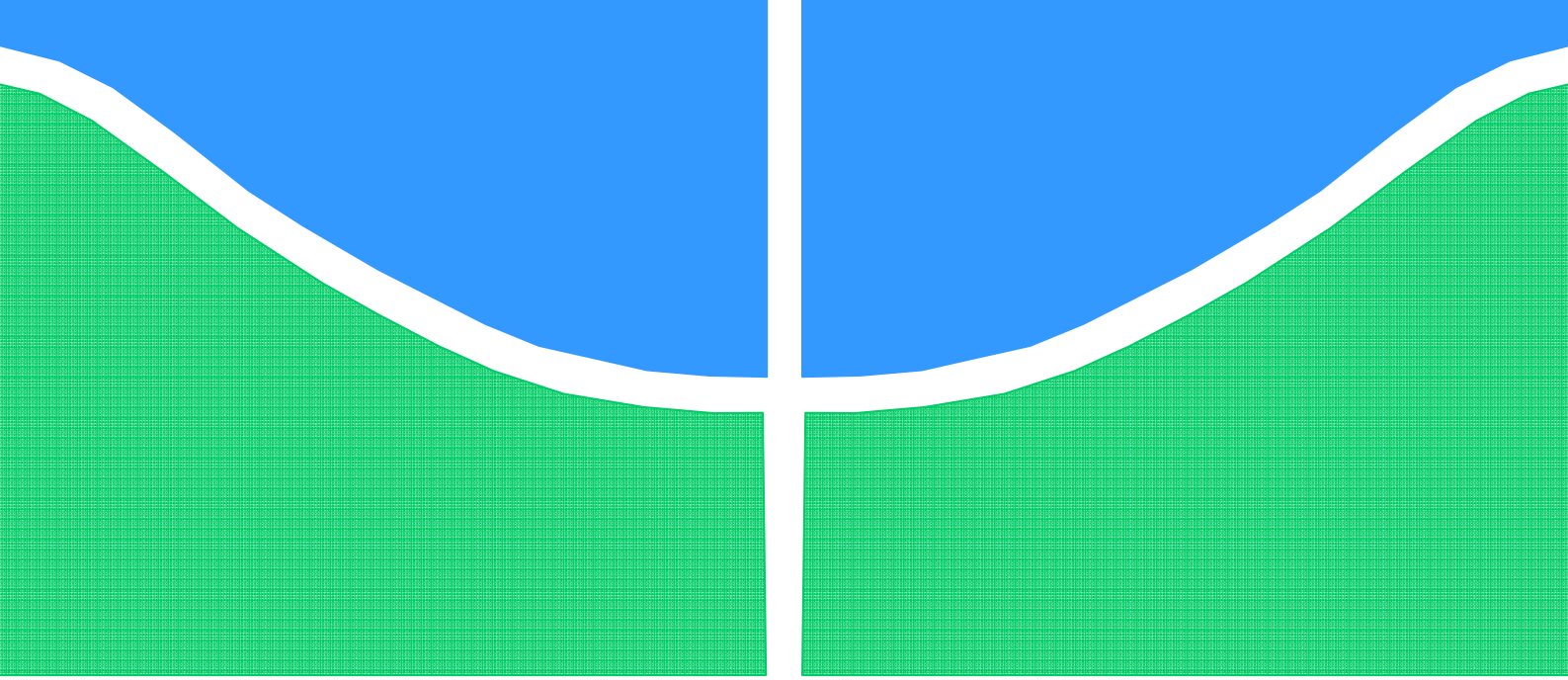

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE SOFTWARE EMULADOR DE TERMINAL DE VÍDEO COMPATÍVEL COM PC DE BAIXO CUSTO

Raphael Pereira David

BRASÍLIA

2009

RAPHAEL PEREIRA DAVID

DESENVOLVIMENTO DE SOFTWARE EMULADOR DE TERMINAL DE VÍDEO COMPATÍVEL COM PC DE BAIXO CUSTO

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

ORIENTADOR: RICARDO ZELENOVSKY

APROVADA POR:

_____

Prof. Ricardo Zelenovsky, Doutor (ENE – UnB)
(Orientador)

_____

Eng. Giuler Alberto Cruz Silva (ENE – UnB)
(Examinador Externo)

_____

Eng. Marcelo Alejandro Villegas Sanchez (ENE – UnB)
(Examinador Externo)

BRASÍLIA/DF, 9 DE JULHO DE 2009.

RESUMO

Esse projeto visa desenvolver um *software* com a função de emular o terminal de vídeo VT100. O emulador será escrito com a linguagem de programação Java, visando a sua execução em diferentes sistemas operacionais. Servirá de base para o emulador um computador de baixo custo com especificações de *hardware* modestas, porém suficientes para executar o programa.

ABSTRACT

This project aims the development of a software which will emulate the functions of the VT100 video terminal. The emulator will be written in Java language programming that will propitiate the execution on various operational systems. The program will run on a low-cost computer with modest hardware specifications, but enough to execute the software.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

1.1 – MOTIVAÇÃO

Terminais de vídeo são comumente utilizados na comunicação com um *host* remoto, tendo como finalidade principal apresentar e manipular em sua tela textos enviados pelo *host*. A manipulação dos textos é feita a partir da interpretação de comandos simples, constituídos por uma sequência de caracteres. Esses terminais são empregados em processos de automação comercial, como consulta de estoques ou bancos de dados de uma maneira geral.

