Documento anónimo

filoex.pdf

Examen Práctica 3 MPI Completo

2º Sistemas Concurrentes y Distribuidos

Grado en Ingeniería Informática

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

```
// -----------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Práctica 3. Implementación de algoritmos distribuidos con MPI
//
// Archivo: filosofos-plantilla.cpp
// Implementación del problema de los filósofos (sin camarero).
// Plantilla para completar.
//
// Historial:
// Actualizado a C++11 en Septiembre de 2017
// -----------------------------------------------------------------------------


#include <mpi.h>
#include <thread> // this_thread::sleep_for
#include <random> // dispositivos, generadores y distribuciones aleatorias
#include <chrono> // duraciones (duration), unidades de tiempo
#include <iostream>

using namespace std;
using namespace std::this_thread ;
using namespace std::chrono ;

const int
   num_filosofos = 5 ,
   num_procesos  = 2*num_filosofos + 1;

const int id_camarero = 10;
const int etiq_sentarse = 1;
const int etiq_levantarse = 2;

//**********************************************************************
// plantilla de función para generar un entero aleatorio uniformemente
// distribuido entre dos valores enteros, ambos incluidos
// (ambos tienen que ser dos constantes, conocidas en tiempo de compilación)
//----------------------------------------------------------------------

template< int min, int max > int aleatorio()
{
  static default_random_engine generador( (random_device())() );
  static uniform_int_distribution<int> distribucion_uniforme( min, max ) ;
  return distribucion_uniforme( generador );
}

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

void funcion_filosofos( int id )
{
  int id_ten_izq = (id+1)              % num_procesos, //id. tenedor izq.
      id_ten_der = (id+num_procesos-1) % num_procesos; //id. tenedor der.

  int peticion;
```

```
  while ( true )
  {

    cout <<"Filósofo " <<id << " solicita sentarse." <<endl;
    MPI_Ssend(&peticion, 1, MPI_INT, id_camarero, etiq_sentarse,
MPI_COMM_WORLD);
    cout <<"Filósofo " <<id << " SE HA SENTADO." <<endl;

    cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
    // ... solicitar tenedor izquierdo (completar)
    MPI_Ssend(&peticion, 1, MPI_INT, id_ten_izq, 0, MPI_COMM_WORLD);

    cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
    // ... solicitar tenedor derecho (completar)
    MPI_Ssend(&peticion, 1, MPI_INT, id_ten_der, 0, MPI_COMM_WORLD);

    cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;
    sleep_for(seconds( aleatorio<5,10>() ) );

    cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id_ten_izq <<endl;
    // ... soltar el tenedor izquierdo (completar)
    MPI_Ssend(&peticion, 1, MPI_INT, id_ten_izq, 0, MPI_COMM_WORLD);

    cout<< "Filósofo " <<id <<" suelta ten. der. " <<id_ten_der <<endl;
    // ... soltar el tenedor derecho (completar)
    MPI_Ssend(&peticion, 1, MPI_INT, id_ten_der, 0, MPI_COMM_WORLD);

    cout <<"Filósofo " <<id << " solicita levantarse." <<endl;
    MPI_Ssend(&peticion, 1, MPI_INT, id_camarero, etiq_levantarse,
MPI_COMM_WORLD);
    cout <<"Filósofo " <<id << " SE HA LEVANTADO." <<endl;

    cout << "Filosofo " << id << " comienza a pensar" << endl;
    sleep_for(seconds( aleatorio<5,10>() ) );
    cout << "Filosofo " << id << " termina de pensar" << endl;
 }
}
// ---------------------------------------------------------------------

void funcion_tenedores( int id )
{
  int valor, id_filosofo ;  // valor recibido, identificador del filósofo
  MPI_Status estado ;       // metadatos de las dos recepciones

  while ( true )
  {
     // ..... recibir petición de cualquier filósofo (completar)
     MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &estado );
     // ..... guardar en 'id_filosofo' el id. del emisor (completar)
     id_filosofo = estado.MPI_SOURCE;
     cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;

     // ..... recibir liberación de filósofo 'id_filosofo' (completar)
     MPI_Recv( &valor, 1, MPI_INT, id_filosofo, 0, MPI_COMM_WORLD, &estado );
```

```
     cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id_filosofo <<endl ;
  }
}
// ---------------------------------------------------------------------

void camarero(){

    int etiq_aceptable;
    int sentados = 0;
    int valor;
    MPI_Status estado;

    while(true){

      if(sentados < 4 && estado.MPI_SOURCE != 6){

        etiq_aceptable = MPI_ANY_TAG;

      }else{

        etiq_aceptable = etiq_levantarse;

      }


      MPI_Recv(&valor, 1, MPI_INT, MPI_ANY_SOURCE, etiq_aceptable,
MPI_COMM_WORLD, &estado);

      switch(estado.MPI_TAG){

        case etiq_sentarse:
          sentados ++;
          cout << "HAY " << sentados << " FILOSOFOS SENTADOS" << endl;
          break;

        case etiq_levantarse:

        if(sentados == 4){

        cout << "ESTABAN 4 FILOSOFOS SENTADOS Y SE HA LEVANTADO EL FILOSOFO " <<
estado.MPI_SOURCE << endl;

        }

        if(estado.MPI_SOURCE == 6){

        estado.MPI_SOURCE = 100;

        }

          sentados --;
          break;

      }
```

```
    }


}

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

int main( int argc, char** argv )
{
   int id_propio, num_procesos_actual ;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &id_propio );
   MPI_Comm_size( MPI_COMM_WORLD, &num_procesos_actual );


   if ( num_procesos == num_procesos_actual )
   {
      // ejecutar la función correspondiente a 'id_propio'

      if(id_propio == 10){ //si es el 10 es el camarero.

        camarero();

      }else if(id_propio % 2 == 0 ){ // si es par

        funcion_filosofos( id_propio ); //   es un filósofo

      }else{ // si es impar

        funcion_tenedores( id_propio ); //   es un tenedor

      }

   }
   else
   {
      if ( id_propio == 0 ) // solo el primero escribe error, indep. del rol
      { cout << "el número de procesos esperados es:    " << num_procesos <<
endl
             << "el número de procesos en ejecución es: " << num_procesos_actual
<< endl
             << "(programa abortado)" << endl ;
      }
   }

   MPI_Finalize( );
   return 0;
}

// --------------------------------------------------------------------
```