Problema del “buffer limitato” – II parte

Lorenzo Gallucci

Una vera struttura dati (1/3)

- Obiettivo:
  - contenere le informazioni in arrivo dai produttori, perché i consumatori possano avervi accesso
- Creiamo dunque una struttura dati contenente:
  - Un array di interi per contenere i dati
  - Un intero **in** che indica la prossima posizione da scrivere (quando la struttura è vuota, esso vale -1)
  - Un intero **out** che indica la prossima posizione da leggere (quando la struttura è piena, esso è uguale ad **in**)

Una vera struttura dati (2/3)

- Ad ogni scrittura:
  - Se **in** è < 0 (struttura vuota), sia in che out vengono inizializzati a zero
  - Nella posizione occupata da **in** viene inserito il valore da scrivere
  - **in** viene incrementato
  - Se **in** ha raggiunto la lunghezza dell'array, viene riportato a zero
- Una scrittura è consentita se:
  - **in != out** (c'è ancora spazio)

Una vera struttura dati (3/3)

- Ad ogni lettura:
  - Il valore puntato da **out** viene preso come risultato
  - **out** viene incrementato
  - Se **out** ha raggiunto la dimensione dell'array, viene riportato a zero
  - Se **out** è diventato pari ad **in** (la struttura si è svuotata), ad **in** viene assegnato -1
- Una lettura è consentita se:
  - **in >= 0** (la struttura non è vuota)

Spazio (1/3)

```
package bufferlimitato.vero;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Spazio
{
   Lock      lock;

   Condition nonPieno;

   Condition nonVuoto;

   int[]     buffer;

   /**
    * Indica la prossima posizione da scrivere (all'inizio vale -1)
    */
   int       in;

   /**
    * Indica l'ultima posizione letta (all'inizio vale 0)
    */
   int       out;
```

Spazio (2/3)

```
public Spazio(int spazioTotale)
{
   buffer = new int[spazioTotale];
   in = -1;
   out = 0;
   lock = new ReentrantLock(true);
   nonPieno = lock.newCondition();
   nonVuoto = lock.newCondition();
}

public int leggi()
{
   int risultato = buffer[out++];
   if (out >= buffer.length)
   {
      out = 0;
   }
   if (in == out)
   {
      in = -1;
   }
   return risultato;
}
```

Spazio (3/3)

```
    public boolean letturaConsentita()
    {
        return in >= 0;
    }

    public boolean scritturaConsentita()
    {
        return in != out;
    }

    public void scrivi(int valore)
    {
        if (in < 0)
        {
            in = 0;
            out = 0;
        }
        buffer[in++] = valore;
        if (in >= buffer.length)
        {
            in = 0;
        }
    }
}
```

Produttore/consumatore modificato

- Il produttore genera una sequenza di numeri interi, uno per ogni ciclo
  - Un intero indica il prossimo numero da produrre e passare al metodo scrivi()
- I consumatori visualizzano le informazioni lette
  - … si tratta dei valori restituiti dal metodo leggi()

Consumatore (1/2)

```
package bufferlimitato.vero;

public class Consumatore extends Utente
{
   private final int num;

   public Consumatore(int num, Spazio spazio, int cicli)
   {
      super(spazio, cicli);
      this.num = num;
   }

   protected void esegui() throws InterruptedException
   {
      spazio.lock.lock();
      try
      {
         while (!spazio.letturaConsentita())
            spazio.nonVuoto.await();
         System.out.println("Consumatore " + num + ": leggo");
         attendi();
```

Consumatore (2/2)

```
            int valoreLetto = spazio.leggi();
                System.out.println("Consumatore " + num
    + ": lettura terminata, valore = " + valoreLetto);
                spazio.nonPieno.signalAll();
            }
            finally
            {
                spazio.lock.unlock();
            }
        }
}
```

Produttore (1/2)

```
package bufferlimitato.vero;

public class Produttore extends Utente
{
   private final int num;

   private int      valore;

   public Produttore(int num, Spazio spazio, int cicli)
   {
      super(spazio, cicli);
      this.num = num;
      valore = 0;
   }
```

Produttore (2/2)

```
    protected void esegui() throws InterruptedException
    {
        spazio.lock.lock();
        try
        {
            while (!spazio.scritturaConsentita())
                spazio.nonPieno.await();
            System.out.println("Produttore " + num + ": scrivo");
            attendi();
            int valoreProdotto = valore++;
            spazio.scrivi(valoreProdotto);
            System.out.println("Produttore " + num + ": scrittura terminata, valore prodotto
     = " + valoreProdotto);
            spazio.nonVuoto.signalAll();
        }
        finally
        {
            spazio.lock.unlock();
        }
    }
}
```