Apesar de antigos, os terminais de vídeo são usados atualmente por diversas empresas, especialmente pela simplicidade de uso e pelo baixo custo de manutenção. Porém, a procura por terminais de vídeo não é crescente e a baixa demanda por esses antigos terminais tem aumentado os custos de sua produção e consequentemente o seu valor final. Assim, sentiu-se a necessidade de procurar alternativas para substituir esses terminais, a um custo menor que o praticado pelas fabricantes.

O desenvolvimento de um emulador em uma linguagem de programação de código aberto e que pudesse ser executado em um computador com especificações de *hardware* modestas, sobre um sistema operacional com livre licença de utilização, motivaram a realização desse projeto.

1.2 – OBJETIVOS

A proposta desse projeto foi desenvolver um *software* que emulasse o terminal de vídeo VT100 e que pudesse ser executado em um computador de baixo custo. O terminal VT100 foi escolhido por ser o mais difundido e utilizado pelas empresas.

O computador escolhido foi um modelo que possui, entre outros componentes, um processador de 200 MHz e 128 MB de memória RAM, e que tem um custo aproximado de US$ 100. Apesar dessas especificações modestas, esse modelo é capaz de executar o programa emulador satisfatoriamente.

A linguagem de programação Java foi usada na escrita do programa emulador por ser uma linguagem de código livre e multiplataforma, permitindo, futuramente, executá-lo em um sistema operacional diferente, caso haja necessidade. Porém, essa escolha trouxe algumas dificuldades, principalmente relacionadas à comunicação serial, pelo padrão RS-232.

1.3 – ORGANIZAÇÃO DO TEXTO

Para uma melhor compreensão e maior organização do trabalho, este foi dividido em oito capítulos.

O primeiro capítulo faz uma introdução ao trabalho, contendo a motivação, os objetivos e esta organização do texto.

O segundo capítulo faz a descrição do terminal de vídeo VT100, apresentando as suas características básicas, como a codificação utilizada na comunicação serial e o formato de seu *display*. Também são descritos os principais comandos interpretados pelo terminal, assim como a sintaxe de tais comandos.

O terceiro capítulo faz um resumo teórico sobre a comunicação serial. Apresentam-se os tipos de comunicação serial, as diversas configurações possíveis de serem realizadas em uma transmissão, além de ser descrito o padrão RS-232, responsável por normatizar os sinais envolvidos nessa comunicação.

O quarto capítulo descreve o computador de baixo custo escolhido para servir de base para o projeto, listando as suas configurações e funcionalidades.

Já o quinto capítulo apresenta os sistemas operacionais testados no *hardware* escolhido, descrevendo as vantagens e desvantagens de cada um, e finalmente explicando a escolha por um desses sistemas.

No sexto capítulo é feita uma apresentação da linguagem de programação Java, utilizada na escrita do programa emulador. São descritas as suas características, suas vantagens sobre outras linguagens, assim como as bibliotecas utilizadas para a construção do programa, tanto graficamente, como funcionalmente.

O sétimo capítulo faz uma descrição do emulador. São comentados alguns pontos principais do código fonte, além de ser apresentada a interface gráfica do programa. Por fim, são apresentados os resultados obtidos ao término do projeto, as dificuldades encontradas e os possíveis desenvolvimentos futuros.

O oitavo capítulo apresenta a conclusão do trabalho.

CAPÍTULO 2 – O TERMINAL DE VÍDEO VT100

2.1 – INTRODUÇÃO

O VT100 é um terminal de vídeo que foi feito originalmente pela *Digital Equipment Corporation* (DEC), tornando-se o terminal padrão para diversas empresas. Ele foi introduzido em 1978, substituindo o antigo VT52. Ele realiza comunicação serial com o *host*, fazendo uso do conjunto de caracteres ASCII (*American Standard Code for Information Interchange*) e sequências de controle (também chamadas de sequências de escape) padronizadas pela ANSI (*American National Standards Institute*).

O terminal VT100 foi o primeiro terminal digital a incorporar controles gráficos, como a modificação do cursor (*blinking*), a formatação da fonte com efeitos como negrito e sublinhado, além de um *display* selecionável entre 80 e 132 colunas. Todas as configurações do VT100 eram realizadas através de *displays* interativos na tela, sendo armazenadas em uma memória não-volátil.

As sequências de escape utilizadas pelo VT100 são baseadas no padrão ANSI X3.64 (posteriormente chamados de ECMA-48 e ISO/IEC 6429). O VT100 também foi o primeiro terminal digital a ser baseado em um microprocessador padrão na indústria da época (o Intel 8080). Adicionalmente, o terminal suportava a conexão de uma impressora externa, além de opções de vídeo avançadas (AVO – *Advanced Video Option*), que permitia ao *display* mostrar 24 linhas quando no modo de 132 colunas. A Figura 2.1 mostra o terminal VT100 original.

Figura 2.1 – Terminal de Vídeo VT100. [1]

2.2 – OPERAÇÃO

Ao operador, o terminal VT100 é bastante simples, podendo comportar-se realizando duas funções: ele pode funcionar como um dispositivo de entrada do *host* (através dos comandos do teclado), e como dispositivo de saída (recebendo os comandos do *host* e exibindo no monitor). Existem diversas características de configurações e mensagens, que são típicos dos equipamentos com as limitações das antigas tecnologias.

