PablaO
www.wuolah.com/student/PablaO

prodcons.pdf

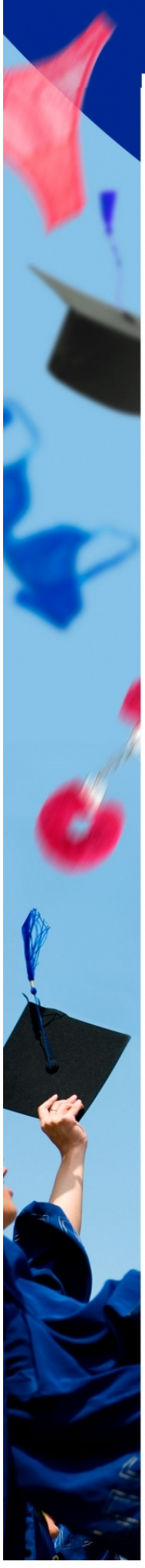
Practica 3-Ejercicios

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     0
#define Buffer        1
#define Consumidor    2
#define ITERS        20

using namespace std;

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

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

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

      // enviar valor
      MPI_Ssend( &i, 1, MPI_INT, Buffer, 0, MPI_COMM_WORLD );
   }
}
// ---------------------------------------------------------------------

void buffer()
{
   int         value ,
               peticion ;
   MPI_Status  status ;

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

      MPI_Ssend( &value,   1, MPI_INT, Consumidor, 0, MPI_COMM_WORLD);
      cout << "Buffer envía valor " << value << " a Consumidor " << endl <<
flush;
   }
}
// ---------------------------------------------------------------------

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

   for (unsigned int i=0;i<ITERS;i++)
   {
      MPI_Ssend(&peticion, 1, MPI_INT, Buffer, 0, MPI_COMM_WORLD);
```



```
      MPI_Recv(&value, 1,     MPI_INT, Buffer, 0, 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)) );

      // calcular raíz del valor recibido
      raiz = sqrt( value );
   }
}
// ---------------------------------------------------------------------

int main( int argc, char *argv[] )
{
   int rank ,  // identificador de proceso (comienza en 0)
       size ;  // numero de procesos

   // inicializar MPI y leer identificador 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 3)
   if ( size != 3 )
   {
      cout << "El numero de procesos debe ser 3, pero es " << size << "." <<
endl << flush ;
      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;
}
// ---------------------------------------------------------------------
```