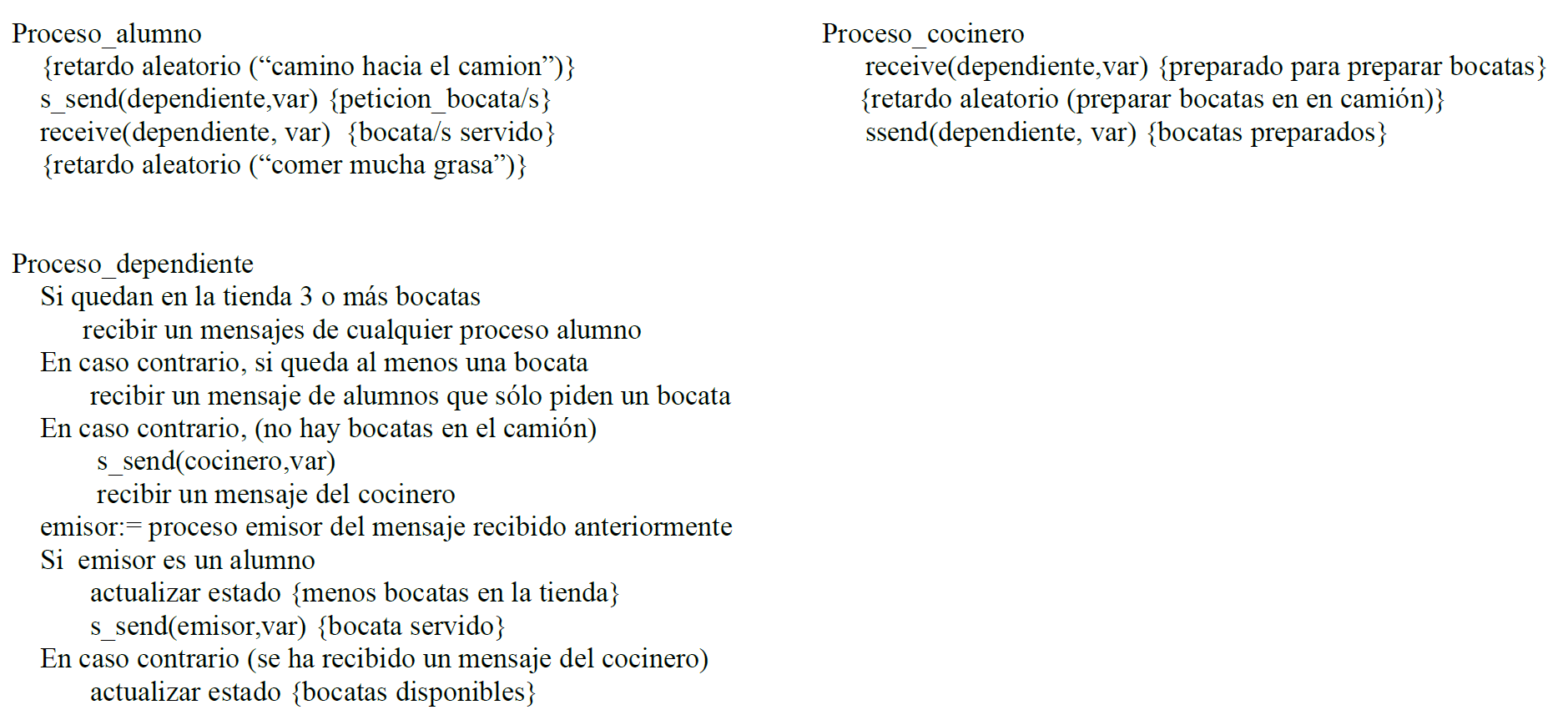
1. Construir un programa MPI, que se deberá llamar prod\_cons\_ex.cpp, con operaciones de paso de mensaje síncronas siguiendo el esquema del productor-consumidor, en el que tenemos cuatro procesos consumidores (identificadores del 0 al 3) y seis procesos productores (identificadores del 4 al 9) que producirán 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.

1. 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”).



## 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); //  Cuando 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;          //  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" << endl << 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, &estado);          }          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 << endl << 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 << endl << flush;              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;  } |