El problema del productor-consumidor

Necesito dos variables, que he llamado entrada y salida. Estas dos se encargarán de llevar la cuenta de por donde vamos en el vector en cada una de las hebras (productora y consumidora respectivamente). Además usaremos dos variables de tipo semáforo que he llamado puede_escribir y puede_leer. Al comienzo del programa el semáforo de escribir se encontrará a 1, para que se pueda realizar dicha acción y por el contrario leer estará a 0. La utilidad de estos reside en que controlan si una hebra consumidora, en este caso que lee, lo puede hacer después de que una hebra productora haya escrito sobre la misma variable que la otra hebra pretende leer.

```
#include <iostream>
#include <cassert>
#include <thread>
#include <mutex>
#include <random>
#include "Semaphore.h"

using namespace std ;
using namespace SEM ;

//**********************************************************************
// variables compartidas

const int num_items = 40 ,   // número de items
          tam_vec   = 10 ;   // tamaño del buffer
unsigned  cont_prod[num_items] = {0}, // contadores de verificación: producidos
          cont_cons[num_items] = {0}; // contadores de verificación: consumidos

// Semáforos compartidos
Semaphore puede_escribir = 1;
Semaphore puede_leer = 0;

int entrada = 0, salida = 0;
int buffer[tam_vec];


//**********************************************************************
// 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 );
}

//**********************************************************************
// funciones comunes a las dos soluciones (fifo y lifo)
//----------------------------------------------------------------------

int producir_dato()
{
   static int contador = 0 ;
   this_thread::sleep_for( chrono::milliseconds( aleatorio<20,100>() ));

   cout << "producido: " << contador << endl << flush ;

   cont_prod[contador] ++ ;
   return contador++ ;
}
//----------------------------------------------------------------------

void consumir_dato( unsigned dato )
```

```
{
   assert( dato < num_items );
   cont_cons[dato] ++ ;
   this_thread::sleep_for( chrono::milliseconds( aleatorio<20,100>() ));

   cout << "                  consumido: " << dato << endl ;

}


//----------------------------------------------------------------------

void test_contadores()
{
   bool ok = true ;
   cout << "comprobando contadores ...." ;
   for( unsigned i = 0 ; i < num_items ; i++ )
   {  if ( cont_prod[i] != 1 )
      {  cout << "error: valor " << i << " producido " << cont_prod[i] << " veces." << endl ;
         ok = false ;
      }
      if ( cont_cons[i] != 1 )
      {  cout << "error: valor " << i << " consumido " << cont_cons[i] << " veces" << endl ;
         ok = false ;
      }
   }
   if (ok)
      cout << endl << flush << "solución (aparentemente) correcta." << endl << "fin" << endl <
< flush ;
}

//----------------------------------------------------------------------

void  funcion_hebra_productora(  )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      int dato = producir_dato() ;
      // completar ........
      sem_wait(puede_escribir);

      buffer[entrada] = dato;
      entrada = (entrada+1)%tam_vec;

      sem_signal(puede_leer);
   }
}

//----------------------------------------------------------------------

void funcion_hebra_consumidora(  )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      int dato ;
      // completar ......
      sem_wait(puede_leer);
      dato=buffer[salida];
      salida=(salida+1)%tam_vec;

      sem_signal(puede_escribir);
      consumir_dato( dato ) ;
    }
}
//----------------------------------------------------------------------

int main()
{
   cout << "-----------------------------------------------------" << endl
        << "Problema de los productores-consumidores (solución LIFO)." << endl
        << "-----------------------------------------------------" << endl
        << flush ;

   thread hebra_productora ( funcion_hebra_productora ),
```

```
          hebra_consumidora( funcion_hebra_consumidora );

   hebra_productora.join() ;
   hebra_consumidora.join() ;

   test_contadores();
}
```

El problema de los fumadores.