Para o terminal desejado, que é baseado em um PC com processador de 200 MHz, com recursos de vídeo e de comunicação muito mais avançados, determinadas configurações tornam-se desnecessárias. Por exemplo, a quantidade de linhas e colunas, as cores de *background* e até a função de algumas teclas do teclado estão, em um PC atual, contidas em um conjunto de configurações bem diferente, no sentido de que o usuário provavelmente não estará limitado às poucas opções trazidas nas versões originais desses terminais.

2.3 – TECLADO DO VT100

O teclado dos antigos terminais VT100 obedecia ao padrão mostrado na Figura 2.2. Nele, é possível ver a maior parte das teclas encontradas nos PC's atuais, com poucas diferenças. Naturalmente, devido à utilização de um computador comum como base para o *software* emulador, um teclado tradicional será utilizado e, portanto, deverá haver adaptações no sentido de que as mesmas funções do teclado original possam ser executadas utilizando-se um teclado tradicional.

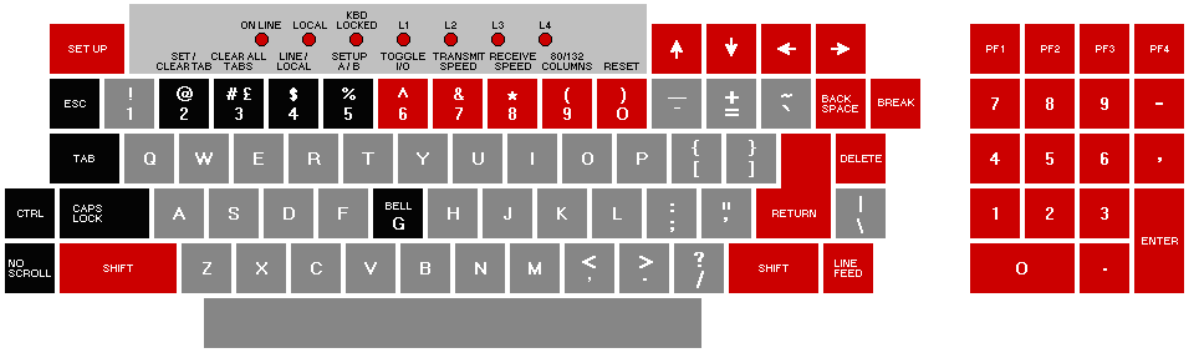

Figura 2.2 – Teclado do terminal VT100. [1]

2.3.1 – Teclas Especiais

Os antigos terminais contavam com algumas teclas especiais, que não constam nos teclados atuais, descritas a seguir:

- **SETUP**: Tecla que era utilizada em conjunto com outras teclas para executar funções específicas, como rolagem de tela e alterações das características e configurações do terminal.
- **BREAK**: Essa tecla enviava um sinal de parada ao *host*.
- **BELL**: Quando era pressionada em conjunto com a tecla CTRL, enviava um código de alerta (campainha) ao *host*.
- **NO SCROLL**: Quando era pressionada pela primeira vez, interrompia a transmissão de dados do *host* para o terminal e, ao ser pressionada pela segunda vez, continuava a transmissão do ponto em que houve a interrupção.
- **LINEFEED**: Enviava um código para o acréscimo de uma nova linha.

2.3.2 – Indicadores Luminosos

O teclado do terminal VT100 original trazia algumas luzes de indicação, que tinham como função informar as situações descritas a seguir:

- **ON LINE**: Indica que o terminal está pronto para enviar e receber mensagens.
- **LOCAL**: Indica que o terminal está *off-line* e não pode comunicar-se com o *host*. No modo local, o teclado permanece ativo e os caracteres eventualmente digitados aparecem na tela.
- **KEYBOARD LOCKED**: Indica que o teclado está desligado. O terminal pode receber dados do *host* assim mesmo.
- **L1 a L4**: Esses indicadores podem ser ligados e desligados a partir do *host*. O significado de cada indicador pode ser personalizado a partir do *host* e da programação das funções do terminal.

No *software* emulador, não existe a necessidade de se acrescentarem indicadores luminosos com LED's propriamente ditos, já que esses indicadores podem ser feitos na tela do próprio programa.

2.4 – COMANDOS DE CONTROLE DO TERMINAL

2.4.1 – Caracteres de Controle

Caracteres de controle são códigos que representam algum tipo de ação a ser tomada pelo terminal. Esses caracteres fazem parte da codificação ASCII, possuindo valores decimais de 0 a 31 e 127, porém nem todos eles são reconhecidos pelo VT100. A Tabela 2.1 lista todos os caracteres de controle, destacando os reconhecidos pelo terminal.

Tabela 2.1 – Caracteres de Controle.

Decimal	Hexa	Abreviação	Descrição
00	**00**	**NUL**	**Null**
01	01	SOH	Start of Header
02	02	STX	Start of Text
03	03	ETX	End of Text
04	04	EOT	End of Transmission
05	**05**	**ENQ**	**Enquiry**
06	06	ACK	Acknowledgement
07	**07**	**BEL**	**Bell**
08	**08**	**BS**	**Backspace**
09	**09**	**HT**	**Horizontal Tabulation**
10	**0A**	**LF**	**Line Feed**
11	**0B**	**VT**	**Vertical Tabulation**
12	**0C**	**FF**	**Form Feed**
13	**0D**	**CR**	**Carriage Return**
14	**0E**	**SO**	**Shift Out**
15	**0F**	**SI**	**Shift In**
16	10	DLE	Data Link Escape
17	**11**	**DC1**	**Device Control 1 (XON)**
18	12	DC2	Device Control 2
19	**13**	**DC3**	**Device Control 3 (XOFF)**
20	14	DC4	Device Control 4
21	15	NAK	Negative Acknowledgement
22	16	SYN	Synchronous Idle
23	17	ETB	End of Transmission Block
24	**18**	**CAN**	**Cancel**
25	19	EM	End of Medium
26	**1A**	**SUB**	**Substitute**
27	**1B**	**ESC**	**Escape**
28	1C	FS	File Separator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator
127	**7F**	**DEL**	**Delete**

2.4.2 – Sequências de Escape

O terminal VT100 tem diversos comandos que não têm a função de exibir caracteres na tela. Assim, o *host* é capaz de comandar o terminal, movendo o cursor, modificando estados e exibindo sinais e alertas sonoros.

