE << endl << flush;
                    pos--;
                    break;
                }
            }
        }

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

        void consumidor(int rank)
        {
            int             value,
                             peticion = 1 ;
            float           raiz ;
            MPI_Status   status ;

            for (unsigned int i=0;i<ITERS;i++)
            {
                MPI_Ssend( &peticion, 1, MPI_INT, Buffer, Consumidor, MPI_COMM_WORLD );
                MPI_Recv ( &value, 1,       MPI_INT, Buffer, 0, MPI_COMM_WORLD,&status );
                cout << "Consumidor " <<rank << " recibe valor " << value << " de Buffer "
<< endl << flush ;

                // espera bloqueado durante un intervalo de tiempo aleatorio
                // (entre una decima de segundo y un segundo)
                usleep( 1000U * (100U+(rand()%900U)) );

                raiz = sqrt(value) ;
            }
        }
        // ---------------------------------------------------------------------

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

            // inicializar MPI, leer identif. de proceso y numero de procesos
            MPI_Init( &argc, &argv );
            MPI_Comm_rank( MPI_COMM_WORLD, &rank );
            MPI_Comm_size( MPI_COMM_WORLD, &size );

            // inicializa la semilla aleatoria:
            srand ( time(NULL) );
```

```
    // comprobar el numero de procesos con el que el programa
    // ha sido puesto en marcha (debe ser 10)
    if ( size != 10 )
    {
        cout<< "El numero de procesos debe ser 10 "<<endl;
        return 0;
    }

    // verificar el identificador de proceso (rank), y ejecutar la
    // operaci 贸 n apropiada a dicho identificador
    if ( rank < Buffer )
        productor(rank);
    else if ( rank == Buffer )
        buffer();
    else
        consumidor(rank);

    // al terminar el proceso, finalizar MPI
    MPI_Finalize( );
    return 0;
}
```

Salida:

```
--------------------------------------------------------------------------
yang@yang-VirtualBox:~/Escritorio/scdprac3$ mpirun -np 10 prod-cons
Productor 0 produce valor 0
Productor 4 produce valor 0
Productor 1 produce valor 0
Productor 3 produce valor 0
Productor 2 produce valor 0
Buffer recibe 0 de Productor 0
Productor 0 produce valor 1
Consumidor 6 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 6
Buffer recibe 0 de Productor 4
Productor 4 produce valor 1
Consumidor 7 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 7
Buffer recibe 0 de Productor 1
Productor 1 produce valor 1
Consumidor 8 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 8
Buffer recibe 0 de Productor 2
Consumidor 9 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 9
Buffer recibe 0 de Productor 3
Productor 3 produce valor 1
Productor 2 produce valor 1
Buffer recibe 1 de Productor 0
Productor 0 produce valor 2
Consumidor 6 recibe valor 1 de Buffer
Buffer envía 1 a Consumidor 6
Consumidor 7 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 7
```