Productor-consumidor

Las variables utilizados son :

num_items : numero de datos que se van a generar.

tam_vector : tamaño del vector intermedio.

primera_libre : indicador de posicion del vector para productor.

primera_ocupada : indicador de posicion del vector para consumidor

Para determinar en que posicion de puede escribir y en que posicion se puede leer, utilizamos los indicadores primera_libre y primera_ocupada para productor y consumidor respectivamente. Estos indicadores va incrementando de uno en uno conforme se vaya produciendo datos o consumiendo. Cuando llega al final del vector, mediante la operacion modulo, vuelven a apuntarse a la primera posicion del vector.

Los semáforos utilizados son las siguientes :

puede_escribir : se utiliza para controlar el productor, para que no produzca más datos cuando el vector ya esta lleno.

Su valor inicial es tamaño del vector.

El sem_wait se utiliza antes de producir el dato, y sem_post despues de consumir el dato.

puede_leer : se utiliza para que el consumidor no lea los datos cuando en el vector no hay ningun valor pendiente de leer.

Su valor inicial es 0.

El sem_wait se utiliza antes de leer el dato, y sem_post despues de producir el dato.

mutex : se utiliza para sincronizar las salidas.

Su valor inicial es 1.

El sem_wait se utiliza antes de los cout, y el sem_post despues de los cout.

Codigo:

```
#include <iostream>
#include <cassert>
#include <pthread.h>
#include <semaphore.h>

using namespace std ;

// ------------------------------------------------------------------
--
// constantes
const unsigned
```

```
   num_items    = 60,
   tam_vector = 10 ;



// ------------------------------------------
   int vec_inter[tam_vector];
   int primera_libre=0;
   int primera_ocupada=0;


   sem_t puede_escribir;
   sem_t puede_leer;
   sem_t mutex;




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


unsigned producir_dato()
{
   static int contador = 0 ;
   sem_wait(&mutex);
```

```
   cout << "producido: " << contador << endl << flush ;
   sem_post(&mutex);
   return contador++ ;
}
// ----------------------------------------------------------------------
                        --

void consumir_dato( int dato )
{
                        sem_wait(&mutex);
      cout << "                                    dato recibido: " << dato << endl ;
      sem_post(&mutex);

}
// ----------------------------------------------------------------------
                        --

void * productor( void * )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      sem_wait(&puede_escribir);
```

```
                        int dato = producir_dato() ;
      // falta: insertar "dato" en el vector
                                vec_inter[primera_libre]=dato;
                                primera_libre=(primera_libre+1)%tam_vector;

                            sem_post(&puede_leer);
      // .................
   }
   return NULL ;
}
// ----------------------------------------------------------------------
                        --

void * consumidor( void * )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      int dato ;

                            sem_wait(&puede_leer);
      // falta: leer "dato" desde el vector intermedio
                              dato=vec_inter[primera_ocupada];
```

```
                        primera_ocupada=(primera_ocupada+1) %tam_vector;
      // .................
                        consumir_dato( dato ) ;
                        sem_post(&puede_escribir);

   }
   return NULL ;
}
//----------------------------------------------------------------------
                    ---

int main()
{

   // falta: crear y poner en marcha las hebras, esperar que terminen
   // ....
   sem_init(&puede_escribir,0,tam_vector);
   sem_init(&puede_leer,0,0);
   sem_init(&mutex,0,1);


   pthread_t hebra_productor,hebra_consumidor;
```

```
    pthread_create(&hebra_productor,NULL,productor,NULL);
    pthread_create(&hebra_consumidor,NULL,consumidor,NULL);

    pthread_join(hebra_productor,NULL);
    pthread_join(hebra_consumidor,NULL);

    sem_destroy(&puede_escribir);
    sem_destroy(&puede_leer);
    sem_destroy(&mutex);

    cout<<"FIN DEL PROGRAMA";

    pthread_exit(NULL);

     return 0 ;
}
```