Resolucion-SCD.pdf

pr0gramming_312823

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

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

EXAMEN SCD P3 2020 - 2021

1. Construir un programa MPI, que se deberá llamar prod_cons_ex.cpp, con operaciones de paso de mensaje síncronas siguiendo el esquema del productor-consumidor, en el que tenemos cuatro procesos consumidores (identificadores del 0 al 3) y seis procesos productores (identificadores del 4 al 9) que producirán 15 elementos cada uno. Ambos tipos de procesos se comunican mediante un proceso intermedio "buffer" con identificador 10. Dicho proceso intermedio gestiona un vector de tamaño 10. Otras consideraciones:

   - La gestión del vector se debe hacer en modo LIFO, es decir, se debe consumir el último elemento producido.

   - El proceso buffer trata de equilibrar la atención a productores y consumidores independientemente de la ocupación del vector. Por tanto, cada vez que atiende a cuatro productores de forma consecutiva (sin atender entre medias a ningún consumidor), no atiende más peticiones de los productores hasta que no haya atendido a un proceso consumidor.

2. Un food-truck está aparcado cerca de la ETSIIT y sólo vende bocatas de panceta con morcilla untada. Es frecuentado por 12 alumnos de SCD poco cuidadosos con su alimentación (procesos 0 al 11). Los alumnos con identificador par compran un bocata en cada iteración y los alumnos con identificador impar son más "tragones" y compran dos bocatas en cada iteración. En cada iteración, los alumnos mandan un mensaje al proceso dependiente (rank 12) con etiquetas distintas según el número de bocatas a comprar. Si hay menos de 3 bocatas en el camión, el proceso dependiente solo acepta peticiones de un bocata. Cuando no quedan bocatas, el proceso dependiente manda un mensaje al proceso cocinero (rank 13), que se encarga de preparar en el camión 20 nuevos bocatas. Suponer que el camión tiene inicialmente 20 bocatas, considerar todos los procesos como bucles infinitos e implementar dicho programa en MPI con operaciones de paso de mensaje síncronas y mensajes para seguir la traza del programa (el archivo se deberá llamar "ejerciciop2.cpp").

```
Proceso_alumno
   {retardo aleatorio ("camino hacia el camion")}
   s_send(dependiente,var) {peticion_bocata/s}
   receive(dependiente, var)  {bocata/s servido}
   {retardo aleatorio ("comer mucha grasa")}
```

```
Proceso_cocinero
    receive(dependiente,var) {preparado para preparar bocatas}
    {retardo aleatorio (preparar bocatas en en camión)}
    ssend(dependiente, var) {bocatas preparados}
```

```
Proceso_dependiente
   Si quedan en la tienda 3 o más bocatas
        recibir un mensajes de cualquier proceso alumno
   En caso contrario, si queda al menos una bocata
        recibir un mensaje de alumnos que sólo piden un bocata
   En caso contrario, (no hay bocatas en el camión)
         s_send(cocinero,var)
         recibir un mensaje del cocinero
   emisor:= proceso emisor del mensaje recibido anteriormente
   Si  emisor es un alumno
        actualizar estado {menos bocatas en la tienda}
        s_send(emisor,var) {bocata servido}
   En caso contrario (se ha recibido un mensaje del cocinero)
        actualizar estado {bocatas disponibles}
```