Nesse contexto, uma sequência de escape é um conjunto de caracteres usados para mudar o estado de computadores ou terminais de vídeo, sendo também conhecida por sequência de controle, devido ao seu uso para controlar dispositivos. Sequências de escape usam um caractere de escape (ESC) para mudar o sentido dos caracteres que se seguem. Assim, os caracteres seguintes são interpretados como um comando a ser executado, ao invés de dados a serem impressos na tela.

A funcionalidade dessas sequências não é imposta pelos padrões ANSI, ou seja, cada fabricante tem flexibilidade para escolher a função de cada sequência. Entretanto, os terminais VT100 operaram com as funções padronizadas pela ANSI. A seguir são definidos os elementos básicos das sequências de escape ANSI:

- **CSI (*Control Sequence Introducer*)**: Indica que os caracteres que se seguem constituem um comando de controle e é por si só um prefixo para os comandos de controle. No VT100 o CSI é o conjunto das teclas de escape e colchete – `ESC [`.
- **Parâmetro**: Uma *string* de zero ou mais caracteres decimais (de 0 a 9) que representam um único valor.
- **Parâmetro Numérico**: Um parâmetro que representa um número, designado por `Pn`.
- **Parâmetro Seletivo**: Um parâmetro que seleciona uma subfunção, designado por `Ps`. No geral, a sequência de controle com mais de um parâmetro seletivo tem o mesmo efeito de várias sequências de controle.
- ***String* de Parâmetros**: Uma *string* de parâmetros separados por ponto-e-vírgula.
- **Valor Padrão**: Um valor que depende da função, sendo suposto quando nenhum valor é explicitado.
- **Caractere Final**: Um caractere que encerra a sequência de escape.

Dados os elementos básicos, é possível formar então uma sequência de escape. A Figura 2.3 apresenta o formato dessas sequências.

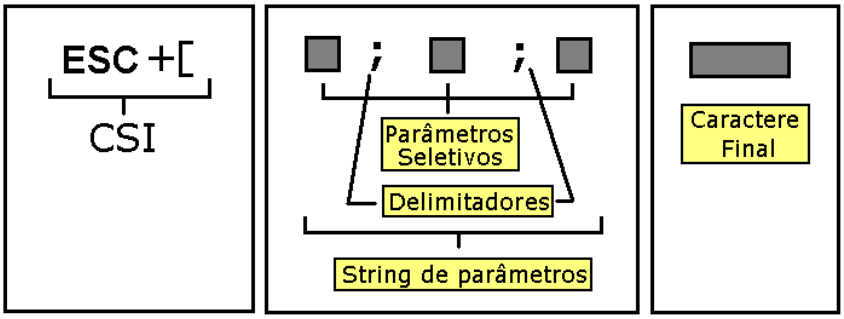

Figura 2.3 – Formato de uma Sequência de Escape.

2.4.3 – Listagem das Sequências de Escape

A seguir são apresentadas as principais sequências de escape utilizadas pelo terminal VT100. Quando não for especificado, a sequência representa um comando enviado do *host* para o terminal.

CPR – Cursor Position Report (VT100 para o *host*)

```
ESC [ Pn ; Pn R
```

A sequência CPR indica ao *host* a posição do cursor. Os dois parâmetros numéricos `Pn` indicam, respectivamente, a linha e a coluna em que o cursor se encontra naquele instante. Caso não seja informado os parâmetros ou o valor deles seja zero ou 1, significa que o cursor está na posição *home* (1ª linha e 1ª coluna). O valor padrão é igual a 1.

CUB – Cursor Backward (VT100 para o *host* e *host* para o VT100)

```
ESC [ Pn D
```

A sequência CUB move o cursor para trás, ou seja, para a esquerda. O parâmetro numérico `Pn` indica em quantas colunas para esquerda o cursor será movido. Se o parâmetro for zero ou 1, o cursor volta exatamente uma coluna para a esquerda. Caso haja uma tentativa de se mover o cursor além da margem esquerda, o cursor para na margem esquerda, isto é, na primeira coluna da tela. O valor padrão é igual a 1.

CUD – Cursor Down (VT100 para o *host* e *host* para o VT100)

```
ESC [ Pn B
```

A sequência CUD move o cursor para baixo, sem alterar a coluna em que se encontra. A quantidade de linhas movidas é indicada pelo parâmetro numérico Pn. Se esse parâmetro for zero ou 1, o cursor desce exatamente uma linha. Caso haja uma tentativa de se mover o cursor além da margem inferior, o cursor para na margem inferior, isto é, na última linha. O valor padrão é igual a 1.

