Steve Blackbird (TardoFreak)

PC E LINEE SERIALI CON JAVA

7 March 2011

Introduzione

Fino a qualche anno addietro si utilizzavano le **linee seriali RS232** per collegare apparecchiature al PC. Il caso tipico era il programmatore di EPROM o di MCU collegato tramite il cavo seriale (ne ho ancora un paio nell' armadio) o il modem analogico a 56K per collegarsi ad internet tramite la linea telefonica. Ora questa linea di comunicazione è oramai obsoleta ed è stata sostituita dall' **USB** che ha in qualche modo monopolizzato la connessione di periferiche di ogni tipo con i PC. Anche i **microcontrollori** si sono adeguati e molti di quelli in commercio hanno la possibilita' di connettersi e comunicare con un PC tramite l' USB.

Esistono anche in commercio circuiti di conversione da USB a RS232 proprio per poter connettere un microcontrollore sprovvisto di USB ma munito di **UART** (difficile trovare un micro senza interfaccia di comunicazione seriale) ad un PC per gli usi più disparati. In questo momento sto sviluppando un' apparecchiatura da connettere al PC tramite USB e che emula la porta seriale come fanno i vari FT232 o MCP2200 della Microchip. Quest' ultimo utilizza un driver standard di Windows chiamato usbser.sys che è quello che uso anch' io per la mia applicazione.

Premetto che, se nella programmazione dei microcontrollori posso dire la mia, non sono per niente bravo a fare programmi su PC. In ogni caso ho dovuto trovare un modo per gestire convenientemente una linea seriale tramite un programma scritto su un PC. La comunicazione seriale è anche stata oggetto di diverse discussioni nel forum di EY ed ho notato che c' è anche parecchio interesse per l' argomento quindi ho deciso di scrivere questo modesto articolo per tentare di fornire qualche indicazione.

Java

I programmi che ho scritto per i PC li ho scritti in passato utilizzando il Turbo Pascal ed il Turbo C della Borland e la gestione delle seriali era molto semplice: si aprivano e si utilizzavano come semplici files di testo leggendo e scrivendo. Inutile dire che li ho scritti nel neaolitico ed oggi utilizzare questi linguaggi non ha più senso. Il massimo a cui sono arrivato è stato il Visual Basic 6, anch' esso caduto in disuso e soppiantato dalle nuove versione del 2005, 2008 e 2010.
A quasi 50 anni l' idea di dover imparare una altro linguaggio per scrivere i

programmi sul PC non mi proccupava, semplicemente mi terrorizzava, quindi ho cercato un linguaggio che mi permettesse, una volta imparato, di vivere di rendita per un po' di anni. La scelta è caduta sul linguaggio **Java** per una serie di motivi:

- è un bel linguaggio ad oggetti e la sintassi assomiglia molto a C (per non dire che è quasi identica) e quindi per me semplice da imparare.
- può funzionare su diverse piattaforme. Evito cosi' la rogna di dover imparare un linguaggio (o dialetto se vogliamo) per ogni piattaforma. Java funziona bene sul PC con Windows, Linux e sul MAC.
- il sistema di sviluppo NetBeans è ottimo, potente, facile da utilizzare e completamente gratuito
- ci sono parecchi tutorial in rete che ne rendono facile l' apprendimento di Java
- le linee seriali si gestiscono con facilità tramite il pacchetto RxTx
- a quanto pare è un linguaggio di moda e probabilmente (almeno lo spero) sarà utilizzato ancora per molti anni anche perché, con Java, si scrivono le applicazioni per i telefonini.

Per provare le comunicazioni seriali ho realizzato da me un programma di emulazione di terminale come l' HyperTerminal. Essere in grado di fare un programma di terminale vuol dire essere poi in grado di fare qualsiasi altro programma che scriva e legga in una linea seriale. In effetti se si riesce ad inviare un carattere inviare una stringa viene di conseguenza, lo stesso per la ricezione. A dire il vero ho provato anche con Visual Basic ma non sono riuscito ad ottenere risultati soddisfacenti, forse perché non è il mio campo e quindi l' imbranataggine mi perseguita. Resta comunque il fatto che con Java ci sono riuscito e, devo ammetterlo, senza troppi sforzi.

