

Problema del productor - consumidor

1. Escritura y lectura

He creado las variables:

- *libre*: índice en el vector de la primera celda libre.
- *buffer[]*: vector auxiliar

Se puede escribir en la posición número *libre* y se puede leer en la posición *libre* - 1.

La variable *libre* se aumenta cuando se escribe un dato y se disminuye cuando se lee, por tanto la solución implementada usa el método LIFO.

2. Semáforos

Se necesitan tres semáforos:

- *produce*: controla que *funcion_productor* funcione solo cuando *funcion_consumidor* haya terminado.
- *consume*: controla que *funcion_consumidor* funcione solo cuando *funcion_productor* haya terminado.
- *mutex*: es un semáforo genérico que controla que todo se haga en orden.

3. Código Fuente

```
#include <iostream>
#include <cassert>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h> // necesario para {\ttbf usleep()}
#include <stdlib.h> // necesario para {\ttbf random()}, {\ttbf srand()}
#include <time.h> // necesario para {\ttbf time()}

using namespace std ;

// -----
// constantes configurables:

const unsigned num_items = 60 , // numero total de items que se producen o consumen
tam_vector = 10 ; // tamaño del vector, debe ser menor que el número de items

int libre = 0;
int buffer[tam_vector]; //Vector auxiliar

sem_t produce, consume, mutex;

// -----
// introduce un retraso aleatorio de duración comprendida entre
// 'smin' y 'smax' (dados en segundos)

void retraso_aleatorio( const float smin, const float smax ){
    static bool primera = true ;

    if ( primera ){ // si es la primera vez:
        srand(time(NULL)); // inicializar la semilla del generador
        primera = false ; // no repetir la inicialización
    }

    // calcular un número de segundos aleatorio, entre {\ttbf smin} y {\ttbf smax}

    const float tsec = smin+(smax-smin)*((float)random()/(float)RAND_MAX);

    // dormir la hebra (los segundos se pasan a microsegundos, multiplicándos por 1 millón)

    usleep( (useconds_t) (tsec*1000000.0) );
}

// -----
// función que simula la producción de un dato
```

```
unsigned producir_dato(){
    static int contador = 0 ;
    contador = contador + 1 ;
    retraso_aleatorio( 0.1, 0.5 );

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

    return contador ;
}
// -----
// función que simula la consumición de un dato

void consumir_dato( int dato ){
    retraso_aleatorio( 0.1, 1.5 );

    cout << "Consumidor:                                dato consumido: " << dato << endl << flush ;
}
// -----
// función que ejecuta la hebra del productor

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

        sem_wait(&produce);
        sem_wait(&mutex);

        buffer[libre] = dato;    //Inserto el dato en el vector intermedio
        libre++;

        sem_post(&consume);
        sem_post(&mutex);

        cout << "Productor:                                dato insertado: " << dato << endl << flush ;
    }
    return NULL ;
}
// -----
// función que ejecuta la hebra del consumidor

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

        sem_wait(&consume);
```

```
        sem_wait(&mutex);

        dato = buffer[libre - 1]; //Leo el dato desde el vector intermedio
        libre--;

        sem_post(&produce);
        sem_post(&mutex);

        cout << "Consumidor:          dato extraído : " << dato << endl << flush ;
        consumir_dato( dato ) ;
    }
    return NULL ;
}
//-----

int main(){
    pthread_t productor, consumidor;

    sem_init(&produce, 0, tam_vector);
    sem_init(&consume, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&productor, NULL, funcion_productor, NULL);
    pthread_create(&consumidor, NULL, funcion_consumidor, NULL);

    pthread_join(productor, NULL);
    pthread_join(consumidor, NULL);

    cout << "\n\nfin\n\n";

    sem_destroy(produce);
    sem_destroy(consume);
    sem_destroy(mutex);

    return 0;
}
```