Productor-Consumidor 2

Tendremos 3 procesos: 4 productores, 5 consumidores y un buffer.

- Productor:

```
18
19 void productor(int rank)
20 {
21    int value ;
22
23    for ( unsigned int value=0; value < ITERS/5 ; value++ )
24    {
25       cout << "Productor " << rank << " produce el valor " << value << endl << flush ;
26
27       // espera bloqueado durante un intervalo de tiempo aleatorio
28       // (entre una décima de segundo y un segundo)
29       usleep( 1000U * (100U+(rand()%900U)) );
30
31       // enviar 'value'
32       MPI_Ssend( &value, 1, MPI_INT, Buffer, Productor, MPI_COMM_WORLD );
33    }
34 }
```

Manda 4 números al proceso Buffer.

- Consumidor:

```
 80
 81 void consumidor(int rank)
 82 {
 83    int         value,
 84                peticion = 1 ;
 85    float       raiz ;
 86    MPI_Status  status ;
 87
 88    for (unsigned int i=0;i<ITERS/4;i++)
 89    {
 90       MPI_Ssend( &peticion, 1, MPI_INT, Buffer, Consumidor, MPI_COMM_WORLD );
 91       MPI_Recv ( &value, 1,    MPI_INT, Buffer, 0, MPI_COMM_WORLD,&status );
 92       cout << "Consumidor " << rank << "recibe el valor " << value << " de Buffer " << endl << flush ;
 93
 94       // espera bloqueado durante un intervalo de tiempo aleatorio
 95       // (entre una décima de segundo y un segundo)
 96       usleep( 1000U * (100U+(rand()%900U)) );
 97
 98       raiz = sqrt(value) ;
 99    }
100 }
```

Envía una petición al Buffer.
Recibe el dato del Buffer.

- Buffer:

```
36
37 void buffer()
38 {
39    int        value[TAM] ,
40               peticion ,
41               pos  =  0,
42               rama ;
43    MPI_Status status ;
44
45    for( unsigned int i=0 ; i < ITERS*2 ; i++ )
46    {
47       if ( pos==0 )      // el consumidor no puede consumir
48          rama = 0 ;
49       else if (pos==TAM) // el productor no puede producir
50          rama = 1 ;
51       else               // tanto el consumidor como elproductor pueden actuar
52       {
53          // leer 'status' del siguiente mensaje (esperando si no hay)
54          MPI_Probe( MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status );
55
56          // calcular la rama en función del origen del mensaje
57          if ( status.MPI_SOURCE == Productor )
58             rama = 0 ;
59          else
60             rama = 1 ;
61       }
62       switch(rama)
63       {
64          case 0:
65             MPI_Recv(&value[pos], 1, MPI_INT, MPI_ANY_SOURCE, Productor, MPI_COMM_WORLD, &status);
66             cout << "Buffer recibe " << value[pos] << " de Productor " << endl << flush;
67             pos++;
68             break;
69          case 1:
70             MPI_Recv(&peticion, 1, MPI_INT, MPI_ANY_SOURCE, Consumidor, MPI_COMM_WORLD, &status);
71             MPI_Ssend(&value[pos-1], 1, MPI_INT, status.MPI_SOURCE, 0, MPI_COMM_WORLD);
72             cout << "Buffer envía " << value[pos-1] << " a Consumidor " << endl << flush;
73             pos--;
74             break;
75       }
76    }
77 }
78
```

Establecemos dos casos distintos:
1- Cuando el Buffer envía datos al consumidor
2- Cuando el Buffer recibe datos del productor

Cuando el consumidor no puede consumir mas datos se activa el 2.
Cuando el productor no puede producir más datos se activa el 1.
En caso de que ambos puedan actuar se activará un caso u otro en función de la petición que se recoja antes.

*Al usar mensajes síncronos el MPI_Receive se espera hasta que reciba un mensaje.

Filósofos

Tenemos 10 procesos: 10 filósofos y 10 tenedores.

