```
 1: // -----------------------------------------------------------------------------
 2: //
 3: // Sistemas concurrentes y Distribuidos.
 4: // PrÃ¡ctica 3. ImplementaciÃ³n de algoritmos distribuidos con MPI
 5: //
 6: // Archivo: filosofos-plantilla.cpp
 7: // ImplementaciÃ³n del problema de los filÃ³sofos (sin camarero).
 8: // Plantilla para completar.
 9: //
10: // Historial:
11: // Actualizado a C++11 en Septiembre de 2017
12: // -----------------------------------------------------------------------------
13:
14:
15: #include <mpi.h>
16: #include <thread> // this_thread::sleep_for
17: #include <random> // dispositivos, generadores y distribuciones aleatorias
18: #include <chrono> // duraciones (duration), unidades de tiempo
19: #include <iostream>
20:
21: using namespace std;
22: using namespace std::this_thread ;
23: using namespace std::chrono ;
24:
25: const int
26:    num_filosofos = 5 ,
27:    num_procesos  = 2*num_filosofos ;
28:
29:
30: //**************************************************************************
31: // plantilla de funciÃ³n para generar un entero aleatorio uniformemente
32: // distribuido entre dos valores enteros, ambos incluidos
33: // (ambos tienen que ser dos constantes, conocidas en tiempo de compilaciÃ³n)
34: //--------------------------------------------------------------------------
35:
36: template< int min, int max > int aleatorio()
37: {
38:   static default_random_engine generador( (random_device())() );
39:   static uniform_int_distribution<int> distribucion_uniforme( min, max ) ;
40:   return distribucion_uniforme( generador );
41: }
42:
43: // -----------------------------------------------------------------------------
44:
45: void funcion_filosofos( int id )
46: {
47:   int id_ten_1 , //id. tenedor izq.
```

```
48:       id_ten_2, //id. tenedor der.
49:       peticion;
50:
51:   while ( true )
52:   {
53:      if(id < num_filosofos){
54:         id_ten_1 = (id+1) % num_procesos;
55:         id_ten_2 = (id+num_procesos-1) % num_procesos;
56:      }
57:      else{
58:         id_ten_2 = (id + 1) % num_procesos;
59:         id_ten_1 = (id + num_procesos-1) % num_procesos;
60:      }
61:     cout <<"FilÃ³sofo " <<id << " solicita ten. 1." <<id_ten_1 <<endl;
62:     // ... solicitar tenedor izquierdo (completar)
63:     MPI_Ssend( &peticion, 1, MPI_INT, id_ten_1, 0, MPI_COMM_WORLD );
64:
65:     cout <<"FilÃ³sofo " <<id <<" solicita ten. 2." <<id_ten_2 <<endl;
66:     // ... solicitar tenedor derecho (completar)
67:     MPI_Ssend( &peticion, 1, MPI_INT, id_ten_2, 0, MPI_COMM_WORLD );
68:
69:     cout <<"FilÃ³sofo " <<id <<" comienza a comer" <<endl ;
70:     sleep_for( milliseconds( aleatorio<10,100>() ) );
71:
72:     cout <<"FilÃ³sofo " <<id <<" suelta ten. 1. " <<id_ten_1 <<endl;
73:     // ... soltar el tenedor izquierdo (completar)
74:     MPI_Ssend( &peticion, 1, MPI_INT, id_ten_1, 0, MPI_COMM_WORLD );
75:
76:     cout<< "FilÃ³sofo " <<id <<" suelta ten. 2. " <<id_ten_2 <<endl;
77:     // ... soltar el tenedor derecho (completar)
78:     MPI_Ssend( &peticion, 1, MPI_INT, id_ten_2, 0, MPI_COMM_WORLD );
79:
80:     cout << "Filosofo " << id << " comienza a pensar" << endl;
81:     sleep_for( milliseconds( aleatorio<10,100>() ) );
82:  }
83: }
84: // -----------------------------------------------------------------------
85:
86: void funcion_tenedores( int id )
87: {
88:   int valor, id_filosofo ;  // valor recibido, identificador del filÃ³sofo
89:   MPI_Status estado ;       // metadatos de las dos recepciones
90:
91:   while ( true )
92:   {
93:      // ..... recibir peticiÃ³n de cualquier filÃ³sofo (completar)
94:        MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
```

```
 95:       // ..... guardar en 'id_filosofo' el id. del emisor (completar)
 96:         id_filosofo = estado.MPI_SOURCE;
 97:       cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;
 98:       // ..... recibir liberaciÃ³n de filÃ³sofo 'id_filosofo' (completar)
 99:         MPI_Recv( &valor, 1, MPI_INT, id_filosofo, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
100:       cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id_filosofo <<endl ;
101:    }
102: }
103: // ---------------------------------------------------------------------
104:
105: int main( int argc, char** argv )
106: {
107:    int id_propio, num_procesos_actual ;
108:
109:    MPI_Init( &argc, &argv );
110:    MPI_Comm_rank( MPI_COMM_WORLD, &id_propio );
111:    MPI_Comm_size( MPI_COMM_WORLD, &num_procesos_actual );
112:
113:
114:    if ( num_procesos == num_procesos_actual )
115:    {
116:       // ejecutar la funciÃ³n correspondiente a 'id_propio'
117:       if ( id_propio % 2 == 0 )          // si es par
118:          funcion_filosofos( id_propio ); //   es un filÃ³sofo
119:       else                               // si es impar
120:          funcion_tenedores( id_propio ); //   es un tenedor
121:    }
122:    else
123:    {
124:       if ( id_propio == 0 ) // solo el primero escribe error, indep. del rol
125:       { cout << "el nÃºmero de procesos esperados es:    " << num_procesos << endl
126:              << "el nÃºmero de procesos en ejecuciÃ³n es: " << num_procesos_actual << endl
127:              << "(programa abortado)" << endl ;
128:       }
129:    }
130:
131:    MPI_Finalize( );
132:    return 0;
133: }
134:
135: // ---------------------------------------------------------------------
136:
```