Comunicação Serial Utilizando a API da SUN

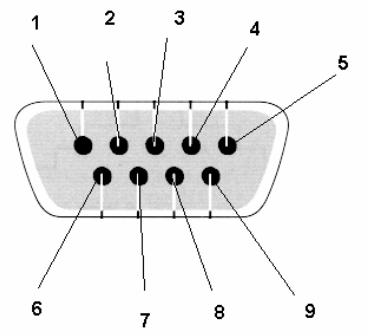
Daniel Vasconcelos Gomes

Uma introdução a programação serial em Java utilizando a API de comunicação serial da SUN.

Introdução

Programar uma rotina em Assembly para comunicar dados de um teclado até um motor de passo é uma tarefa bastante complicada, mas exemplifica o uso de comunicação serial. É complicado, porque, Assembly não é uma linguagem muito acessível. E também, porque é preciso lidar com máscaras de interrupção, sincronizar clock, programar uma porta paralela, reconhecer stop bits, os dados, paridade e ainda lidar com o código ASCII gerado pelo teclado. Com o padrão atual de microprocessadores para PCs, não é preciso esse trabalho, basta usar os recursos de alto nível. Para WIN32 temos as opções das APIs e Visual Basic. E a melhor delas todas, por ser um meio termo, é a API Java Communications da Sun. A máquina virtual Java faz o trabalho sujo para você.

Veja abaixo a pinagem de um conector DB9:



Pinagem do conector DB9

1- DCC (Sinal de controle)
2- Dados recebidos
3- Dados transmitidos
4- DTR (Terminal de dados pronto) Sinal de controle
5- Terra
6- DSR (Sinal de controle)
7- RTS(Requisição para enviar) Sinal de controle
8- CTS (Pronto para enviar) Sinal de controle
9- Indicador de Anel

A tensão lida entre os terminais 3 e 5 fornece o sinal de saída/transmitido. O sinal lido entre 2 e 5 nos fornece o sinal recebido. Os outros pinos nos fornecem sinais de controle (todos acessíveis através da API da SUN é claro) conhecidos como “handshaking”.

USB?

No futuro, as portas seriais e os protocolos de comunicação serial serão substituídos pelo padrão USB ("Universal Serial Bus). Há um link bem interessante com detalhes de como ampliar a API de comunicação Java para que ela possa suportar a USB:

http://www.syngress.com/book_catalog/177_lego_JAVA/sample.htm

O Padrão EIA-232

Também incorretamente chamado de "RS-232", o padrão EIA-232 específica o meio físico de comunicação serial. Acrescido ao número, há uma letra que indica a revisão atual do padrão. Se não me engano, a versão atual é EIA-232-F. O nível de tensão varia de -12 Volts à -3 Volts para o nível lógico 1 e de +3 Volts à +12 Volts para nível lógico 0.

Como meu objetivo é mostrar o uso da API, não vou tentar descrever em maiores detalhes o padrão. Só devo deixar claro, que o programador não precisa converter os dados seriais para paralelo diretamente no código. Isso quem faz é a UART do PC.

Algumas Definições:

DTE: (Data terminal equipament), o equipamento que envia os dados .
DCE: (Data communication equipament) o equipamento que o recebe,
Baudrate: é taxa de bits por segundo que é transmitida. Assim, quando falarmos em um baudrate de 9600 bps, queremos dizer 9600 bits por segundo, ou seja, cada bit dura 1/9600 segundos.
Timeout: é o tempo que a CPU espera para obter uma resposta do DCE. Caso esse tempo seja excedido, a comunicação não é estabelecida.
Parity : Paridade. Os bits de paridade são bits de controle. Paridade par significa que o total de níveis "1" lógicos transmitidos juntamente com a paridade associada deve ser um número par.

A API de Comunicação da SUN

A SUN fornece como download gratuito a API de comunicação serial e paralela na URL:

http://java.sun.com/products/javacomm/index.jsp

Basta baixar a API e realizar os procedimentos de instalação. Após baixar a API, descompactá-la, você terá:

- Copiar o arquivo win32com.dll para o diretório C:\JavaSDK\BIN (isto é, o diretório onde o J2SDK foi instalado no seu PC).
- Copiar o arquivo comm.jar para o diretório C:\JavaSDK\BIN\LIB.
- Copiar o arquivo javax.comm.properties para o diretório C:\JavaSDK\BIN\LIB.
- Em seguida configure o CLASSPATH para que ele reconheça o arquivo comm.jar.