NetBeans

E' il sistema di sviluppo che ho utilizzato dapprima per imparare il Java e poi per scrivere questo ed altri programmi. E' molto potente e ben fatto anche se, qualche volta, carente di documentazione. Ha la possibilità di progettare un' interfaccia grafica in modo interattivo e lo fa piuttosto bene. Per il programma di terminale non ho voluto utilizzare l' interfaccia grafica costruita da NetBeans un po' per imparare il posizionamento manuale dei controlli, e un po' per fare in modo che il sorgente potesse essere indipendente dal sistema di sviluppo. Parecchia documentazione e i preziosissimi "HowTo" si trovano nel sito della Oracle dal quale si scarica sia NetBeans che il Java Developement Kit.

Installazione

Il Java non è un linguaggio compilato ma utilizza un codice intermedio, quindi per poter fare funzionare programmi è necessario avere installata la macchina virtuale

che, chiaramente, cambia da piattaforma a piattaforma. Probabilmente, visto che in EY si usa il programma FidoCadJ per disegnare circuiti elettrici da inserire nei commenti, è più che probabile che i frequentatori dei forum abbiano già installata la macchina virtuale. Senza di questa FidoCadJ non funzionerebbe. La macchina virtuale e' il cosiddetto JRE (acronimo che sta per Java Runtime Enviroment) ma per sviluppre programmi c' è bisogno del sistema di sviluppo chiamato JDK (Java developement Kit). Oggi la Oracle fornisce un pacchetto unico comprendente JRE, JDK ed il sistema di sviluppo NetBeans che si possono scaricare a **questo indirizzo** ed installare insieme in un sol colpo.
Quindi si scarica il file, lo si apre ed il gioco è fatto.
Tuttavia, per poter utilizzare le porte seriali è necessario installare il pacchetto **RxTx**.

Il pacchetto RxTx

Inzialmente la Sun (oggi è tutto in mano alla Oracle) forniva un pacchetto chiamato javaxcomm. Gli utenti preferivano invece il pacchetto RxTx (sviluppato altrove sotto licenza GNU) fino al punto che Oracle non supporta più javaxcomm perchè non c' era abbastanza interesse da parte dell' utenza e quindi è finito nel dimenticatoio. RxTx (**rxtx-2.1-7-bins-r2.zip**) è disponibile a **questo link**
Una volta scaricato il pacchetto ed aperto, all' interno troviamo la cartella **rxtx-2.1-7-bins-r2**. Spostiamo quindi la cartella sul desktop e facciamo le seguenti operazioni supponendo che Java sia installato (come fa in automatico il programma d' installazione descritto prima) in **c:\Programmi\Java**.

- Copiare **rxtxSerial.dll**
  che si trova nella cartella **rxtx-2.1-7-bins-r2\windows\i368-mingw32**
  in **c:\Programmi\Java\jre1.6.0_23\bin**
- Copiare **RXTXcomm.jar**
  che si trova nella cartella **rxtx-2.1-7-bins-r2**
  in **c:\Programmi\Java\jre1.6.0_23\lib\ext**
- Copiare **rxtxSerial.dll**
  che si trova nella cartella **rxtx-2.1-7-bins-r2\windows\i368-mingw32**
  in **c:\Programmi\Java\jdk1.6.0_23\jre\bin**
- Copiare **RXTXcomm.jar**
  che si trova nella cartella **rxtx-2.1-7-bins-r2**
  in **c:\Program Files\Java\jdk1.6.0_23\jre\lib\ext**

Il pacchetto RxTx contiene anche il necessario non solo per la gestione delle porte seriali ma anche per quelle parallele. Se vogliamo installare anche quelle (abbiamo fatto 30, facciamo 31!) bisogna:

- Copiare **rxtxParallel.dll**
  che si trova nella cartella **rxtx-2.1-7-bins-r2\windows\i368-mingw32**
  in **c:\Programmi\Java\jre1.6.0_23\bin**

- Copiare **rxtxParallel.dll**
  che si trova nella cartella **rxtx-2.1-7-bins-r2\windows\i368-mingw32**
  in **c:\Programmi\Java\jdk1.6.0_23\jre\bin**

Ora abbiamo tutto il necessario per sbizzarrirci con le porte di comunicazione seriali e parallele.

