3º curso / 1º cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Sistemas Concurrentes y Distribuidos
Práctica 1
Problema 1.

Estudiante: Miguel Lentisco Ballesteros
Grupo de prácticas: A1

Productor/Consumidor

Problema. Se nos pide dar una solución al problema del Productor/Consumidor siguiendo la plantilla proporcionada. Para la solución se usará como estructura para el buffer tanto como una estructura *LIFO* (pila), como *FIFO* (cola).

LIFO

Con el buffer como una pila, para saber donde estamos escribiendo y leyendo tendremos que usar una variable global que funcione como índice, esta variable será incrementada cuando el Productor escriba en el buffer y será decrementada y luego se lee el dato por el Consumidor, representando así el índice la posición donde tiene que escribir o leer respectivamente.

Por supuesto tenemos que tener en cuenta que pasa cuando vayan a escribir y leer en el buffer el Consumidor y el Productor van a tener que modificar la variable indice pudiendo probar problemas de escritura/lectura; por ello cuando vayamos a usar esta variable tendremos que usar exclusión mutua, en este caso usaremos un semaforo de acceso al buffer.

FIFO

Si el buffer es una cola, tenemos dos posibilidades en este caso:

- En este caso, el bucle del productor es hasta la cantidad de items a producir, luego podemos aprovechar esta ventaja y usar la variable *i* como el índice ya que simplemente se va a escribir en *i* %*buffer_tam* cuando el semáforo de escritura lo permita (explicado a continuación), e independientemente del productor; el consumidor irá leyendo en el mismo orden *i* %*buffer_tam* cuando el semáforo de lectura deje hacerlo, por tanto tendremos variables locales que no interfieren entre sí, no necesitamos exclusión mutua.
- Por otro lado, si tuviéramos que llevar la cuenta inicialmente o en cualquier momento saber donde se está escribiendo o leyendo entonces tendríamos que tener dos índices, uno de escritura y otro de lectura, variables globales para saberlo. Igualmente que en el primer caso, estas variables serían independientes entre sí y no habría problemas de carrera, luego de nuevo, no necesitamos exclusión mutua. Se hace igual, sustituyendo *i* por el índice lectura/escritura correspondiente y incrementando cada índice al escribir/leer en el buffer.

Semáforo

Ahora, para resolver el problema con semáforos, visto ya en clase, solo tendremos que poner tres semáforos, uno que sea para el productor (para escribir), otro para el consumidor (para leer) y finalmente para el buffer (acceso a él). Inicializamos el de escritura al tamaño del buffer (puede escribir hasta el máximo), el de lectura a 0 (tiene que esperar a que el productor escriba) y el del buffer a 1 (solo una hebra en el buffer en cada momento); entonces cada vez que se vaya a producir haya un *sem_wait* de escritura, y cuando escriba se produzca un *sem_signal* de lectura para el consumidor que está esperando en *sem_wait* de lectura, pueda leer el dato y dar un *sem_signal* para indicar que hay un hueco vacío al productor.

El semáforo de escritura sirve para parar al consumidor para que no siga escribiendo si el buffer está completo, y el de lectura para el consumidor que se espere a que haya datos que leer. Obviamente al productor le damos un margen de libertad del tamaño del buffer al principio y al de lectura ninguno, ya que se tiene que esperar a que el productor escriba.

El semáforo del buffer sirve para implementar la exclusión mútua de sección crítica, cuando se modifica el buffer y su índice solo puede hacerlo una hebra a la vez.

Solución

En los archivos fuentes, se encuentran las soluciones con *FIFO* y *LIFO* a este problema siguiendo la plantilla proporcionada y completando debidamente las funciones de consumidor y productor, añadiendo las variables globales consideradas (índices y buffer) y mostrando por pantalla cada vez que se escribe/lee en buffer así como cuando cada hebra acaba su cometido con un *fin*. Por supuesto, está comprobado con muchas veces que funciona correctamente.