Reconhecendo As Portas

Antes de iniciarmos a comunicação com a porta serial, precisamos reconhecer as portas existentes em sua estação de trabalho. A API de comunicação nos fornece o método getPortIdentifiers() integrante da classe CommPortIdentifier que retorna em uma estrutura Enumeration, as portas disponíveis. A Classe CommPortIdentifier pode ser instanciada e representar uma porta. Para isso, precisamos varrer a

estrutura retornada por getPortIdentifiers() e instanciando cada porta através de uma conversão (casting) simples:

```
//....
Enumeration listaDePortas;
listaDePortas = CommPortIdentifier.getPortIdentifiers();
//...
//..
int i = 0;
         portas = new String[10];
         while (listaDePortas.hasMoreElements()) {
              CommPortIdentifier ips =
              (CommPortIdentifier)listaDePortas.nextElement();
              portas[i] =  ips.getName();
              i++;
         }
//..
```

O método hasMoreElements() retorna o próximo elemento da estrutura listaDePortas, mas o loop while garante que todos os elementos sejam passados ao Array portas através do método getName().

Abrindo portas para comunicar dados

O método getPortIdentifier(String porta) da classe CommPortIdentifier retorna um identificador da porta escolhida. Precisamos instanciar um objeto para receber esse identificador:

```
CommIdentifier cp = CommPortIdentifier.getPortIdentifier(minhaPortaEscolhida);
```

Em seguida criamos uma instância da classe SerialPort utilizando o identificador. Note que uma conversão deverá ser feita. A porta só pode ser instanciada através desse “casting” e ao mesmo tempo abrimos a porta para comunicação:

```
SerialPort porta = (SerialPort)cp.open("SComm",timeout);
```

O método open() tem como parâmetros o nome da classe principal (faça isso para não gerar conflitos) e o valor desejado para timeout. Em seguida, precisamos atribuir fluxos de entrada e saída. Basta utilizar as classes Abstratas OutputStream e InputStream, já que a classe SerialPort implementa os métodos de entrada e saída dessas classes para comunicação serial. Para ler dados na porta serial:

```
InputStream entrada = porta.getInputStream();
```

E para escrever dados na porta serial:

```
OutputStream saida = porta.getOutputStream();
```

Em seguida precisamos configurar os parâmetros de comunicação serial, para isso utilizamos o método setSerialPortParams:

```
porta.setSerialPortParams(baudrate, porta.DATABITS_8, porta.STOPBITS_2, porta.PARITY_NONE);
```

Enviando bytes para a porta serial

Depois de configurar a porta para comunicar e definido o fluxo de saída, podemos comunicar os dados. Isso é bem simples:

```
String msg = “Olá Mundo!”;
saida.write(msg.getBytes());
Thread.sleep(100);
saida.flush();
```

Recebendo dados na porta serial

Até agora tudo foi relativamente simples, e então entramos na área mais complicada: ler dados! A API de comunicações Java facilita bastante o trabalho, mas mesmo assim, são várias linhas de código. Basicamente o que deve ser feito é:

- Criar um fluxo de entrada.
- Adicionar um gerenciador de eventos para dados na porta serial.
- Instanciar uma Thread para aguardar os eventos
- Tratar o evento e receber os dados

A etapa i. já foi detalhada. Para adicionarmos um gerenciador de eventos para a porta serial basta fazer:

```
porta.addEventListener(this);
```

Em seguida precisamos notificar o objeto porta criado de que podem exister dados para serem lidos:

```
porta.notifyOnDataAvailable(true);
```

Agora falta apenas tratar o evento. Infelizmente, essa é a parte mais complicada. Primeiro instanciamos um Array de bytes. Esse Array será nosso buffer de dados.

```
public void serialEvent(SerialPortEvent ev){
              switch (ev.getEventType()) {
               //…
               case SerialPortEvent.DATA_AVAILABLE:
                    byte[] bufferLeitura = new byte[20];
```

Já definimos entrada como nosso fluxo de entrada de dados. O método available() retorna sempre 0 se InputStream (nesse caso entrada) é classe da qual ele é invocado.

```
                    try {
                        while ( entrada.available() > 0 ) {
                           nodeBytes = entrada.read(bufferLeitura);
                        }
```

O metodo read(byte[] b) faz toda a leitura. Ele copia os bytes lidos para o Array bufferLeitura e retorna um inteiro representando o número de bytes lidos. Podemos converter esses valores para uma String como mostrado abaixo:

```
                        String Dadoslidos = new String(bufferLeitura);
```