CUF – Cursor Forward (VT100 para o *host* e *host* para o VT100)

```
ESC [ Pn C
```

A sequência CUF move o cursor para a frente, ou seja, para a direita. O parâmetro numérico Pn indica em quantas colunas para a direita o cursor será movido. Se o parâmetro for zero ou 1, o cursor avança exatamente uma posição para a direita. Caso haja uma tentativa de se mover o cursor além da margem direita, o cursor para na margem direita, isto é, na última coluna da tela. O valor padrão é igual a 1.

CUU – Cursor Up (VT100 para o *host* e *host* para o VT100)

```
ESC [ Pn A
```

A sequência CUU move o cursor para cima, sem alterar a coluna em que se encontra. A quantidade de linhas movidas é indicada pelo parâmetro numérico Pn. Se esse parâmetro for zero ou 1, o cursor sobe exatamente uma linha. Caso haja uma tentativa de se mover o cursor além da margem superior, o cursor para na margem superior, isto é, na primeira linha da tela. O valor padrão é igual a 1.

CUP – Cursor Position

```
ESC [ Pn ; Pn H
```

Essa sequência move o cursor para uma posição indicada por dois parâmetros numéricos Pn. Esses dois parâmetros indicam, respectivamente, a linha e a coluna para onde o cursor deve se mover. Caso sejam indicados os valores zero ou 1 para a linha ou coluna, o cursor irá se mover para a primeira linha ou coluna, respectivamente. A condição padrão (sem valores para os parâmetros) corresponde enviar o cursor para a posição *home*.

ED – Erase In Display

```
ESC [ Ps J
```

A sequência ED apaga alguns ou todos os caracteres da tela, dependendo do parâmetro seletivo escolhido. Os parâmetros possíveis encontram-se na Tabela 2.2.

Tabela 2.2 – Parâmetros para o comando Erase In Display. [1]

Valor de `Ps`	Significado
0	Apaga da posição atual até o final da tela (valor padrão).
1	Apaga do início da tela até a posição atual.
2	Apaga toda a tela, mantendo o cursor em sua posição atual.

EL – Erase In Line

```
ESC [ Ps K
```

A sequência EL apaga alguns ou todos os caracteres da linha onde se encontra o cursor, dependendo do parâmetro `Ps`. Os parâmetros possíveis encontram-se na Tabela 2.3.

Tabela 2.3 – Parâmetros para o comando Erase In Line. [1]

Valor de `Ps`	Significado
0	Apaga da posição atual até o final da linha (valor padrão).
1	Apaga do início da linha até a posição atual.
2	Apaga toda a linha, mantendo o cursor em sua posição atual.

DECSC – Save Cursor

```
ESC 7
```

Essa sequência salva a posição atual do cursor.

DECRC – Restore Cursor

```
ESC 8
```

Essa sequência restaura a posição do cursor previamente salva pelo comando DECSC.

IND – Index

```
ESC D
```

Essa sequência faz a atual posição do cursor se mover uma linha para baixo, sem alterar a posição da coluna. Se o cursor se encontrar na margem inferior, a tela “rola” uma linha para cima.

NEL – Next Line

```
ESC E
```

Essa sequência faz a atual posição do cursor se mover para a primeira coluna da linha de baixo. Assim como o comando IND, se o cursor se encontrar na margem inferior, a tela “rola” uma linha para cima.

RI – Reverse Index

```
ESC M
```

Essa sequência move a posição atual do cursor para a linha de cima, não alterando a posição da coluna. Se o cursor se encontrar na margem superior, a tela “rola” uma linha para baixo.

CAPÍTULO 3 – COMUNICAÇÃO SERIAL

3.1 – INTRODUÇÃO

As mensagens digitais, normalmente, são mais longas que alguns poucos bits. Quando não há a necessidade de transferir todos os bits de uma mensagem simultaneamente, a mensagem é quebrada em partes menores e transmitida sequencialmente. A transmissão bit-serial converte a mensagem em um bit por vez através de um canal. Cada bit representa uma parte da mensagem. Os bits individuais são então rearranjados no destino para compor a mensagem original. Em geral, um canal irá passar apenas um bit por vez. A transmissão bit-serial é normalmente chamada de transmissão serial, e é o método de comunicação escolhido por diversos dispositivos eletrônicos, entre eles, o terminal de vídeo VT100.

3.2 – TRANSMISSÃO ASSÍNCRONA VS. TRANSMISSÃO SÍNCRONA

Geralmente, dados serializados não são enviados de maneira uniforme através de um canal, eles são enviados em intervalos não-regulares. Os pacotes de dados binários são enviados dessa maneira, possivelmente com comprimentos de pausa variável entre pacotes, até que a mensagem tenha sido totalmente transmitida. O circuito receptor dos dados deve saber o momento apropriado para ler os bits individuais desse canal e saber exatamente quando um pacote começa e termina. Quando essa temporização for conhecida, o receptor é dito estar sincronizado com o transmissor. Porém, falhas na manutenção do sincronismo durante a transmissão irão causar a corrupção ou perda de dados.

Em sistemas síncronos, canais separados são usados para transmitir informações de dados e informações de controle. O canal de controle transmite pulsos de *clock* para o receptor. Através da recepção de um pulso de *clock*, o receptor lê o canal de dado e armazena o valor do bit encontrado naquele momento. O canal de dados não é lido novamente até que o próximo pulso de *clock* seja recebido. Como o transmissor é responsável pelos pulsos de dados e de controle, o receptor irá ler o canal de dados apenas quando comandado pelo transmissor e, portanto, a sincronização é garantida.

