Productor / Consumidor

1. Cambios realizados sobre el programa de partida:

- He modificado las **constantes**:
  - NUM_PRODUCTORES, indica el número de productores totales (5).
  - NUM_CONSUMIDORES, indica el número de consumidores totales (4).
  - BUFFER, es el identificador que le doy al proceso buffer (5).
  - PRODUCTOR, es el identificador que le doy a todos los procesos productores (1).
  - CONSUMIDOR, es el identificador que le doy a todos los procesos consumidores (2).

- Proceso **productor()**:
  - Añado MPI_Comm_rank(MPI_COMM_WORLD, &rank) para saber el número de proceso cuando escribo el mensaje.

- Proceso **consumidor()**:
  - Añado MPI_Comm_rank(MPI_COMM_WORLD, &rank) para saber el número de proceso cuando escribo el mensaje.

- **main()**:
  - if (size != NUM_PRODUCTORES+NUM_CONSUMIDORES+1) {
    cout << "El numero de procesos debe ser "
    << NUM_PRODUCTORES+NUM_CONSUMIDORES+1 << endl;
    return -1;
    }

    Modifico la comparación para ajustarlo al número de parámetros que debe recibir

  - if (rank<NUM_PRODUCTORES)
    productor();
    else if (rank==NUM_PRODUCTORES)
    buffer();
    else
    consumidor();

    Se ejecutan tantos procesos productores como se dice en la constante, el 5 será el buffer y los restantes los consumidores.

  - MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

    if (status.MPI_TAG == PRODUCTOR)
    rama = 0;
    else
    rama = 1;

    Dentro del if para saber si nos llega un productor o un consumidor, lo único que cambia es el MPI_Probe, para recoger el status. Como se quiere recibir todo, se recibe cualquier fuente y cualquier etiqueta. Después de recoger el status, se comprueba quién ha sido.

- MPI_Recv(&value[pos], 1, MPI_INT, MPI_ANY_SOURCE, PRODUCTOR, MPI_COMM_WORLD, &status);

  Dentro de la rama 0 (la rama del productor), hay que recibir el valor que ha producido cualquier productor. Para eso, se recibe la información de cualquier fuente pero con etiqueta PRODUCTOR.

- MPI_Recv(&peticion, 1, MPI_INT, MPI_ANY_SOURCE, CONSUMIDOR, MPI_COMM_WORLD, &status);

  Dentro de la rama 1 (la rama del consumidor), se recibe la petición de que el consumidor está preparado para consumir. Para eso, se recibe la petición de cualquier fuente pero con etiqueta CONSUMIDOR.

- MPI_Ssend(&value[pos - 1], 1, MPI_INT, status.MPI_SOURCE, CONSUMIDOR, MPI_COMM_WORLD);

  Cuando ya se ha recibido la petición del consumidor, ahora hay que devolvérsela. Como se ha quedado en status la información del consumidor, se utiliza status.MPI_SOURCE para enviárselo exclusivamente a él y además se utiliza la etiqueta CONSUMIDOR.

2. Código fuente de la solución aportada:

```
#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 NUM_PRODUCTORES     5
#define NUM_CONSUMIDORES    4
#define BUFFER       5
#define PRODUCTOR    1  // Lo identifica en el paso de mensajes
#define CONSUMIDOR   2  // Lo identifica en el paso de mensajes
#define ITERS       20
#define TAM          5

using namespace std;

//////////////////////////////////////////////////////////////////////////
// Función que ejecuta cada productor
//
void productor(){

   // Rank se utiliza para guardar el número del proceso
   int value, rank;

   // Se guarda en rank el número del proceso del productor
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
   for (unsigned int i=0; i<ITERS/NUM_PRODUCTORES; i++) {

      value = i;
      cout << "Productor " << rank << " produce valor "
           << value << endl << flush;

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

      // Enviar 'value'
      MPI_Ssend(&value, 1, MPI_INT, BUFFER, PRODUCTOR, MPI_COMM_WORLD);

   }

}

//////////////////////////////////////////////////////////////////////////
// Función que ejecuta el proceso que regula los datos
//
void buffer() {

   int value[TAM], peticion, pos = 0, rama;
   MPI_Status status;

   for (unsigned int i = 0; i < ITERS * 2; i++) {

      // El consumidor no puede consumir
      if (pos == 0)
         rama = 0;
      // El productor no puede producir
      else if (pos == TAM)
         rama = 1;
      // Ambas guardas son ciertas
      else{

         // 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 función del origen del mensaje
         if (status.MPI_TAG == PRODUCTOR)
            rama = 0;
         else
            rama = 1;

      }
```

```
      switch (rama){

         // Productor
         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;

         // Consumidor
         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, CONSUMIDOR,
MPI_COMM_WORLD);
            cout << "Buffer envía " << value[pos - 1] << " a Consumidor "
                  << status.MPI_SOURCE << endl << flush;
            pos--;
            break;

      }

   }

}

//////////////////////////////////////////////////////////////////////////////
// Proceso que ejecuta el consumidor
//
void consumidor() {

   int value, peticion = 1, rank;
   float raiz;
   MPI_Status status;

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   for (unsigned int i = 0; i < ITERS/NUM_CONSUMIDORES; i++) {

      MPI_Ssend(&peticion, 1, MPI_INT, BUFFER, CONSUMIDOR, MPI_COMM_WORLD);
      MPI_Recv(&value, 1, MPI_INT, BUFFER, CONSUMIDOR, MPI_COMM_WORLD,
&status);
      cout << "Consumidor " << rank << " recibe valor " << value
            << " de Buffer " << endl << flush;
```