- Filósofos:

```
41
42 void Filosofo( int id, int nprocesos )
43 {
44    int izq = (id+1) % nprocesos;
45    int der = ((id+nprocesos)-1) % nprocesos;
46
47    while(1)   {
48     if(id == 0) {
49           // Solicita tenedor izquierdo
50           cout << "Filosofo "<<id<< " solicita tenedor izq ..." << izq << endl << flush;
51           MPI_Ssend(NULL, 0, MPI_INT, izq, cogerT, MPI_COMM_WORLD);
52
53           // Solicita tenedor derecho
54           cout <<"Filosofo "<<id<< " coge tenedor der ..." << der << endl << flush;
55           MPI_Ssend(NULL, 0, MPI_INT, der, cogerT, MPI_COMM_WORLD);
56     }else{
57           // Solicita tenedor derecho
58           cout <<"Filosofo "<<id<< " coge tenedor der ..." << der << endl << flush;
59           MPI_Ssend(NULL, 0, MPI_INT, der, cogerT, MPI_COMM_WORLD);
60
61           // Solicita tenedor izquierdo
62           cout << "Filosofo "<<id<< " solicita tenedor izq ..." << izq << endl << flush;
63           MPI_Ssend(NULL, 0, MPI_INT, izq, cogerT, MPI_COMM_WORLD);
64     }
65
66       cout<<"Filosofo "<<id<< " COMIENDO"<<endl<<flush;
67       sleep((rand() % 3)+1);  //comiendo
68
69       // Suelta el tenedor izquierdo
70       cout <<"Filosofo "<<id<< " suelta tenedor izq ..." << izq << endl << flush;
71       MPI_Ssend(NULL, 0, MPI_INT, izq, soltarT, MPI_COMM_WORLD);
72
73       // Suelta el tenedor derecho
74       cout <<"Filosofo "<<id<< " suelta tenedor der ..." << der << endl << flush;
75       MPI_Ssend(NULL, 0, MPI_INT, der, soltarT, MPI_COMM_WORLD);
76
77       // Piensa (espera bloqueada aleatorio del proceso)
78       cout << "Filosofo " << id << " PENSANDO" << endl << flush;
79
80       // espera bloqueado durante un intervalo de tiempo aleatorio
81       // (entre una décima de segundo y un segundo)
82       usleep( 1000U * (100U+(rand()%900U)) );
83  }
84 }
```

Envían peticiones a los tenedores (derecho e izquierdo).
Come.
Suelta ambos tenedores.
Piensa.

-Tenedor:

```
101 void Tenedor(int id, int nprocesos)
102 {
103   int buf;
104   MPI_Status status;
105   int Filo;
106
107   while( true )
108   {
109     // Espera un peticion desde cualquier filosofo vecino ...
110     MPI_Recv(&buf, 1, MPI_INT, MPI_ANY_SOURCE, cogerT, MPI_COMM_WORLD, &status);
111
112     // Recibe la peticion del filosofo ...
113     Filo=status.MPI_SOURCE; // Obtiene el rank del filosofo
114
115     cout << "Ten. " << id << " recibe petic. de " << Filo << endl << flush;
116
117     // Espera a que el filosofo suelte el tenedor...
118     MPI_Recv(&Filo, 1, MPI_INT, Filo, soltarT, MPI_COMM_WORLD, &status);
119     cout << "Ten. " << id << " recibe liberac. de " << Filo << endl << flush;
120   }
121 }
122
```

Espera la petición de un filósofos.
Espera a que el filósofo lo suelte.

Filósofos 2

Tenemos 11 procesos: 10 filósofos, 10 tenedores y un camarero.