Existem técnicas que compõem o sinal de *clock* e de dados em um único canal. Isso é usual quando transmissões síncronas são enviadas através de um *modem*. Dois métodos no qual os sinais de dados contêm informação de tempo são: codificação NRZ (*Non-Return-to-Zero*) e a codificação Manchester.

Já em sistemas assíncronos, cada transmissão é iniciada com um bit de partida (*start*), que é enviado previamente para cada byte, caractere ou código de palavra, e um sinal de parada (*stop*) é enviado após cada código de palavra. O número de bits de dados e a velocidade de transmissão devem ser pré-estabelecidas entre as partes envolvidas na comunicação. Um oscilador no receptor irá gerar um sinal de *clock* interno que é igual (ou muito próximo) ao do transmissor. Para o protocolo serial mais comum, os dados são enviados em pequenos pacotes de 10 ou 11 bits, dos quais 8 constituem a mensagem. Quando o canal está em repouso, o sinal correspondente no canal tem um nível lógico alto. Um pacote de dados sempre começa com um nível lógico baixo (bit de partida) para sinalizar ao receptor que uma transmissão foi iniciada. O bit de partida inicializa um temporizador interno no receptor avisando que a transmissão começou. Após o bit de partida, 8 bits de dados de mensagem são enviados na taxa de transmissão especificada. O pacote é concluído com os bits de paridade e de parada. Depois do bit de parada, a linha pode ficar ociosa indefinidamente, ou outra mensagem pode ser imediatamente inicializada. A Figura 3.1 mostra um pacote típico de uma transmissão serial.

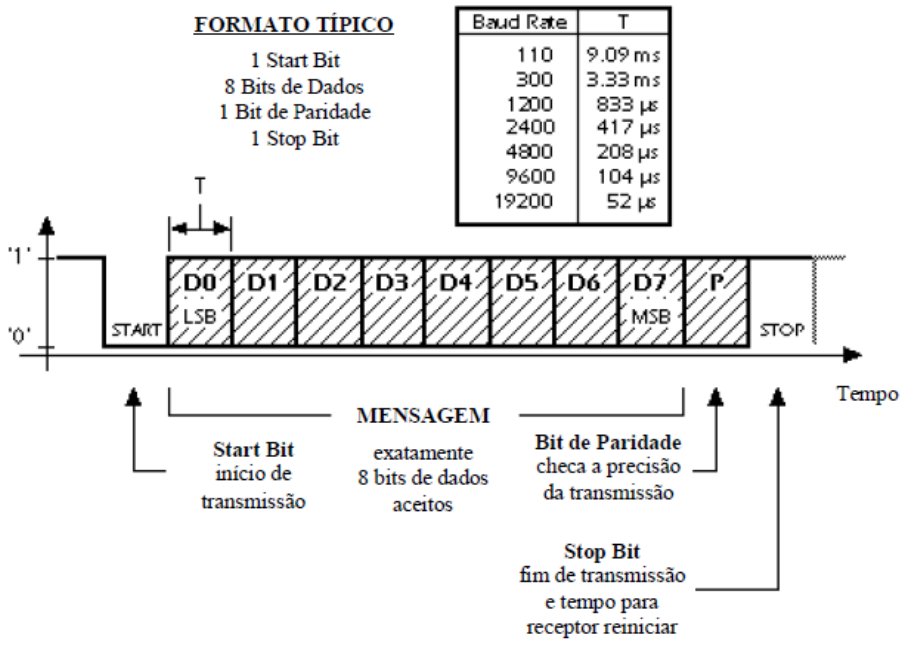

Figura 3.1 – Formato de uma transmissão serial.

3.3 – RS-232

Um padrão recomendado ou RS (*Recommended Standard*) relata uma padronização de uma interface comum para comunicação de dados entre equipamentos, criada no início dos

anos 60, pela EIA (*Electronics Industries Association*). Naquele tempo, a comunicação de dados compreendia a troca de dados digitais entre um computador central (*mainframe*) e terminais de computadores remotos, ou entre dois terminais sem o envolvimento do computador. Esses dispositivos poderiam ser conectados através de linha telefônica, e consequentemente necessitavam de um *modem* em cada lado para fazer a decodificação dos sinais. Dessas ideias, nasceu o RS-232, um padrão para troca serial de dados binários entre um DTE (*Data Terminal Equipment*) e um DCE (*Data Communication Equipment*). Ele especifica as tensões, temporizações e funções dos sinais, um protocolo para troca de informações, e as conexões mecânicas.

A EIA, que padronizou o RS-232 em 1969, define:

- Características elétricas como níveis de tensão, taxa de sinalização, nível máximo de tensão, comportamento de curto-circuito e carga máxima da capacitância;
- Características mecânicas da interface, conectores "plugáveis" e identificação dos pinos;
- Funções de cada circuito no conector da interface;
- Subconjuntos padrões de circuitos de interface para aplicações selecionadas de telecomunicação.

O padrão não define elementos como:

- Codificação de caracteres (por exemplo, ASCII, código Baudot ou EBCDIC);
- Enquadramento dos caracteres no fluxo de dados (bits por caractere, bits de partida e parada, paridade);
- Protocolos para detecção de erros ou algoritmos para compressão de dados;
- Taxas de bit para transmissão, apesar de o padrão dizer ser destinado a taxas de bits menores que 20000 bits por segundo. Muitos dispositivos modernos suportam velocidade de 115200 bits/s;
- Fornecimento de energia para dispositivos externos.