Se a dimensão do buffer for igual zero, isso nos dirá que nenhum byte foi lido. Se a dimensão do buffer for igual a 1, saberemos que apenas um byte foi lido. Caso contrário, a estrutura bufferLeitura recebe os bytes lidos. O primerio byte lido é armazenado em bufferLeitura[0], o segundo em bufferLeitura[1] e assim por diante.

```
                        if (bufferLeitura.length == 0) {
                           System.out.println("Nada lido!");
                        } else if (bufferLeitura.length == 1 ){
                           System.out.println("Apenas um byte foi lido!");
                        } else {
                           System.out.println(Dadoslidos);
                        }
                    } catch (Exception e) {
                      System.out.println("Erro durante a leitura: "  + e );
                    }
                    System.out.println("n.o de bytes lidos : " + nodeBytes );
                    break;
               }
      }
```

O Package SrCom

Eu criei um package SrCom com duas classes: SerialComm e SComm.

SerialComm reconhece as portas disponíveis e SComm, envia um mensagem para a porta especificada. Também pode ler uma mensagem. Para isso a classe SComm implementa as interfaces Runnable e o listener de eventos da porta serial.

Classe SerialComm

```
//**************************************************************************
//COMUNICAÇÃO SERIAL UTILIZANDO A API DA SUN
//**************************************************************************
// CLASSE SerialCom.java É PARTE DO PACKAGE SrCom
//**************************************************************************
//AUTOR : Daniel V. Gomes
//EMAIL: dansovg@ig.com.br
//DATA INICIAL: 29/04/04
//DATA ATUAL: 03/05/04
//**************************************************************************
//NOTA: Você pode utilizar este código a vontade mas não me responsabilizo por
//erros durante sua execução. Quaisquer dúvidas ou sugestões,
//envie-me por email.
//**************************************************************************
package SrCom;
import javax.comm.*;
import java.io.*;
import java.util.*;
public class SerialCom {
        //*********************************
        //Variáveis
        //*********************************
        //variáveis para identificar portas
        protected String[] portas;
        protected Enumeration listaDePortas;
        //construtor
        public SerialCom(){
          listaDePortas = CommPortIdentifier.getPortIdentifiers();
        }
        //retorna as portas disponíveis
        public String[] ObterPortas(){
                return portas;
        }
        //Copia portas para um Array
        protected void ListarPortas(){
          int i = 0;
          portas = new String[10];
          while (listaDePortas.hasMoreElements()) {
               CommPortIdentifier ips =
               (CommPortIdentifier)listaDePortas.nextElement();
               portas[i] =  ips.getName();
               i++;
          }
        }
        //pesquisa se a Porta existe
        public boolean PortaExiste(String COMp){
           String temp;
           boolean e = false;
           while (listaDePortas.hasMoreElements()) {
               CommPortIdentifier ips =
               (CommPortIdentifier)listaDePortas.nextElement();
               temp = ips.getName();
               if (temp.equals(COMp)== true) {
                   e = true;
               };
           }
           return e;
        }
        //imprime as portas disponíveis
        protected void ImprimePortas(){
               for (int i = 0 ; i < portas.length ; i ++ ) {
                   if (portas[i] != null ) {
                      System.out.print(portas[i] + "  ");
```

```
                }
            }
            System.out.println(" ");
        }
}//FIM DA CLASSE
```

