PablaO
www.wuolah.com/student/PablaO

prodcons2.pdf

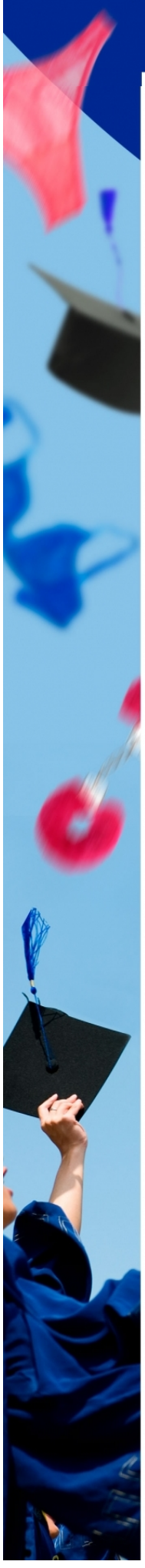
Practica 3-Ejercicios

2º Sistemas Concurrentes y Distribuidos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

```
#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    4
#define Buffer       5
#define Consumidor   6
#define ITERS       20
#define TAM          5

using namespace std;

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

void productor()
{
   int value ;

   for ( unsigned int i = 0; i < ITERS/5; i++ ) {
      value = i ;
      cout << "Productor 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 );
   }
}
// ---------------------------------------------------------------------

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

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

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

   for (unsigned int i=0;i<ITERS/4;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 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) ;
   }
}
// ----------------------------------------------------------------------

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 != 10 )
   {
      cout << "El numero de procesos debe ser 10 "<<endl;
      return 0;
   }
```

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

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