3.3.1 – Conectores

O padrão RS-232 especifica 20 diferentes sinais de conexão, sendo um conector em forma de D comumente usado. O conector recomendado pelo padrão é o DB-25 (com 25 pinos), apesar do seu uso não ser obrigatório. Como a maioria dos dispositivos faz uso somente de alguns dos 20 sinais especificados, conectores menores normalmente são usados. Por exemplo, o conector DE-9 (popularmente conhecido por DB-9), que possui 9 pinos, tem

sido usado na maioria dos computadores pessoais desde os anos 1980, tendo sido padronizado pela norma TIA-574.

A Tabela 3.1 especifica os 22 pinos utilizados pelo padrão RS-232 do conector DB-25. Além de 20 pinos utilizados para sinais, outros dois são usados para terra (*ground*). Nessa tabela, as funções dos pinos estão subdivididas em seis categorias:

1) Sinal de terra e blindagem
2) Canal de Comunicação Primário: usado para troca de dados, incluindo sinais de controle de fluxo.
3) Canal de Comunicação Secundário: quando implementado, é usado para controle remoto do *modem*, requisição de retransmissão quando da ocorrência de erros e controle sobre o *setup* do canal primário.
4) Sinais de Controle e de *Status* de *Modem*: esses sinais indicam o *status* do *modem* e fornece pontos de checagem intermediários durante o estabelecimento da conexão.
5) Sinais de Temporização de Transmissão e Recepção: se for usada uma conexão assíncrona, esses sinais fornecem informações sobre a temporização do transmissor e receptor, que podem operar com taxas diferentes.
6) Sinais de Teste do Canal de Comunicação: antes que os dados sejam trocados, o canal pode ser testado sobre a sua integridade e a taxa de transmissão pode ser ajustada automaticamente para a máxima taxa suportada pelo canal.

Tabela 3.1 – Pinagem do padrão RS-232.

PINO	NOME	DESCRIÇÃO
Sinais de Terra		
1	Shield	Sinal de terra de proteção (malha de aterramento do cabo e carcaça do conector).
7	Ground (GND)	Sinal de terra utilizado como referência para outros sinais.
Canal de Comunicação Primário		
2	Transmitted Data (TxD)	Este sinal está ativo quando dados estiverem sendo transmitidos do DTE para o DCE. Quando nenhum dado estiver sendo transmitido, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
3	Received Data (RxD)	Este sinal está ativo quando o DTE receber dados do DCE. Quando o DCE estiver em repouso, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
4	Request To Send (RTS)	Este sinal é habilitado (nível lógico “0”) para preparar o DCE para aceitar dados transmitidos pelo DTE. Esta preparação inclui a habilitação dos circuitos de recepção, ou a seleção da direção do canal em aplicações *half-duplex*. Quando o DCE estiver pronto, ele

		responde habilitando o sinal CTS.
5	Clear To Send (CTS)	Este sinal é habilitado (nível lógico “0”) pelo DCE para informar ao DTE que a transmissão pode começar. Os sinais RTS e CTS são comumente utilizados no controle do fluxo de dados em dispositivos DCE.
Canal de Comunicação Secundário		
14	Secondary Transmitted Data (STxD)	Equivalente ao sinal TxD, porém válido para o canal secundário.
16	Secondary Received Data (SRxD)	Equivalente ao sinal RxD, porém válido para o canal secundário.
19	Secondary Request To Send (SRTS)	Equivalente ao sinal RTS, porém válido para o canal secundário.
13	Secondary Clear To Send (SCTS)	Equivalente ao sinal CTS, porém válido para o canal secundário.
Sinais de Controle e de *Status* de *Modem*		
6	DCE Ready (DSR)	Também chamado de *Data Set Ready*. Quando originado de um *modem*, este sinal é habilitado (nível lógico “0”) quando as seguintes condições forem satisfeitas: 1) O *modem* estiver conectado a uma linha telefônica ativa e “fora do gancho”; 2) O *modem* estiver no modo dados; 3) O *modem* tiver completado a discagem e está gerando um tom de resposta. Se a linha for tirada do gancho, uma condição de falha for detectada, ou uma conexão de voz for estabelecida, o sinal DSR é desabilitado (nível lógico “1”).
20	DTE Ready (DTR)	Também chamado de *Data Terminal Ready*. Este sinal é habilitado (nível lógico “0”) pelo DTE quando for necessário abri o canal de comunicação. Se o DCE for um *modem*, a habilitação do sinal DTR prepara o *modem* para ser conectado ao circuito do telefone, e uma vez conectado, mantém a conexão. Quando o sinal DTR for desabilitado (nível lógico “1”), o *modem* muda para a condição “no gancho” e termina a conexão.
8	Received Line Signal Detector (CD)	Também chamado de *Data Carrier Detect* (DCD). Este sinal é relevante quando o DCE for um *modem*. Ele é habilitado (nível lógico “0”) quando a linha telefônica está “fora do gancho”, uma conexão for estabelecida, e um tom de resposta começar a ser recebido do *modem* remoto. Este sinal é desabilitado (nível lógico “1”) quando não houver tom de resposta sendo recebido, ou quando o tom de resposta for de qualidade inadequada para o *modem* local.
12	Secondary Received Line Signal Detector (SCD)	Este sinal é equivalente ao CD, porém refere-se ao canal de comunicação secundário.