EXAMEN SCD P3 2020 - 2021

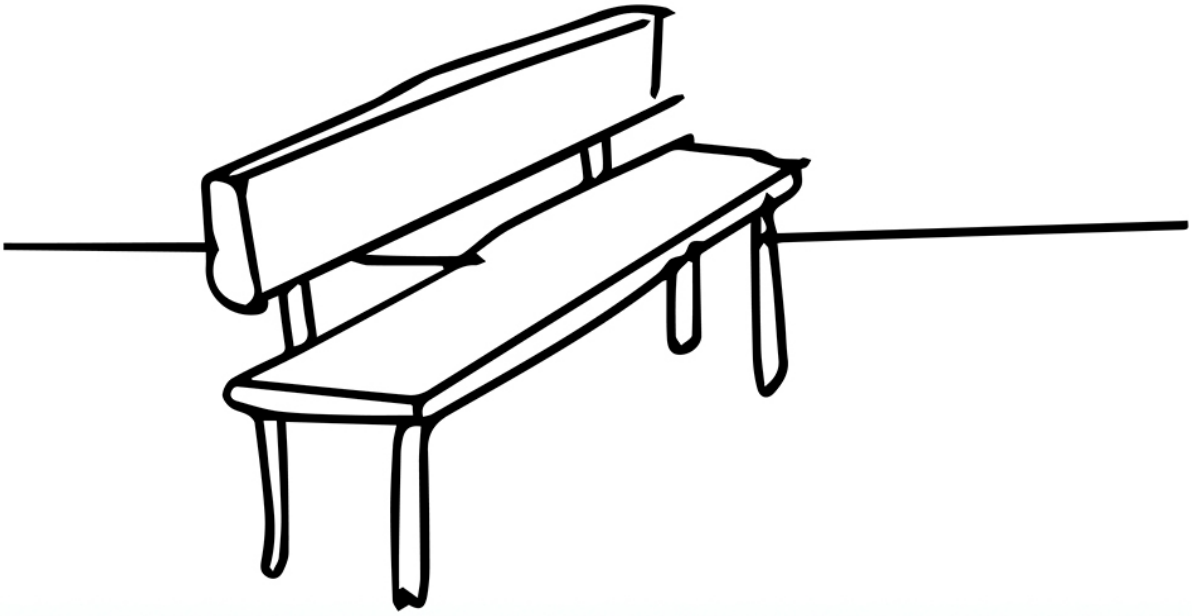
Resoluciones

El primero no lo he podido resolver del todo.

2.

```
//  Compilar:  mpicxx -std=c++11 ejerciciop2.cpp -o ejerciciop2
//  Ejecutar:  mpirun -np 14 ./ejerciciop2
#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_productores       = 1, //   Número de dependientes
    num_consumidores      = 12, //  Número de alumnos
    id_productor          = 13, //  ID del cocinero
    id_dependiente        = 12, //  ID del dependiente
    etiq_tragones         = 1, // etiqueta para alumnos tragones
    etiq_delgados         = 2,  // etiqueta para alumnos no tragones
    etiq_cocinero         = 3,  // etiqueta para el cocinero
    num_procesos_esperado = num_productores + num_consumidores + 1; // número de procesos que
espera el programa (14)


//**********************************************************************
// 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_cocinero()
{
    int peticion,
        valor_prod,
        valor_rec = 1;
    MPI_Status estado;

    while(true){
        MPI_Recv ( &valor_rec, 1, MPI_INT, id_dependiente, 0, MPI_COMM_WORLD, &estado); //  Cu
ando recibo el mensaje del dependiente
        cout << "Se empiezan a preparar 10 bocatas." << endl << flush;
        sleep_for(seconds(aleatorio<1,3>()));
        cout << "Se han preparado 10 bocatas." << endl << flush;
        cout << "Cocinero va a enviar 10 bocatas." << endl << flush;
        MPI_Ssend( &valor_prod, 1, MPI_INT, id_dependiente, etiq_cocinero, MPI_COMM_WORLD); //
 Le mando los 20 bocatas
    }
}

void funcion_alumno(int id_propio){
   int         peticion,
               valor_rec = 1 ;
   MPI_Status  estado ;

    while(true){
        cout << "\t\tEl alumno " << id_propio << " va hacia el camion" << endl << flush;
        sleep_for(seconds(aleatorio<1,2>()));
        cout << "\t\tEl alumno " << id_propio << " ha llegado al camion" << endl << flush;
        cout << "\t\tEl alumno " << id_propio << " va a pedir un bocata" << endl << flush;
```

este es el único banco del que te tienes que preocupar este mes



si no entiendes nada...

