Problema del “buffer limitato”

Lorenzo Gallucci

Buffer limitato

- Immaginiamo una struttura dati che contiene delle informazioni
- La struttura può, in ogni momento, avere dello spazio libero oppure no
- Se vi è dello spazio libero, può essere **prodotto** un nuovo elemento
- Se lo spazio libero è inferiore al massimo, almeno un elemento può essere **consumato**

Buffer limitato

- Esempio di struttura:

```
package bufferlimitato;

public class Spazio
{
    public int spazioTotale;
    public int spazioLibero;

    public Spazio(int spazioTotale)
    {
        this.spazioTotale = spazioTotale;
        spazioLibero = spazioTotale;
    }

    public boolean letturaConsentita()   {   return spazioLibero < spazioTotale;    }

    public boolean scritturaConsentita()     {      return spazioLibero > 0;    }

    public void leggi()     {        spazioLibero++;    }

    public void scrivi()     {        spazioLibero--;    }
}
```

Buffer limitato

- Problema: come coordinare più thread che accedono alla medesima struttura, ma con diverse modalità d'accesso?
  - → classico problema produttore/consumatore
- Uno o più thread "scrivono" le informazioni
- Uno o più thread "leggono" le informazioni

Buffer limitato (con synchronized)

- Lo stesso oggetto di classe "Spazio" funge da sincronizzatore
  - Ogni oggetto deriva da Object, che ha funzioni di sincronizzazione
- Su di esso si può invocare *synchronized* () (per assicurare la mutua esclusione nell'accesso)
- Se determinate condizioni richieste non sono soddisfatte, si può invocare wait()
  - Wait() rilascia l'accesso in mutua esclusione a favore di altri thread
- L'avvenuto soddisfacimento di determinate condizioni si può segnalare, ad altri thread, con *notify()* e *notifyAll()*
  - Al termine del blocco sincronizzato, un thread potrà tentare l'accesso (potranno seguirne altri se si è usato notifyAll())

Buffer limitato (con synchronized)

- Classe che incapsula un generico utente dello spazio condiviso:

```
package bufferlimitato;

import java.util.Random;

public abstract class Utente implements Runnable
{
    protected Spazio spazio;
    private int      cicli;

    protected Utente(Spazio spazio, int cicli)
    {        this.spazio = spazio;        this.cicli = cicli;    }

    public final void run()
    {
        try
        {          for (int i = 0; i < cicli; i++)           {             esegui();          }       }
        catch (InterruptedException e)
        {            System.out.println("Non posso attendere");            e.printStackTrace();        }
    }

    protected abstract void esegui() throws InterruptedException;
    protected void attendi() throws InterruptedException
    {
        final int tempo = new Random().nextInt(250);
      System.out.print("Attendo " + tempo + " ms ... "); Thread.sleep(tempo); System.out.println(" ok.");
    }
}
```

Buffer limitato (con synchronized)

- Esempio di produttore:

```
package bufferlimitato;

public class Produttore extends Utente
{
   private final int num;

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

   protected void esegui() throws InterruptedException
   {
      synchronized (spazio)
      {
         while (!spazio.scritturaConsentita())
         {               spazio.wait();           }
         System.out.println("Produttore " + num + ": scrivo");
         attendi();
         System.out.println("Produttore " + num + ": scrittura terminata");
         spazio.scrivi();
         spazio.notifyAll();
      }
   }
}
```

Buffer limitato (con synchronized)

- Esempio di consumatore:

```
package bufferlimitato;

public class Consumatore extends Utente
{
   private int num;

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

   protected void esegui() throws InterruptedException
   {
      synchronized (spazio)
      {
         while (!spazio.letturaConsentita())
         {               spazio.wait();           }
         System.out.println("Consumatore " + num + ": leggo");
         attendi();
         System.out.println("Consumatore " + num + ": lettura terminata");
         spazio.leggi();
         spazio.notifyAll();
      }
   }
}
```

Buffer limitato (con synchronized)

- Esempio d'uso:

```
package bufferlimitato;

public class EsempioProduttoreConsumatore
{
   public static void main(String[] args)
   {
      Spazio spazio = new Spazio(1);
      Utente consumatore1 = new Consumatore(1, spazio, 5);
      Utente consumatore2 = new Consumatore(2, spazio, 5);
      Produttore produttore = new Produttore(1, spazio, 10);
      Thread threadConsumatore1 = new Thread(consumatore1);
      Thread threadConsumatore2 = new Thread(consumatore2);
      Thread threadProduttore = new Thread(produttore);
      threadConsumatore1.start();
      threadConsumatore2.start();
      threadProduttore.start();
      try
      {
         threadConsumatore1.join();
         threadConsumatore2.join();
         threadProduttore.join();
      }
      catch (InterruptedException e)
      {
         System.out.println("Non posso attendere la conclusione dei thread");
         e.printStackTrace();
      }
   }
}
```

Buffer limitato (con Lock)

- Nell'esempio precedente, sia i produttori che i consumatori effettuavano le medesime operazioni per addormentarsi in attesa di un evento (wait()) o svegliare altri thread (notify(), notifyAll())

- Vi è una sola coda di attesa per ogni Object: produttori e consumatori vengono accodati sulla medesima coda

Buffer limitato (con Lock)

- Lock consente di creare code separate ("Condition"), a seconda di quale condizione viene considerata
  - I produttori in attesa di spazio disponibile potrebbero disporre di una coda propria
  - I consumatori in attesa di informazioni potrebbero accodarsi su una propria coda
- Gli oggetti Condition non esimono però dal controllo della condizione vera e propria
  - È sempre necessario un ciclo while !

Buffer limitato (con Lock)

- Spazio con utilizzo dei Lock:

```
package bufferlimitato.lock;

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

import bufferlimitato.Spazio;

public class SpazioConLock extends Spazio
{
   Lock      lock;

   Condition nonPieno;

   Condition nonVuoto;

   public SpazioConLock(int spazioTotale)
   {
      super(spazioTotale);
      lock = new ReentrantLock(true);
      nonPieno = lock.newCondition();
      nonVuoto = lock.newCondition();
   }
}
```

Buffer limitato (con Lock)

- La classe Utente resta pressochè identica:

```
package bufferlimitato.lock;

import java.util.Random;

public abstract class Utente implements Runnable
{
   protected SpazioConLock spazio;
   private int      cicli;

   protected Utente(SpazioConLock spazio, int cicli)
   {
      this.spazio = spazio;       this.cicli = cicli;
   }

   public final void run()
   {
      try
      {          for (int i = 0; i < cicli; i++)          {              esegui();          }       }
      catch (InterruptedException e)
      {           System.out.println("Non posso attendere");          e.printStackTrace();       }
   }

   protected abstract void esegui() throws InterruptedException;

   protected void attendi() throws InterruptedException
   {
      final int tempo = new Random().nextInt(250);
      System.out.print("Attendo " + tempo + " ms ... ");       Thread.sleep(tempo);       System.out.println(" attesa
      terminata.");
   }
}
```

Buffer limitato (con Lock)

- Nuovo produttore:

```
package bufferlimitato.lock;

public class Produttore extends Utente
{
   private final int num;

   public Produttore(int num, SpazioConLock spazio, int cicli)
   {
      super(spazio, cicli);        this.num = num;
   }

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

Buffer limitato (con Lock)

- Nuovo consumatore:

```
package bufferlimitato.lock;

public class Consumatore extends Utente
{
   private final int num;

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