PablaO
www.wuolah.com/student/PablaO

ejercicio_extra.pdf

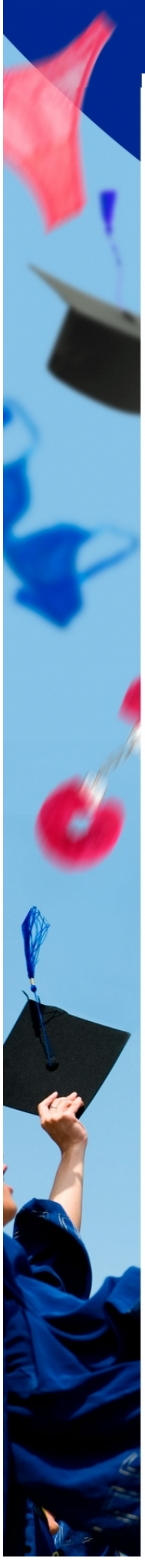
Practica 3-Ejercicios

2º Sistemas Concurrentes y Distribuidos

Grado en Ingeniería Informática

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

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

using namespace std;

void productor(int id) {
      int valor;

      while (true) {
            for (int i = 0; i < 10; i++) {
                  valor = rand() % 10;

                  cout << "Productor " << id << " produce el valor: " << valor
<< endl << flush;

                  usleep( 1000U * (100U+(rand()%900U)) );  //Espera aleatoria

                  MPI_Ssend(&valor, 1, MPI_INT, 4, 0, MPI_COMM_WORLD);
            }
      }
}

void consumidor(int id) {
      int v_recibido;
      MPI_Status status;

      while (true) {
            MPI_Recv(&v_recibido, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
&status);
            cout << "Consumidor " << id << " recibe el valor " << v_recibido <<
endl << flush;

            usleep( 1000U * (100U+(rand()%900U)) ); //Espera aleatoria
      }
}

void intermedio() {
      int prod;
      MPI_Status status;

      while (true) {
            MPI_Recv(&prod, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
&status);
            cout << "Itermediario recibe el valor: " << prod << " del productor
" << status.MPI_SOURCE << endl << flush;

            if (prod%2 == 0) {
                  cout << "Intermediario envía el valor: " << prod << " al
consumidor 0." << endl << flush;
                  MPI_Ssend(&prod, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
            }
            else {
                  cout << "Intermediario envía el valor: " << prod << " al
consumidor 1." << endl << flush;
                  MPI_Ssend(&prod, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
            }
      }

}

int main(int argc, char** argv) {
```



```
      int rank, size;

      srand(time(0));
   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );

      if (size != 5) {
            cout << "Error: el programa debe ejecutarse con 5 procesos." <<
endl;
            MPI_Finalize();
            return 0;
      }

      if (rank == 2 || rank == 3)        //Los procesos 2 y 3 son los
productores
            productor(rank);
      else if (rank == 4)                          //El proceso 4 es el
intermediario
            intermedio();
      else
            consumidor(rank);                      //El resto de procesos (0 y
1) son los consumidores

      MPI_Finalize();
      return 0;
}
```