Modificación Lectores - Escritores

Salvador Romero Cortés | Solucionar problema de inanición

Este ejercicio consiste en modificar la implementación original de los lectores-escritores que favorecía a los lectores para evitar la posible inanición de las hebras escritoras. Esto ocurría porque las hebras lectoras podían entrar muchas más veces que las escritoras y en muchos casos podían dejar a las hebras escritoras en la cola de condición por un tiempo indefinido.

Implementación original (en C++)

```
class Lec_Esc : public HoareMonitor
{
private:
   bool escrib; //indica si un escritor esta escribiendo
   int n_lec;       //número de lectores simultaneos
   CondVar lectura,escritura;
public:
   Lec_Esc();
   void ini_lectura();
   void fin_lectura();
   void ini_escritura();
   void fin_escritura();
};

Lec_Esc::Lec_Esc(){
   escrib = false;
   n_lec = 0;
   lectura = newCondVar();
   escritura = newCondVar();
}

void Lec_Esc::ini_lectura(){
   if (escrib)
      lectura.wait();
   n_lec += 1;
   lectura.signal();
}

void Lec_Esc::fin_lectura(){
   n_lec -= 1;
   if (n_lec == 0)
      escritura.signal();
}

void Lec_Esc::ini_escritura(){
   if (n_lec > 0 || escrib)
      escritura.wait();
   escrib = true;
}

void Lec_Esc::fin_escritura(){
   escrib = false;
   if (lectura.empty())
      escritura.signal();
```

```
   else
      lectura.signal();
}
```

Para evitar que las hebras escritoras no accedan a la estructura de datos lo que hacemos es llevar la cuenta de los escritores que quieran entrar a la estructura. Esta cuenta la llevamos consultando el método empty de la clase condición.

De esta manera si vemos que hay hebras escritores esperando en la cola de la condición sólo podrán entrar hasta 3 lectores más. Estos lectores extras tienen su propio contador que será aumentado/decrementado en función de la cola de condición de escritura.

Este contador lo gestiona la hebra lectora cuando entra y cuando sale de la estructura. Cuando entra, si hay escritores esperando quiere decir que es una hebra del conjunto extra (aumentamos el contador). Análogamente cuando sale de la estructura comprueba que hay escritores esperando y se disminuye el contador en caso afirmativo.

Así cuando salgan todas las hebras lectoras, se avisa a los escritores.

Además también añadimos una salida por pantalla para saber cuando un escritor está esperando.

El código final queda como:

```
class Lec_Esc : public HoareMonitor
{
private:
   static const int maximo_lectores = 3;
   bool escrib; //indica si un escritor esta escribiendo
   int n_lec;       //número de lectores simultaneos
   int lectores_extra;
   CondVar lectura,escritura;
public:
   Lec_Esc();
   void ini_lectura();
   void fin_lectura();
   void ini_escritura();
   void fin_escritura();
};

Lec_Esc::Lec_Esc(){
   escrib = false;
   n_lec = 0;
   lectores_extra = 0;
   lectura = newCondVar();
   escritura = newCondVar();
}

void Lec_Esc::ini_lectura(){
   if (escrib || lectores_extra == maximo_lectores)
      lectura.wait();
// si hay escritores esperando, las hebras nuevas son extras
   if (!escritura.empty())
      lectores_extra++;

   n_lec++;
```

```
// si no hay escritores esperando, le da paso a más lectores
   if (escritura.empty())
      lectura.signal();
}

void Lec_Esc::fin_lectura(){
// si hay escritores esperando reducimos el contador de extras
   if (!escritura.empty())
      lectores_extra--;
   n_lec --;
   if (n_lec == 0)
      escritura.signal();
}

void Lec_Esc::ini_escritura(){
   if (n_lec > 0 || escrib){
      cout << "==== Escritor está esperando a entrar ====" << endl << flush;
      escritura.wait();
   }
   escrib = true;
}

void Lec_Esc::fin_escritura(){
   escrib = false;
   if (lectura.empty())
      escritura.signal();
   else
      lectura.signal();

}
```