Para la correcta realización de esta parte, he usado 4 semáforos en total, uno de ellos será el del mostrador que indicará si el estanquero se encuentra disponible para atender a un fumador, tomando el valor 1, o por el contrario si ya está atendiendo a alguno, tomando el valor 0. A esto le añadiremos 3 semáforos en un vector que son cada uno de los ingredientes que se producen. Se encuentran inicializados a 0 hasta que no se produce alguno de ellos. Cada uno de los fumadores, será una hebra qué realizarán un *wait* o un *signal* dependiendo de si se encuentran fumando, o si se encuentran a la espera de que aparezca el ingrediente que necesitan para continuar fumando en el mostrador.

```
#include <iostream>
#include <cassert>
#include <thread>
#include <mutex>
#include <random> // dispositivos, generadores y distribuciones aleatorias
#include <chrono> // duraciones (duration), unidades de tiempo
#include "Semaphore.h"

using namespace std ;
using namespace SEM ;

const int num_fumadores = 3;
Semaphore mostrador=1; //1 si no se atiende a nadie, 0 si está ocupado con algún fumador
std::vector<Semaphore> ingredientes; // 1 si ingrediente i está disponible, 0 si no. Inicializ
ado a 0 para solo poder entrar en la función estanquero.

//**********************************************************************
// 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 );
}

//----------------------------------------------------------------------
// Función que simula la acción de producir un ingrediente, como un retardo
// aleatorio de la hebra (devuelve número de ingrediente producido)

int producir_ingrediente()
{
   // calcular milisegundos aleatorios de duración de la acción de fumar)
```

```
   chrono::milliseconds duracion_produ( aleatorio<10,100>() );

   // informa de que comienza a producir
   cout << "Estanquero : empieza a producir ingrediente (" << duracion_produ.count() << " mili
segundos)" << endl;

   // espera bloqueada un tiempo igual a ''duracion_produ' milisegundos
   this_thread::sleep_for( duracion_produ );

   const int num_ingrediente = aleatorio<0,num_fumadores-1>() ;

   // informa de que ha terminado de producir
   cout << "Estanquero : termina de producir ingrediente " << num_ingrediente << endl;

   return num_ingrediente ;
}

//----------------------------------------------------------------------
// función que ejecuta la hebra del estanquero

void funcion_hebra_estanquero(  )
{
   int num_fumador;
   while(true){
      sem_wait(mostrador);
      num_fumador=producir_ingrediente();

      cout << "Se ha puesto el ingrediente número: " << num_fumador << endl;


      sem_signal(ingredientes[num_fumador]);
   }

}

//----------------------------------------------------------------------
// Función que simula la acción de fumar, como un retardo aleatoria de la hebra

void fumar( int num_fumador )
{

   // calcular milisegundos aleatorios de duración de la acción de fumar)
   chrono::milliseconds duracion_fumar( aleatorio<20,200>() );

   // informa de que comienza a fumar

    cout << "Fumador " << num_fumador << "  :"
          << " empieza a fumar (" << duracion_fumar.count() << " milisegundos)" << endl;

   // espera bloqueada un tiempo igual a ''duracion_fumar' milisegundos
   this_thread::sleep_for( duracion_fumar );

   // informa de que ha terminado de fumar

    cout << "Fumador " << num_fumador << "  : termina de fumar, comienza espera de ingrediente
." << endl;

}

//----------------------------------------------------------------------
// función que ejecuta la hebra del fumador
void  funcion_hebra_fumador( int num_fumador )
{
   while( true )
```

```
   {
      sem_wait(ingredientes[num_fumador]);


      cout << "Se retira el ingrediente número: " << num_fumador << endl;


      sem_signal(mostrador);
      fumar(num_fumador);

   }
}

//----------------------------------------------------------------------

int main()
{
   // declarar hebras y ponerlas en marcha
   // ......

   for(int i = 0; i < num_fumadores; i++)
      ingredientes.push_back(0);

   thread hebra_estanquero(funcion_hebra_estanquero);
   thread hebra_fumador[num_fumadores];

   for(unsigned long i = 0; i < num_fumadores; i++)
      hebra_fumador[i] = thread(funcion_hebra_fumador, i);
   hebra_estanquero.join();

   for(unsigned long i = 0; i < num_fumadores; i++)
      hebra_fumador[i].join();


}
```