Classe SComm

```
//*****************************************************************************
//COMUNICAÇÃO SERIAL UTILIZANDO A API DA SUN
//*****************************************************************************
// CLASSE Scomm.java É PARTE DO PACKAGE SrCom
//*****************************************************************************
//AUTOR : Daniel V. Gomes
//EMAIL: dansovg@ig.com.br
//DATA INICIAL: 29/04/04
//DATA ATUAL: 03/05/04
//*****************************************************************************
//NOTA: Você pode utilizar este código a vontade mas não me responsabilizo por
//erros durante sua execução. Quaisquer dúvidas ou sugestões,
//envie-me por email.
//*****************************************************************************
package SrCom;
import javax.comm.*;
import java.io.*;
//classe Principal
public class SComm implements Runnable, SerialPortEventListener {
        //propriedades
        private String Porta;
        public String Dadoslidos;
        public int nodeBytes;
        private int baudrate;
        private int timeout;
        private CommPortIdentifier cp;
        private SerialPort porta;
        private OutputStream saida;
        private InputStream entrada;
        private Thread threadLeitura;
        //indicadores
        private boolean IDPortaOK; //true porta EXISTE
        private boolean PortaOK;// true porta aberta
        private boolean Leitura;
        private boolean Escrita;
        //construtor default  paridade : par
        //baudrate: 9600 bps stopbits: 2 COM 1
        public SComm() {
                Porta = "COM1";
                baudrate = 9600;
                timeout = 1000;
        };
        //um Objeto ComObj é passado ao construtor
        //com detalhes de qual porta abrir
        //e informações sobre configurações
        public SComm( String p , int b , int t ){
                this.Porta = p;
                this.baudrate = b;
                this.timeout = t;
        };
        //habilita escrita de dados
        public void HabilitarEscrita(){
                Escrita = true;
                Leitura = false;
        }

        //habilita leitura de dados
        public void HabilitarLeitura(){
                Escrita = false;
                Leitura = true;
        }
        //Obtém o ID da PORTA
        public void ObterIdDaPorta(){
             try {
                 cp =  CommPortIdentifier.getPortIdentifier(Porta);
                 if ( cp == null ) {
                    System.out.println("A " + Porta + " nao existe!" );
                    System.out.println("ERRO!Abortando..." );
```

```
                IDPortaOK = false;
                System.exit(1);
            }
            IDPortaOK = true;
        } catch (Exception e) {
            System.out.println("Erro durante o procedimento. STATUS" + e );
            IDPortaOK = false;
            System.exit(1);
        }
    }
    //Abre a comunicação da porta
    public void AbrirPorta(){
        try {
            porta = (SerialPort)cp.open("SComm",timeout);
            PortaOK = true;
            System.out.println("Porta aberta com sucesso!");
            //configurar parâmetros
            porta.setSerialPortParams(baudrate,
                                      porta.DATABITS_8,
                                      porta.STOPBITS_2,
                                      porta.PARITY_NONE);
        } catch (Exception e) {
            PortaOK = false;
            System.out.println("Erro ao abrir a porta! STATUS: " + e );
            System.exit(1);
        }
    }
    //função que envie um bit para a porta serial
    public void EnviarUmaString(String msg){
           if (Escrita==true) {
             try {
                 saida = porta.getOutputStream();
                 System.out.println("FLUXO OK!");
             } catch (Exception e) {
                  System.out.println("Erro.STATUS: " + e );
             }
             try {
                System.out.println("Enviando um byte para " + Porta );
                System.out.println("Enviando : " + msg );
                saida.write(msg.getBytes());
                Thread.sleep(100);
                saida.flush();
             } catch (Exception e) {
                System.out.println("Houve um erro durante o envio. ");
                System.out.println("STATUS: " + e );
                System.exit(1);
             }
            } else {
                   System.exit(1);
            }
    }
    //leitura de dados na serial
    public void LerDados(){
           if (Escrita == true){
             try {
                 entrada = porta.getInputStream();
                 System.out.println("FLUXO OK!");
             } catch (Exception e) {
                 System.out.println("Erro.STATUS: " + e );
                 System.exit(1);
             }
             try {
                 porta.addEventListener(this);
                 System.out.println("SUCESSO. Porta aguardando...");
             } catch (Exception e) {
                 System.out.println("Erro ao criar listener: ");
                 System.out.println("STATUS: " + e);
                 System.exit(1);
             }
             porta.notifyOnDataAvailable(true);
             try {
                 threadLeitura = new Thread(this);
                 threadLeitura.start();
             } catch (Exception e) {
                 System.out.println("Erro ao iniciar leitura: " + e );
             }
            }
    }
```

```
        //método RUN da thread de leitura
        public void run(){
                 try {
                     Thread.sleep(5000);
                 } catch (Exception e) {
                     System.out.println("Erro. Status = " + e );
                 }
        }
        //gerenciador de eventos de leitura na serial
        public void serialEvent(SerialPortEvent ev){
                switch (ev.getEventType()) {
                 case SerialPortEvent.BI:
                 case SerialPortEvent.OE:
                 case SerialPortEvent.FE:
                 case SerialPortEvent.PE:
                 case SerialPortEvent.CD:
                 case SerialPortEvent.CTS:
                 case SerialPortEvent.DSR:
                 case SerialPortEvent.RI:
                 case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                      break;
                 case SerialPortEvent.DATA_AVAILABLE:
                      byte[] bufferLeitura = new byte[20];
                      try {
                          while ( entrada.available() > 0 ) {
                             nodeBytes = entrada.read(bufferLeitura);
                          }
                          String Dadoslidos = new String(bufferLeitura);
                          if (bufferLeitura.length == 0) {
                             System.out.println("Nada lido!");
                          } else if (bufferLeitura.length == 1 ){
                             System.out.println("Apenas um byte foi lido!");
                          } else {
                             System.out.println(Dadoslidos);
                          }
                      } catch (Exception e) {
                        System.out.println("Erro durante a leitura: "  + e );
                      }
                      System.out.println("n.o de bytes lidos : " + nodeBytes );
                      break;
                }
        }
        //função que fecha a conexão
        public void FecharCom(){
                try {
                     porta.close();
                     System.out.println("CONEXAO FECHADA>>FIM..");
                } catch (Exception e) {
                     System.out.println("ERRO AO FECHAR. STATUS: " + e );
                     System.exit(0);
                }
        }
        //Acessores
        public String obterPorta(){
                return Porta;
        }
        public int obterBaudrate(){
                return baudrate;
        }
}
```