Il programma JavaTerm

Per scrivere questo programma ho ricercato in giro per la rete alcuni esempi di utilizzo. Dopo diversi tentativi sono riuscito a scrivere il software necessario e quindi lo illustro qui di seguito. Le parti di questo programma interessanti per l' utilizzo delle seriali sono:

- L' acquisizione della lista delle porte disponibili tramite il metodo **portList**
- La connessione alla porta selezionata tramite il metodo **portInit**
- Come inviare i caratteri con il metodo **output.write** dell' OutpuStream output
- Il gestore dell' evento di ricezione dei caratteri **serialEvent**
- Il metodo per chiudere la porta **portClose**

portList()

All' inizio del programma richiamo il metodo **portList()** che mi scandisce le porte seriali del computer e mi carica i nomi delle porte che ha trovato su un combo-box, un controllo per la selezione a tendina della porta.

```
/**
 * Questa serve per caricare il combo box con la lista delle porte di comunicazione
 * disponibili.
 */
private void portList()
{
  cmbPort.removeAllItems();
  Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();

  // Scandisce tutte le porte presenti per caricare il combo
  while (portEnum.hasMoreElements())
  {
    CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();
    cmbPort.addItem(currPortId.getName());
  }
}
```

La parte più importante è, ovviamente, il **while** che scandisce tutte le porte per averne il nome. Questo nome verrà poi utilizzato per aprire la porta selezionata.

portInit(String comPort)

La connessione alla porta selezionata avviene mediante la pressione del pulsante "Connect" che recupera la stringa che indica la porta a cui connettersi dal combo-box e chiama il metodo **portInit(String comPort)**.

```
/**
 * Inizializzazione della porta di comunicazione.
 */
public boolean portInit(String comPort)
{
  boolean connesso;

  CommPortIdentifier portId = null;
  Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();
  connesso = false;

  // esegue l' iterazione cercando la porta
  while (portEnum.hasMoreElements())
  {
    CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();
    if (currPortId.getName().equals(comPort))
    {
      portId = currPortId;
      connesso = true;
      break;
    }
  }

  if (portId == null)
  {
    JOptionPane.showMessageDialog(null, "Can't find "+comPort);
    return(false);
  }

  try
  {
    // open serial port, and use class name for the appName.
    serialPort = (SerialPort) portId.open(this.getClass().getName(),TIME_OUT);
```

```
      // set port parameters
      serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_RTSCTS_IN | SerialPort.FLOWC
      serialPort.setSerialPortParams(DATA_RATE,
          SerialPort.DATABITS_8,
          SerialPort.STOPBITS_1,
          SerialPort.PARITY_NONE);

      // open the streams
      input = serialPort.getInputStream();
      output = serialPort.getOutputStream();

      // add event listeners
      serialPort.addEventListener((SerialPortEventListener) this);
      serialPort.notifyOnDataAvailable(true);
    }
    catch (Exception e)
    {
      JOptionPane.showMessageDialog(null, "Connection Error: "+ e.toString());
      connesso = false;

    }
    return(connesso);
  }
```

La prima parte di questo metodo ricorda la lista delle porte. Infatti anche qui viene cercata la porta di comunicazione indicata dalla stringa **comPort**. Se non viene trovata il metodo esce immediatamente e ritorna un valore false.
La parte sotto il **try** ed il **catch** inizializza ed apre la porta. La velocità di comunicazione e la timeout sono stati assegnati durante la dichiarazione:

```
  /** Milliseconds to block while waiting for port open */
  private static final int TIME_OUT = 2000;

  /** Default bits per second for COM port. */
  private static final int DATA_RATE = 921600;
```

e, come gli altri parametri, si possono variare a seconda del bisogno o implementando un' interfaccia grafica interattiva. A me questo non interessava quindi non l' ho implementato anche per cercare di rendere il programma più semplice possibile in modo da crearmi uno scheletro da riutilizzare per le varie applicazioni.

