1 Fumadores - Lukas Häring

1.1 Comentarios

Encontramos en éste ejemplo (n+2) semáforos, vamos a comentar cada uno de ellos

1. En la línea 13, encontramos un semáforo para el estanquero, el cual tiene valor inicial 1, pues éste debe producir al principio, éste es llamado con un **sem_signal** cuando un consumidor aleatorio es elegido por la **función_hebra_estanquero** haciéndo que dicho consumidor empiece a consumir (Cambiando su signal a 1). Además el estanquero no producirá un producto hasta que que no sea pedido por un fumador tras haber consumido su producto en un tiempo aleatorio.

2. En la línea 14, encontramos "n" semáforos, pero en éste ejemplo están asignados 3 semáforos, éstos se encargan de controlar a los fumadores y comenzarán estando desactivados (Valor 0) pues no hay estanquero que les haya dado un producto, es decir estarán esperando hasta que el estanquero les haga un sem_signal aleatoriamente y así ellos poder realizar la función **fumar()** y finalmente volver a llamar al estanquero para que vuelva a producir.

3. La línea 15 contiene un semáforo que actúa de *mutex* (Valor inicial en 1), éste es usado para poder realizar llamadas al sistema (Como por ejemplo escribir en la consola) y que dos comentarios no se interpongan, pues así conseguimos una homogeneidad en los comentarios de cada una de las funciones, no se caracteriza por nada en concreto.

1.2 Código

```
1  #include <iostream>
2  #include <vector>
3  #include <cassert>
4  #include <thread>
5  #include <random> // dispositivos, generadores y distribuciones aleatorias
6  #include <chrono> // duraciones (duration), unidades de tiempo
7  #include "Semaphore.h"
8
9  using namespace std ;
10 using namespace SEM ;
11
12 const int CONSUMIDORES = 3;
13 Semaphore estanquero(1);
14 vector<Semaphore> consumidores;
15 Semaphore mutex_fuma(1);
16 //**********************************************************************
17 // plantilla de funcion para generar un entero aleatorio uniformemente
18 // distribuido entre dos valores enteros, ambos incluidos
19 // (ambos tienen que ser dos constantes, conocidas en tiempo de compilacion)
20 //----------------------------------------------------------------------
21
22 template< int min, int max > int aleatorio(){
23   static default_random_engine generador( (random_device())() );
24   static uniform_int_distribution<int> distribucion_uniforme( min, max ) ;
25   return distribucion_uniforme( generador );
26 }
27
28 //----------------------------------------------------------------------
```

```
29 // funcion que ejecuta la hebra del estanquero
30 void funcion_hebra_estanquero(){
31   while(true){
32     estanquero.sem_wait();
33
34     // Mutex nos permite escribir el cout
35     int num_fumador = aleatorio<0,CONSUMIDORES-1>();
36     mutex_fuma.sem_wait();
37     cout << endl << "Produce para el fumador: " << num_fumador << endl;
38     mutex_fuma.sem_signal();
39     consumidores[num_fumador].sem_signal();
40   }
41 }
42
43 //----------------------------------------------------------------------
44 // Funcion que simula la accion de fumar, como un retardo aleatoria de la
       hebra
45 void fumar(int num_fumador){
46
47   // calcular milisegundos aleatorios de duracion de la accion de fumar)
48   chrono::milliseconds duracion_fumar( aleatorio<20,200>() );
49
50   // informa de que comienza a fumar
51   mutex_fuma.sem_wait();
52   cout << "Fumador " << num_fumador << " empieza a fumar (" << duracion_fumar
       .count() << " milisegundos)" << endl;
53   mutex_fuma.sem_signal();
54
55   // espera bloqueada un tiempo igual a ''duracion_fumar' milisegundos
56   this_thread::sleep_for(duracion_fumar);
57
58   // informa de que ha terminado de fumar
59   mutex_fuma.sem_wait();
60   cout << "Fumador " << num_fumador << "  : termina de fumar, comienza espera
       de ingrediente." << endl;
61   mutex_fuma.sem_signal();
62
63 }
64
65 //----------------------------------------------------------------------
66 // funcion que ejecuta la hebra del fumador
67 void  funcion_hebra_fumador(int num_fumador){
68   while(true){
69     consumidores[num_fumador].sem_wait();
70     mutex_fuma.sem_wait();
71     cout << "Consume";
72     mutex_fuma.sem_signal();
73     fumar(num_fumador);
74     estanquero.sem_signal();
75   }
76 }
77
78 //----------------------------------------------------------------------
79
80 int main(){
81   // declarar hebras y ponerlas en marcha
82   thread hebra_productora(funcion_hebra_estanquero);
83   vector<thread> hebra_consumidora;
84
85   // Declaramos semaforos en "rojo" para cada consumidor
```

```
86    // pues estos no pueden consumir si al principio no hay nada producido.
87    for(int f = 0; f < CONSUMIDORES; f++){ consumidores.push_back(Semaphore(0))
        ; }
88
89    // Creamos una hebra para cada consumidor.
90    for(int f = 0; f < CONSUMIDORES; f++){ hebra_consumidora.push_back(thread(
        funcion_hebra_fumador, f)); }
91
92    // Anhadimos en la cola de ejecucion.
93    for (auto& hebra : hebra_consumidora){
94      hebra.join();
95    }
96    hebra_productora.join();
97
98    return 0;
99  }
```