Extendendo a capacidade das classes

Nesses exemplos, eu utilizei o modo console. Assim, várias mensagens de erro e de notificação de status são enviadas à tela. No entanto, o leitor/a poderia converter essas mensagens para strings e passá-las para JLabels, que informam as mensagens pertinentes. Abaixo, apresento um programa simples que tenta escrever e depois ler um dado na porta serial COM2, se ela existir.

```
//*************************************************************************
//COMUNICAÇÃO SERIAL UTILIZANDO A API DA SUN
//*************************************************************************
//AUTOR : Daniel V. Gomes
//EMAIL: dansovg@ig.com.br
//*************************************************************************
//NOTA: Você pode utilizar este código a vontade mas não me responsabilizo por
//erros durante sua execução. Quaisquer dúvidas ou sugestões,
```

```
//envie-me por email.
//*****************************************************************************
import SrCom.*;
public class Stest2 extends SerialCom {
       public Stest2(){
              super();
       }
       public static void main(String[] args){
                Stest2 st = new Stest2();
                if ( st.PortaExiste("COM2") == true) {
                   System.out.println("Iniciando comunicação!");
                   SComm sc = new SComm("COM2",9600,2000);
                   sc.HabilitarEscrita();
                   sc.ObterIdDaPorta();
                   sc.AbrirPorta();
                   sc.EnviarUmaString("Olá mundo!");
                   sc.FecharCom();
                }
               Stest2 st2 = new Stest2();
               if ( st2.PortaExiste("COM2") == true) {
                   System.out.println("Iniciando comunicação!");
                   SComm sc2 = new SComm("COM2",9600,2000);
                   sc2.HabilitarLeitura();
                   sc2.ObterIdDaPorta();
                   sc2.AbrirPorta();
                   sc2.LerDados();
                   sc2.FecharCom();
                }
       }
}
```

Conclusão

Utilizar a API de comunicação da SUN é uma alternativa à utilizar as APIs do Windows ou mesmo utilizar um componente do Visual Basic. A vantagem é que é orientado a objeto, é rapidamente assimilável e possui ótima performance.

Referências

Sun Communications API:
- http://java.sun.com/developer/Books/javaprogramming/cookbook/
"Java Cook Book", Capítulo 11, disponível para download gratuito .
- http://www.syngress.com/book_catalog/177_lego_JAVA/sample.htm
Outro sample gratuito.
- http://community.borland.com/article/0%2C1410%2C31915%2C00.html
Artigo sobre o uso da API de comunicação elaborado por Rick Proctor da Borland.
- http://java.sun.com/products/javacomm/index.jsp
Site oficial da SUN

Comunicação Serial EIA 232:
- http://www.arcelect.com/rs232.htm
Uma descrição bem completa do padrão serial
- http://www.idc-online.com/html/pocket_guides_free_download.html
IDC. Eles têm alguns tutoriais gratuitos em pdf muito bons sobre conversores A/D, D/A, automação industrial e comunicações (serial, TCP-IP).
- http://users.telenet.be/educypedia/

Daniel Vasconcelos Gomes (*dansovg@ig.com.br*) é um Engenheiro Eletricista bastante interessado em Java e C++. Ele elabora manuais técnicos de PLCs, mas preferiria trabalhar com desenvolvimento. Entusiasta da tecnologia Java, está sempre disposto a mostrar que a linguagem pode ser utilizada não somente para aplicativos web.