serialEvent(SerialPortEvent oEvent)

Il gestore di eventi **serialEvent(SerialPortEvent oEvent)** è quello che riceve i caratteri che, tramite qualche pastrocchio magari non molto ortodosso (non fucilatemi, vi prego) viene appeso al testo della JTextArea **txtRx** che funge da schermo del terminale.

```
/**
 * Gestisce l' evento della porta seriale. Riceve il carattere e lo visualizza
 */
 public synchronized void serialEvent(SerialPortEvent oEvent)
 {
   int i,l,available;
   String s,s2;
   byte chunk[];

   if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE)
   {
     try
     {
       available = input.available();
       chunk = new byte[available];
       input.read(chunk, 0, available);
       for (i=0;i<available;i++)
       {
         if(chunk[i]==8)  // processa il carattere di backspace
         {
           s = txtRx.getText();
           l = s.length()-1;
           if(l > -1)
           {
             s2 = s.substring(0,l);
             txtRx.setText(s2);
           }
         }
         else
         {
           s = String.valueOf((char)chunk[i]);
           txtRx.append(s);
         }
         txtRx.setCaretPosition(txtRx.getText().length());
       }
     }
```

```
      catch (Exception e)
      {
        JOptionPane.showMessageDialog(null, "RX error: "+ e.toString());
      }
    }
  }
```

L' array di bytes **chunk** contiene tutti i bytes disponibili nel buffer d' ingresso della porta seriale. Per poter riconoscere il tasto di backspace e fargli eliminare l' ultimo carattere dall' area di testo ho dovuto scandire tutti i caratteri ricevuti. Quindi quella parte potrebbe benissimo essere semplificata in un' applicazione che non usa la visualizzazione dei caratteri ricevuti ma li tratta in altro modo.

L' invio del carattere premuto

Il carattere premuto arriva dal gestore di eventi **keyPressed(KeyEvent e)** generato, appunto, dalla TextArea ed inviato sulla linea seriale semplicemente tramite il metodo **output.write(ch)**.

```
  /**
   * Gestore dell' evento keyPressed
   * @param e
   */
  public void keyPressed(KeyEvent e)
  {
    char ch;

    if (e.getSource() == txtRx)
    {
      ch = e.getKeyChar();
      if (ch=='\n') ch = '\r';
      if(ch!= 65535)
      {
        try
        {
          output.write(ch);
        }
        catch (IOException ex)
        {
          JOptionPane.showMessageDialog(null, "TX error: "+ ex.toString());
        }
      }
      e.consume();
```

```
    }
  }
```

L' istruzione **if(ch!= 65535)** è una di quelle classiche porcherie che però funzionano e mi permette di filtrare i tasti che non fanno parte del terminale. Essendo questo programma un programma di prova l' ho lasciato così. Se però qualcuno ha una soluzione migliore è gentilmente pregato di inserirla in un commento in modo da eliminare quest' obbrobrio.

output.write

output è un oggetto della classe **OutputStream** la cui dichiarazione è:

```
/** The output stream to the port */
  private OutputStream output;
```

E quindi **write** è il metodo che bisogna richiamare per inviare un carattere o una stringa di caratteri. C'e' poco da dire su questo metodo, se voglio inviare la stringa **s** sulla linea seriale basta scrivere **output.write(s)**, se voglio inviare un carattere di ritorno a capo scrivo **output.write("\r\n")** e se voglio inviare il carattere **ch** scrivo **output.write(ch)**

portClose()

Questo metodo serve, come si può facilmente immaginare, per chiudere la porta di comunicazione. E' bene assicurarsi che sia richiamato quando il programma termina per evitare problemi con Linux. Se devo eseere sincero non saprei quali problemi ci potrebbero essere con Linux, ma il programma di demo da cui ho ricavato il mio dice questo. Io, per sicurezza, ho riportato l' avvertenza.

```
  /**
  * Questa deve essere chiamata quando si finisce di utilizzare la porta
  * Previene il bloccaggio della porta in Linux
  */
  public synchronized void portClose()
  {
    if (serialPort != null)
    {
      serialPort.removeEventListener();
      serialPort.close();
    }
  }
```