- Filósofos:

```
47 void Filosofo( int id, int nprocesos )
48 {
49         int izq = (id+1) % (nprocesos-1);
50         int der = (id+nprocesos-2) % (nprocesos-1);
51     MPI_Status status;
52
53    while(1)   {
54
55         // El filosofo pide sentarse
56         cout << "Filosofo " << id << " solicita sentarse " <<  endl;
57         MPI_Send(NULL, 0, MPI_INT, camarero, sentarse, MPI_COMM_WORLD);
58
59         // El filosofo espera a que le digan que puede sentarse
60         MPI_Recv(NULL, 0, MPI_INT, camarero, sentarse, MPI_COMM_WORLD, &status);
61         cout << "Filosofo " << id << " se sienta " << endl;
62
63     // El filosofo se sienta
64
65     // Solicita tenedor derecho
66     cout <<"Filosofo "<<id<< " coge tenedor der ..." << der << endl << flush;
67     MPI_Ssend(NULL, 0, MPI_INT, der, cogerT, MPI_COMM_WORLD);
68
69     // Solicita tenedor izquierdo
70     cout << "Filosofo "<<id<< " solicita tenedor izq ..." << izq << endl << flush;
71     MPI_Ssend(NULL, 0, MPI_INT, izq, cogerT, MPI_COMM_WORLD);
72
73       cout<<"Filosofo "<<id<< " COMIENDO"<<endl<<flush;
74       sleep((rand() % 3)+1);  //comiendo
75
76       // Suelta el tenedor izquierdo
77       cout <<"Filosofo "<<id<< " suelta tenedor izq ..." << izq << endl << flush;
78       MPI_Ssend(NULL, 0, MPI_INT, izq, soltarT, MPI_COMM_WORLD);
79
80       // Suelta el tenedor derecho
81       cout <<"Filosofo "<<id<< " suelta tenedor der ..." << der << endl << flush;
82       MPI_Ssend(NULL, 0, MPI_INT, der, soltarT, MPI_COMM_WORLD);
83
84       // Piensa (espera bloqueada aleatorio del proceso)
85       cout << "Filosofo " << id << " PENSANDO" << endl << flush;
86
87       // el filosofo se levanta
88       cout << "Filosofo " << id << " se levanta " << endl;
89       MPI_Ssend(NULL, 0, MPI_INT, camarero, levantarse, MPI_COMM_WORLD );
90
91       cout << "Filosofo " << id << " PENSANDO" << endl;
92       sleep((rand() % 3)+1);  //pensando
93
94       // espera bloqueado durante un intervalo de tiempo aleatorio
95       // (entre una décima de segundo y un segundo)
96       usleep( 1000U * (100U+(rand()%900U)) );
97  }
98 }
99 // -----------------------------------------------------------------------
100
101 void Tenedor(int id, int nprocesos)
102 {
103   int buf;
104   MPI_Status status;
105   int Filo;
106
107   while( true )
108   {
109     // Espera un peticion desde cualquier filosofo vecino ...
110     MPI_Recv(&buf, 1, MPI_INT, MPI_ANY_SOURCE, cogerT, MPI_COMM_WORLD, &status);
111
112     // Recibe la peticion del filosofo ...
113     Filo=status.MPI_SOURCE; // Obtiene el rank del filosofo
114
115     cout << "Ten. " << id << " recibe petic. de " << Filo << endl << flush;
116
```

Pide permiso al camarero para poder sentarse.
Envían peticiones a los tenedores (derecho e izquierdo).
Come.
Suelta ambos tenedores.
Piensa.
Informa al camarero de que se ha levantado de la mesa.

-Tenedor:

```
101 void Tenedor(int id, int nprocesos)
102 {
103   int buf;
104   MPI_Status status;
105   int Filo;
106
107   while( true )
108   {
109     // Espera un peticion desde cualquier filosofo vecino ...
110     MPI_Recv(&buf, 1, MPI_INT, MPI_ANY_SOURCE, cogerT, MPI_COMM_WORLD, &status);
111
112     // Recibe la peticion del filosofo ...
113     Filo=status.MPI_SOURCE; // Obtiene el rank del filosofo
114
115     cout << "Ten. " << id << " recibe petic. de " << Filo << endl << flush;
116
117     // Espera a que el filosofo suelte el tenedor...
118     MPI_Recv(&Filo, 1, MPI_INT, Filo, soltarT, MPI_COMM_WORLD, &status);
119     cout << "Ten. " << id << " recibe liberac. de " << Filo << endl << flush;
120   }
121 }
122
```

Espera la petición de un filósofos.
Espera a que el filósofo lo suelte.

-Camarero:

```
122
123 void Camarero()
124 {
125     int buf, sentados=0;
126     MPI_Status status;
127     while (true) {
128         if (sentados < 4) // El maximo de filosofos comiendo es 4
129             MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status); // Puede sentarse o levantarse
130         else
131             MPI_Probe(MPI_ANY_SOURCE, levantarse, MPI_COMM_WORLD, &status);  // Solo puede levantarse
132
133
134         if (status.MPI_TAG == sentarse) // se le deja sentarse
135         {
136             buf=status.MPI_SOURCE;
137             MPI_Recv( NULL, 0, MPI_INT, buf, sentarse, MPI_COMM_WORLD,&status);
138             sentados++;
139
140             MPI_Send( NULL, 0, MPI_INT, buf, sentarse, MPI_COMM_WORLD);
141             cout << "Filosofo " << buf << " se sienta. Hay " << sentados << " filosofos sentados. " << endl;
142
143         }
144         if (status.MPI_TAG == levantarse) // Se levanta
145         {
146             buf=status.MPI_SOURCE;
147             MPI_Recv( NULL, 0, MPI_INT, buf, levantarse, MPI_COMM_WORLD,&status);
148             sentados--;
149             cout << "Filosofo " << buf << " se levanta. Hay " << sentados << " filosofos sentados.  " << endl;
150
151         }
152     }
153
154 }
```

Comprueba si hay sitio en la mesa.
De ser asi los filosofos pueden levantarse o sentarse, si se levanta el Camarero recibe el mensaje del filosofo, en coso contrario sería al revés.
Si no hay sitio, los filósofos solo pueden levantarse.