el 1 de noviembre lo entenderás

WUOLAH

```
        //  1. pide bocadilo

        if(id_propio % 2 == 0){     // Si es tragón
            MPI_Ssend( &peticion, 1, MPI_INT, id_dependiente, etiq_tragones, MPI_COMM_WORLD);
        }
        else{   // Si no es tragón
            MPI_Ssend ( &peticion, 1, MPI_INT, id_dependiente, etiq_delgados, MPI_COMM_WORLD);
        }

        //  2. recibe el bocadillo

        MPI_Recv ( &valor_rec, 1, MPI_INT, id_dependiente, 0, MPI_COMM_WORLD, &estado);

        if(id_propio % 2 == 0){
            cout << "\t\tEl alumno " << id_propio << " ha recibido 2 bocatas" << endl << flush
;
        }
        else{
            cout << "\t\tEl alumno " << id_propio << " ha recibido 1 bocata" << endl << flush;
        }

        //  3. se ha comido el bocadillo y se va

        cout << "\t\tEl alumno " << id_propio << " empieza a comer y se aleja del camion" << e
ndl << flush;
        sleep_for(seconds(aleatorio<1,2>()));

    }

}

void funcion_dependiente(){
   int         disponibles = 20,
               valor,
               peticion;
   MPI_Status  estado ;                   // Metadatos del mensaje recibido

    while(true){

      // 1. determinar si puede enviar solo cocinero., solo alumno, o todos

        cout << disponibles << " bocatas disponibles" << endl << flush;
        if( disponibles >= 3 ){
            MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado
);
        }
        else if (disponibles >= 1 ){
            MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, etiq_delgados, MPI_COMM_WORLD, &esta
do);
        }
        else{
            MPI_Ssend( &peticion, 1, MPI_INT, id_productor, 0, MPI_COMM_WORLD);
            MPI_Recv( &valor, 1, MPI_INT, id_productor, etiq_cocinero, MPI_COMM_WORLD, &estado
);
            cout << "Cocinero! Mas bocatas por favor!" << endl << flush;
        }


      switch( estado.MPI_TAG ){   // leer emisor del mensaje en metadatos

         case etiq_cocinero:  // si ha sido el cocinero: insertar bocatas
            disponibles = 20;
            break;

         case etiq_delgados:    // si ha sido el alumno: extraer y enviarle un bocata
            disponibles--;
            cout << "Dependiente va a enviar un bocata al alumno " << estado.MPI_SOURCE << end
l << flush;
            MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, 0, MPI_COMM_WORLD);
            break;
         case etiq_tragones:  // si ha sido el alumno tragón: extraer y enviarle 2 bocatas
            disponibles -= 2;
            cout << "Dependiente va a enviar dos bocatas al alumno " << estado.MPI_SOURCE << e
ndl << flush;
```

EXAMEN SCD P3 2020 - 2021

```
            MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, 0, MPI_COMM_WORLD);
            break;
        }
    }
}

int main( int argc, char *argv[] )
{
  int id_propio, num_procesos_actual; // ident. propio, núm. de procesos

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

  if ( num_procesos_esperado == num_procesos_actual )
  {
    if ( id_propio == id_productor )  // si mi ident. es el del cocinero
      funcion_cocinero();             //    ejecutar función cocinero
    else if ( id_propio == id_dependiente )// si mi ident. es el del dependiente
      funcion_dependiente();                //    ejecutar función buffer
    else                              // en otro caso, mi ident es alumno
      funcion_alumno(id_propio);            //    ejecutar función alumno
  }
  else if ( id_propio == 0 ){  // si hay error, el proceso 0 informa
    cerr << "error: número de procesos distinto del esperado." << endl ;
    cout << "el número de procesos esperados es:    " << num_procesos_esperado << endl
             << "el número de procesos en ejecución es: " << num_procesos_actual << endl
             << "(programa abortado)" << endl ;
  }

  MPI_Finalize( );
  return 0;
}
```