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

        raiz = sqrt(value);

    }

}

//////////////////////////////////////////////////////////////////////////
// main
//
int main(int argc, char *argv[]) {

    int rank, size;

    // Inicializar MPI, leer identif. de proceso y número 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 número de procesos con el que el programa
    // ha sido puesto en marcha (debe ser 10)
    if (size != NUM_PRODUCTORES+NUM_CONSUMIDORES+1) {
        cout << "El numero de procesos debe ser "
             << NUM_PRODUCTORES+NUM_CONSUMIDORES+1 << endl;
        return -1;
    }

    // Verificar el identificador de proceso (rank), y ejecutar la
    // operación apropiada a dicho identificador
    if (rank<NUM_PRODUCTORES)
        productor();
    // El buffer se ejecuta después de que los productores se hayan iniciado
    else if (rank==NUM_PRODUCTORES)
        buffer();
    else
        consumidor();

    // Al terminar el proceso, finalizar MPI
    MPI_Finalize();

    return 0;

}
```

3. Listado de la salida del programa:

```
fran@fran-Lenovo-Ubuntu:~/Escritorio/Universidad/SCD/Mis prácticas/Práctica 3$ mpicxx -o prodcons prodcons.cpp
fran@fran-Lenovo-Ubuntu:~/Escritorio/Universidad/SCD/Mis prácticas/Práctica 3$ mpirun -np 10 ./prodcons
Productor 3 produce valor 0
Productor 0 produce valor 0
Productor 1 produce valor 0
Productor 2 produce valor 0
Productor 4 produce valor 0
Productor 3 produce valor 1
Buffer recibe 0 de Productor 3
Consumidor 6 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 6
Productor 2 produce valor 1
Buffer recibe 0 de Productor 2
Consumidor 7 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 7
Buffer recibe 0 de Productor 0
Productor 0 produce valor 1
Consumidor 8 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 8
Buffer recibe 0 de Productor 1
Productor 1 produce valor 1
Consumidor 9 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 9
Buffer recibe 0 de Productor 4
Productor 4 produce valor 1
Productor 3 produce valor 2
Buffer recibe 1 de Productor 3
Productor 2 produce valor 2
Buffer recibe 1 de Productor 2
Consumidor 6 recibe valor 1 de Buffer
Buffer envía 1 a Consumidor 6
Productor 0 produce valor 2Buffer recibe
1 de Productor 0
Buffer recibe 1 de Productor 1
Productor 1 produce valor 2
Buffer recibe 1 de Productor 4
Buffer envía Consumidor 7 recibe valor 1 de Buffer
1 a Consumidor 7
Consumidor 8 recibe valor 1 de Buffer
Productor 4 produce valor 2
Buffer envía 1 a Consumidor 8
Consumidor 9 recibe valor 1 de Buffer
Buffer envía 1 a Consumidor 9
Buffer recibe 2 de Productor 3
Productor 3 produce valor 3
Consumidor 6 recibe valor 2 de Buffer
Buffer envía 2 a Consumidor 6
Productor 2 produce valor 3Buffer recibe
```

2 de Productor 2
Consumidor 7 recibe valor 2 de Buffer
Buffer envía 2 a Consumidor 7
Consumidor 8 recibe valor 1 de Buffer
Buffer envía 1 a Consumidor 8
Buffer recibe 2 de Productor 0
Buffer recibe 2 de Productor 1
Consumidor 9 recibe valor 2 de Buffer
Productor 1 produce valor 3
Productor 0 produce valor 3
Buffer envía 2 a Consumidor 9
Buffer recibe 2 de Productor 4
Productor 4 produce valor 3
Buffer envía 2 a Consumidor 6
Consumidor 6 recibe valor 2 de Buffer
Buffer envía 2 a Consumidor 7
Consumidor 7 recibe valor 2 de Buffer
Consumidor 8 recibe valor 0 de Buffer
Buffer envía 0 a Consumidor 8
Buffer recibe 3 de Productor 3
Consumidor 9 recibe valor 3 de Buffer
Buffer envía 3 a Consumidor 9
Buffer recibe 3 de Productor 2
Buffer recibe 3 de Productor 0
Buffer recibe 3 de Productor 1
Buffer recibe 3 de Productor 4
Buffer envía 3 a Consumidor 6
Consumidor 6 recibe valor 3 de Buffer
Buffer envía 3 a Consumidor 7
Consumidor 7 recibe valor 3 de Buffer
Consumidor 8 recibe valor 3 de Buffer
Buffer envía 3 a Consumidor 8
Consumidor 9 recibe valor 3 de Buffer
Buffer envía 3 a Consumidor 9