Il resto del programma

Ho fatto in modo che lo schermo del terminale non sia editabile se la porta non è connessa. Per cancellare le scritte dallo schermo del terminale si preme il pulsante "Clear Screen". Una volta che la porta è connessa il pulsante "Connect" cambia scritta e diventa "Disconnect". A quel punto il terminale sta funzionando. Alcune considerazioni:

- Il programma in questione è stato pensato per funzionare in congiunzione con porte seriali virtuali emulate da un dispositivo USB. Proprio per questo ho tralasciato tutta la parte di setup della velocità e degli altri parametri, parte che deve essere implementata se si vuole utilizzare il programma con gli adattatori USB-RS232 come FT232 o similari. Ho quindi selezionato 8 bit di dati, nessuna parità, controllo di flusso hardware per non perdere caratteri (viene gestito dall' emulatore di seriale).
- La cosa che mi interessava maggiormente era l' handshake della porta. E' infatti estremamente importante che non si perdano caratteri per strada. Il pacchetto RxTx svolge bene questo lavoro. Tanto mi basta per le mie applicazioni.
- Gli eventuali errori di comunicazione e/o di connessione li ho visualizzati tramite delle finestre di messaggio. Un programma serio dovrebbe provvedere alla buona gestione degli eventuali errori. Nel caso di un errore sull' apertura della porta mi sono limitato a riportare il messaggio di errore sulla finestra di messaggio.

Quando lo schermo diventa troppo "affollato" di caratteri lo si può cancellare premendo il pulsante **Clear Screen**.

Conclusioni e ringraziamenti

Innanzi tutto è mio dovere ringraziare tutti i forumisti che mi hanno seguito, aiutato a capire e saggiamente consigliato. Senza il loro aiuto non sarei riuscito ad ottenere questo, seppur piccolo, risultato.
Il programma ha ancora un baco che non sono riuscito a risolvere: quando utilizzo il tasto di Backspace (nella tastiera è il tasto con la freccia **<-**) e questo va in auto-ripetizione, il programma si blocca. Qualcuno l' ha provato e mi ha detto che a lui non succede quindi non so cosa dire. Sappiate che mi da questo problema, se poi è solo un mio problema sono anche più contento.
Forse queste informazioni potranno essere di aiuto a qualcuno o forse no, quello che mi auguro è di aver gettato un sasso nello stagno ed aver stimolato qualche programmatore di PC vero, non farlocco come il sottoscritto, a scrivere un articolo di questo tipo per la gestione delle linee seriali con altri linguaggi. **Le comunicazioni fra PC e circuiti con microcontrollori permettono di espandere le possibilità**

di entrambi. Il PC può così diventare, per esempio, la parte di visualizzazione di un data logger realizzato con un microcontrollore, ed un microcontrollore può diventare un' interfaccia hardware per fare dare la possibilità ad un programma su PC di accendere o spegnere apparecchiature elettriche, misurare tensioni, rilevare temperature e quant' altro.

Downloads

Due sono i files allegati a questo articolo. Il primo è **TFJavaTerm.rar** ed è il progetto completo per NetBeans. Per aprirlo con NetBeans, dopo averlo decompresso, basta selezionarne la cartella. Il secondo file è **TFtermMain.rar** che contiene l' unico file del terminale chiamato Main.java per chi sviluppa Java con altri sistemi.

RxTx

12/06/2014 Essendo passato a windows 8.1 ho provato ad utilizzare il vecchio RxTx. Oggi ho modificato il link per il download. Tuttavia per le macchine a 64 bit il vecchio RxTx non va bene, non funziona. Ho trovato delle versioni ricompilate per windows e linux (che funzionano benissimo) e le ho messe a disposizione in **questa pagina** per il download.