1 Productor Consumidor - Lukas Häring

1.1 Comentarios

Encontramos en éste ejemplo 3 semáforos, vamos a comentar cada uno de ellos

1. En la línea 18, encontramos un semáforo que regula cuándo hay suficiente productos o no, es decir si el vector tiene valors, entonces los consumidores pueden obtener los recursos y por ello al principio al no haber productos, éste debe estar en el valor 0, además es usado en **funcion_hebra_consumidora()** y llamado en **funcion_hebra_productora()** (cambio de 0 a 1) cuando se ha generado un producto.

2. En la línea 19, encontramos un semáforo con el valor del tamaño máximo del buffer, se encargará de indicarle al productor que el buffer está lleno y por lo tanto no deberá producir, si el buffer tiene espacio, éste se activará y podrá volver a llenar el vector.

3. La línea 20 contiene un semáforo que actúa de *mutex* (Es obvio que su valor debe estar en 1), se encarga de que los valores se vayan introduciendo o sacando del buffer correctamente de manera lineal (Uno detrás de otro), no es llamado por ninguna otra función más que **funcion_hebra_productora() y funcion_hebra_consumidora()**.

1.2 Código

```
1  #include <iostream>
2  #include <cassert>
3  #include <thread>
4  #include <mutex>
5  #include <random>
6  #include "Semaphore.h"
7
8  using namespace std;
9  using namespace SEM;
10
11 //**********************************************************************
12 // variables compartidas
13
14 const int num_items = 40, buffer_size = 10;   // tamanho del buffer
15 unsigned int buffer[buffer_size] = {0};
16 unsigned int cont_prod[num_items] = {0}; // contadores de verificacion:
       producidos
17 unsigned int cont_cons[num_items] = {0}; // contadores de verificacion:
       consumidos
18 Semaphore hay_productos(0);
19 Semaphore producto_completo(buffer_size);
20 Semaphore mutex_fuma(1);
21 unsigned int contador = 0, libre = 0;
22
23 //----------------------------------------------------------------------
24 // plantilla de funcion para generar un entero aleatorio uniformemente
25 // distribuido entre dos valores enteros, ambos incluidos
26 // (ambos tienen que ser dos constantes, conocidas en tiempo de compilacion)
27 //----------------------------------------------------------------------
28 template<int min, int max> int aleatorio(){
29   static default_random_engine generador((random_device())());
30   static uniform_int_distribution<int> distribucion_uniforme(min, max) ;
31   return distribucion_uniforme(generador);
```

```
32 }
33
34
35 //**********************************************************************
36 // funciones comunes a las dos soluciones (fifo y lifo)
37 //----------------------------------------------------------------------
38
39 int producir_dato(){
40   this_thread::sleep_for(chrono::milliseconds(aleatorio<20, 100>()));
41   mutex_fuma.sem_wait();
42   cout << "Producido: " << contador << endl << flush;
43   mutex_fuma.sem_signal();
44   cont_prod[contador]++;
45   return (contador = (contador+1) % num_items);
46 }
47 //----------------------------------------------------------------------
48
49 void consumir_dato(unsigned dato){
50   assert(dato < num_items);
51   cont_cons[dato]++;
52   this_thread::sleep_for(chrono::milliseconds(aleatorio<20, 100>()));
53   mutex_fuma.sem_wait();
54   cout << "Consumido: " << dato << endl ;
55   mutex_fuma.sem_signal();
56 }
57
58
59 //----------------------------------------------------------------------
60
61 void test_contadores(){
62   bool ok = true ;
63   cout << "Comprobando contadores ...." << endl;
64   for(unsigned i = 0; i < num_items; i++){
65     if (cont_prod[i] != 1){
66       cout << "error: Valor " << i << " producido " << cont_prod[i] << "
     veces." << endl;
67       ok = false ;
68     }
69   }
70   if (ok){
71     cout << "Solucion (aparentemente) correcta." << endl << flush;
72   }
73 }
74
75 //----------------------------------------------------------------------
76
77 void  funcion_hebra_productora(){
78   for(unsigned i = 0 ; i < num_items; i++){
79     int dato = producir_dato();
80     producto_completo.sem_wait();
81     mutex_fuma.sem_wait();
82     buffer[libre++] = dato;
83     mutex_fuma.sem_signal();
84     hay_productos.sem_signal();
85   }
86 }
87
88 //----------------------------------------------------------------------
89
90 void funcion_hebra_consumidora(){
```

```
91    for(unsigned i = 0 ; i < num_items; i++){
92      hay_productos.sem_wait();
93      mutex_fuma.sem_wait();
94      int dato = buffer[--libre];
95      mutex_fuma.sem_signal();
96      producto_completo.sem_signal();
97      consumir_dato(dato);
98    }
99  }
100 //----------------------------------------------------------------------

101
102 int main(){
103   cout << "-------------------------------------------------------------" << endl;
104   cout << "Problema de los productores-consumidores (solucion LIFO)." << endl
        ;
105   cout << "-------------------------------------------------------------" << endl;
106   cout << flush;
107
108   thread hebra_productora(funcion_hebra_productora);
109   thread hebra_consumidora(funcion_hebra_consumidora);
110
111   hebra_productora.join();
112   hebra_consumidora.join();
113
114   test_contadores();
115   return 1;
116 }
```