Solución LIFO

Veamos la solución *LIFO*:

Código fuente 1: prodcons-lifo.cpp

```
1  // variables compartidas
2
3  const int num_items = 40 ,   // número de items
4            tam_vec   = 10 ;   // tamaño del buffer
5  unsigned  cont_prod[num_items] = {0}, // contadores de verificación: producidos
6            cont_cons[num_items] = {0}; // contadores de verificación: consumidos
7  int buffer[tam_vec] = {0}; // Buffer donde ir guardando los datos
8  int indicePila = 0; // Indice de la pila/buffer
9  Semaphore accesoBuffer = 1; // Semáforo para acceder al buffer
10 Semaphore puedoEscribir = tam_vec;  // Semáforo para ir escribiendo
11 Semaphore puedoLeer = 0; // Semáforo para ir leyendo
12 //----------------------------------------------------------------------
13
14 void funcion_hebra_productora() {
15    // Inicializamos el índice de la pila a 0
16    indicePila = 0;
17    for (unsigned i = 0; i < num_items; ++i) {
18       // Producimos el dato
19       int dato = producir_dato();
20       // Espera del semáforo de escritura
21       sem_wait(puedoEscribir);
22       // Escribimos en el buffer e incrementamos el índice
```

```
      sem_wait(accesoBuffer);
      buffer[indicePila++] = dato;
      int aux = indicePila - 1;
      sem_signal(accesoBuffer);
      // Mostramos por pantalla la escritura en buffer
      cout << "Escribo en buffer[" << aux << "] = " << dato << endl;
      // Señal para leer del semáforo de lectura
      sem_signal(puedoLeer);
   }
   // Por pantalla cuando ha acabado de producir
   cout << "\nHebra productora: fin." << endl;
}

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

void funcion_hebra_consumidora() {
   for (unsigned i = 0; i < num_items; ++i) {
      int dato;
      // Espera al semáfoto de lectura
      sem_wait(puedoLeer);
      // Leemos el dato del buffer, decrementando antes el índice
      sem_wait(accesoBuffer);
      dato = buffer[--indicePila];
      int aux = indicePila;
      sem_signal(accesoBuffer);
      // Mostramos por pantalla la lectura
      cout << "                    Leo del buffer[" << aux << "] = " << dato << endl;
      // Señal para escribir al semáforo de escritura
      sem_signal(puedoEscribir);
      // Consumimos el dato
      consumir_dato(dato);
    }
    // Por pantalla cuando ha acabado de consumir
    cout << "\nHebra consumidora: fin." << endl;
}

//----------------------------------------------------------------------
```

Solución FIFO

Veamos la solución *FIFO*:

Código fuente 2: prodcons-fifo.cpp

```
// 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
int buffer[tam_vec] = {0}; // Buffer donde ir guardando los datos
int indiceEscritura = 0; // Indice para escribir
int indiceLectura = 0; // Indice para leer
Semaphore puedoEscribir = tam_vec;  // Semaforo para ir escribiendo
Semaphore puedoLeer = 0; // Semáforo para ir leyendo
//----------------------------------------------------------------------

void funcion_hebra_productora() {
```

```
15    for (unsigned i = 0; i < num_items; ++i) {
16       // Producimos el dato
17       int dato = producir_dato();
18       // Espera del semáforo de escritura
19       sem_wait(puedoEscribir);
20       // Escribimos en el buffer e incrementamos el índice
21       buffer[indiceEscritura++ % tam_vec] = dato;
22       // Mostramos por pantalla la escritura en buffer
23       cout << "Escribo en buffer[" << (indiceEscritura-1) << "] = " << dato << endl;
24       // Señal para leer del semáforo de lectura
25       sem_signal(puedoLeer);
26    }
27    // Por pantalla cuando ha acabado de producir
28    cout << "\nHebra productora: fin." << endl;
29 }
30
31 //----------------------------------------------------------------------
32
33 void funcion_hebra_consumidora() {
34    for (unsigned i = 0; i < num_items; ++i) {
35       int dato;
36       // Espera al semáfoto de lectura
37       sem_wait(puedoLeer);
38       // Leemos el dato del buffer e incrementamos el índice
39       dato = buffer[indiceLectura++ % tam_vec];
40       // Mostramos por pantalla la lectura
41       cout << "Leo del buffer[" << (indiceLectura-1) << "] = " << dato << endl;
42       // Señal para escribir al semáforo de escritura
43       sem_signal(puedoEscribir);
44       // Consumimos el dato
45       consumir_dato(dato);
46     }
47     // Por pantalla cuando ha acabado de consumir
48     cout << "\nHebra consumidora: fin." << endl;
49 }
```