Seminario 2 - Productores - Consumidores con monitores SU

Salvador Romero Cortés

El ejercicio de este seminario consiste en resolver el problema de productores-consumidores usando monitores de espera urgente, tanto la versión FIFO como la versión LIFO.

Versión LIFO:

- Monitor:

```
class ProdConsLIFO : public HoareMonitor
{
private:
   static const int num_celdas_total = 10;
   int buffer[num_celdas_total];
   int primera_libre;
   CondVar ocupadas, libres;

public:
   ProdConsLIFO();
   void insertar(int dato);
   int extraer();
};
```

  Aquí tenemos el monitor SU que se encarga de gestionar las hebras. Tenemos un buffer donde se van insertando y extrayendo los valores que producen o leen las hebras. Aparece también un entero que almacena la posición de donde se lee o se extrae (`int primera_libre;`). Usaremos esta variable también para las variables de tipo `CondVar`.

  Además disponemos de un constructor que inicializa los valores a 0 así como preparar las variables condición.

- Método insertar:

```
void ProdConsLIFO::insertar(int dato)
{
   if (primera_libre == num_celdas_total-1)
      libres.wait();

   buffer[primera_libre] = dato;
   primera_libre++;
   ocupadas.signal();
}
```

  La condición de espera de la variable condición es que el índice donde se escribe no llegue al final del bucle, esto es que sea siempre menor al número de celdas totales menos una unidad, ya que si fuera mayor se pasaría del bucle y produciría un `segmentation fault`.

  Una vez puede entrar, inserta el dato, aumenta el índice y envía la señal a la condición de ocupadas.

- Método extraer

```
int ProdConsLIFO::extraer()
{
   if (primera_libre == 0)
      ocupadas.wait();
   primera_libre--;
   int valor = buffer[primera_libre];
   libres.signal();
   return valor;
}
```

En este caso la condición de espera es la opuesta, que el índice no esté en el 0, puesto que esto significaría que no hay ningún valor en el buffer.

El resto del procedimiento consiste en decrementar el índice, leer el dato, dar la señal a la condición libre y hacer un `return` del dato extraído.

- Funciones hebras

```
void funcion_hebra_productora(MRef<ProdConsLIFO> monitor, int indice)
{
   for (unsigned i = 0; i < num_items; i++)
   {
      int valor = producir_dato(indice);
      monitor->insertar(valor);
   }
}

void funcion_hebra_consumidora(MRef<ProdConsLIFO> monitor, int indice)
{
   for (unsigned i = 0; i < num_items; i++)
   {
      int valor = monitor->extraer();
      consumir_dato(valor, indice);
   }
}
```

Estas funciones son bastante sencillas y se encargan simplemente de lanzar las funciones desde cada hebra.

Versión FIFO

- Monitor

```
class ProdConsFIFO : public HoareMonitor
{
private:
   static const int num_celdas_total = 10;
   int buffer[num_celdas_total];
   int posicion_prod, posicion_cons;
   int num_items;
   CondVar ocupadas, libres;

public:
   ProdConsFIFO();
   void insertar(int dato);
```

```
   int extraer();
};
```

Vemos que la diferencia con respecto a la versión LIFO es la existencia de una nueva variable `num_items` que cuenta el número de items que hay actualmente en el buffer. Además disponemos de dos índices que almacenan la posición de escritura y de lectura por separado.

El constructor inicializa a 0 las variables enteras y prepara las variables condición.

- Método insertar

```
void ProdConsFIFO::insertar(int dato)
{
   if (num_items == num_celdas_total)
      libres.wait();

   buffer[posicion_prod] = dato;
   posicion_prod  = (posicion_prod+1) % num_celdas_total;
   num_items++;
   ocupadas.signal();
}
```

Vemos que tiene algunas pequeñas diferencias pero sigue la misma estructura que la versión LIFO. Ahora la condición de espera es determinada por el número de items en el buffer. En caso de que el número de items sea igual al número de celdas total querrá decir que el buffer está lleno y por tanto no debe insertar más.

A continuación, se inserta el dato y se aumenta el contador en módulo número de celdas total. Finalmente, se envía señal a las ocupadas.

- Método extraer

```
int ProdConsFIFO::extraer()
{
   if (num_items == 0)
      ocupadas.wait();
   int valor = buffer[posicion_cons];
   num_items--;
   posicion_cons = (posicion_cons+1) % num_celdas_total;
   libres.signal();
   return valor;
}
```

Aquí la condición es que no haya items puesto que en ese caso el buffer estaría vacio y leería datos basura. Decrementamos el número de items y aumentamos el contador en módulo tamaño del vector. Finalmente se envía señal a la variable libres.

- Funciones hebra

```
void funcion_hebra_productora(MRef<ProdConsFIFO> monitor, int indice)
{
   for (unsigned i = 0; i < num_items; i++)
   {
      int valor = producir_dato(indice);
      monitor->insertar(valor);
   }
```

```
}

void funcion_hebra_consumidora(MRef<ProdConsFIFO> monitor, int indice)
{
   for (unsigned i = 0; i < num_items; i++)
   {
      int valor = monitor->extraer();
      consumir_dato(valor, indice);
   }
}
```

Las funciones hebras son idénticas a la versión LIFO.

Podemos notar que en los métodos de insertar y extraer no hemos necesitado un bucle `while` y nos ha bastado con usar una sentencia `if`. Esto se debe a que no es un semáforo SC y por tanto, las variables condición no van a ser modificadas por otros procesos mientras espera.