SCD PRACTICA 1: PROBLEMA PRODUCTOR CONSUMIDOR

ANTONIO GARCIA CASTILLO

Para resolver el problema del productor consumidor he optado por una solución en la que he incluido dos variables de posición y dos variables de tipo semáforo. A continuación detallo cada una de ellas.

Variables Globales Usadas

Const int num_items = 40; → Número de ítems a producir, valor arbitrario

Const int tam_vec = 10; → Valor que vamos a asignar a nuestro vector buffer intermedio

Int pos_escribir = 0; →Determinará la posición en el buffer donde escribiremos nuestro ítem

Int pos_leer = 0; →Determinará la posición en el buffer de donde obtendremos la lectura de nuestro ítem

I**nt vector_buffer[tam_vec];** → Buffer en el que almacenamos los datos producidos y de donde se obtienen las lecturas

Semaphore puede_escribir = tam_vec; → Este semáforo lo usaremos para poder saber si la escritura de un dato en el buffer es posible o no. Las razones por la cual lo inicializamos a tam_vector son que este problema debe empezar obviamente por la escritura, lógicamente, dado lo cual debe estar inicializado a mas de cero. La razón por la cual es tam_vector es que de esta manera nos aseguramos que nunca se escriba encima de una posición que tiene que ser leída y sí poder hacer tam_vector escrituras seguidas sin esperar a su lectura.
Este semáforo se usará al principio de la hebra productora decreciendo en una unidad (sem_wait) y al final de la hebra consumidora aumentando en una unidad (sem_signal).

Semaphore puede_leer=0 →Este semáforo lo usaremos para poder saber si la lectura de un dato procedente del buffer es posible o no. Inicializamos el valor del semáforo a cero pues tenemos que forzar a que primero haya una escritura en el buffer.
Este semáforo se usara al final de la hebra productora (sem_signal) y al principio de la hebra consumidora (sem_wait).

SCD PRACTICA 1: PROBLEMA PRODUCTOR CONSUMIDOR

ANTONIO GARCIA CASTILLO

A continuación muestro el código de ambas funciones y la posición de los semáforos, permitiendo perfectamente la concurrencia entre ambas. Se explica paso a paso el procedimiento seguido hasta el consumo del dato.

```
void  funcion_hebra_productora(  )
{
  int dato;                                // Variable local donde almacenamos nuestro dato producido
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
     dato = producir_dato();               // Producimos dato y almacenamos en dato
     sem_wait(puede_escribir);             // Entramos y bajamos en una unidad nuestro semaforo
     vector_buffer[pos_escribir]=dato;     // Escribimos en el buffer nuestro dato
     pos_escribir++;                       // Aumentamos nuestra posicion de escritura
     pos_escribir = pos_escribir%tam_vec; // nos aseguramos de no salirnos del buffer
     sem_signal(puede_leer);               // Aumentamos nuestro semaforo, señal para poder leer
   }
 }
//----------------------------------------------------------------------
void funcion_hebra_consumidora(  )
{
  int dato ;                               // Variable local donde almacenamos el dato leido
   for( unsigned i = 0 ; i < num_items ; i++ )
    {
       sem_wait(puede_leer);               // Esperamos a la señal procedente de la hebra productora
       dato=vector_buffer[pos_leer];       // leemos nuestro dato y lo guardamos en dato
       pos_leer++;                         // aumentamos nuestra posicion de lectura
       pos_leer = pos_leer%tam_vec;        // nos aseguramos de que no se salga del buffer
       sem_signal(puede_escribir);         // Aumentamos el semaforo puede_escribir en una unidad
       consumir_dato(dato);
     }
}
```