22	Ring Indicator (RI)	Este sinal é relevante quando o DCE for um *modem*, e é habilitado (nível lógico “0”) quando um sinal de chamada estiver sendo recebido na linha telefônica. A habilitação desse sinal terá aproximadamente a duração do tom de chamada, e será desabilitado entre os tons ou quando não houver tom de chamada presente.
23	Data Signal Rate Selector	Este sinal pode ser originado tanto no DTE quanto no DCE (mas não em ambos), e é usado para selecionar um de dois “baud rates” pré-configurados. Na condição de habilitação (nível lógico “0”) o “baud rate” mais alto é selecionado.
Sinais de Temporização de Transmissão e Recepção		
15	Transmitter Signal Element Timing (TC)	Também chamado de *Transmitter Clock* (TxC). Este sinal é relevante apenas quando o DCE for um *modem* e operar com um protocolo síncrono. O *modem* gera este sinal de *clock* para controlar exatamente a taxa na qual os dados estão sendo enviados pelo pino TxD, do DTE para o DCE. A transição de um nível lógico “1” para nível lógico “0” nessa linha causa uma transição correspondente para o próximo bit de dado na linha TxD.
17	Receiver Signal Element Timing (RC)	Também chamado de *Receiver Clock* (RxC). Este sinal é similar ao sinal TC descrito acima, exceto que ele fornece informações de temporização para o receptor do DTE.
24	Transmitter Signal Element Timing (ETC)	Também chamado de *External Transmitter Clock*. Os sinais de temporização são fornecidos externamente pelo DTE para o uso por um *modem*. Este sinal é utilizado apenas quando os sinais TC e RC não estão sendo utilizados.
Sinais de Teste do Canal de Comunicação		
18	Local Loopback (LL)	Este sinal é gerado pelo DTE e é usado para colocar o *modem* no estado de teste. Quando o sinal LL for habilitado (nível lógico “0”), o *modem* redireciona o sinal de saída modulado, que normalmente vai para o linha telefônica, de volta para o circuito de recepção. Isto habilita a geração de dados pelo DTE serem ecoados através do próprio *modem*. O *modem* habita os sinal TM reconhecendo que ele está na condição de “loopback”.
21	Remote Loopbalk (RL)	Este sinal é gerado pelo DTE e é usado para colocar o *modem* remoto no estado de teste. Quando o sinal RL é habilitado (nível lógico “0”), o *modem* remoto redireciona seus dados recebidos para a entrada, voltando para o *modem* local. Quando o DTE inicia esse teste, o dado transmitido passa através do *modem* local, da linha telefônica, do *modem* remoto, e volta, para exercitar o canal e confirmar sua integridade.
25	Test Mode (TM)	Este sinal é relevante apenas quando o DCE é um *modem*. Quando habilitado (nível lógico “0”), indica que o *modem* está em condição de teste local (LL) ou remoto (RL).

Com dito anteriormente, a maioria desses sinais não é utilizada pelos dispositivos atuais. Assim, o cabo DE-9 é usado no lugar do DB-25. A Tabela 3.2 faz a relação entre a pinagem do conector DB-25 com a pinagem do DE-9.

Tabela 3.2 – Relação entre os conectores DB-25 e DE-9.

Sinal	DB-25	DE-9
GND	7	5
TxD	2	3
RxD	3	2
DTR	20	4
DSR	6	6
RTS	4	7
CTS	5	8
DCD	8	1
RI	22	9

3.4 – CONFIGURAÇÕES DE UMA CONEXÃO SERIAL

Várias configurações são necessárias para que uma conexão serial assíncrona seja realizada com sucesso, tais como a taxa de transmissão, o número de bits de dados por caractere, paridade e o número de bits de parada por caractere. Nas conexões seriais modernas, usando um circuito integrado UART (*Universal Asynchronous Receiver/Transmitter* – Receptor/Transmissor Assíncrono Universal), todas essas configurações são normalmente controladas pelo *software*; enquanto equipamentos antigos requerem ajustes em *switches* ou *jumpers*. Caso as configurações sejam feitas de forma errada, a conexão não será interrompida, porém qualquer dado recebido será interpretado de forma equivocada.

3.4.1 – Taxa de Transmissão

Conexões seriais usam uma sinalização binária de dois níveis, assim a taxa de transmissão em bits por segundo é igual a taxa de símbolos por segundo. Essas taxas são baseadas em múltiplos das taxas usadas em um teletipo eletromecânico. As taxas de transmissão da porta serial e do equipamento devem coincidir, apesar de que determinadas portas podem não suportar todas as taxas possíveis.

A velocidade de transmissão inclui todos os bits do *frame* (bits de parada, paridade, etc.), portanto a taxa efetiva é sempre menor que a taxa de transmissão de bits. Por exemplo,

na transmissão de um caractere com um *frame* que possui 8 bits de dados, 1 bit de parada e sem paridade, somente 80% dos bits do *frame* estão disponíveis para os dados propriamente ditos.

3.4.2 – Bits de Dados

O número de bits de dados utilizados em cada caractere pode ter os seguintes valores: 5 (para o código Baudot), 6 (raramente usado), 7 (para código ASCII), 8 (para qualquer tipo de dado) e 9 (raramente usado). Aplicações atuais geralmente fazem uso de 8 bits de dados, enquanto 5 ou 7 bits geralmente são usados somente para a compatibilidade com equipamentos antigos.

A maioria dos projetos para comunicação serial envia primeiro os bit menos significativos (LSB – *Least Significant Bit*). Esse padrão é conhecido como *little endian*. Raramente utilizado, o padrão *big endian* envia os bits mais significativos (MSB – *Most Significant Bit*) primeiro. Normalmente, a ordem de envio dos bits não é configurável.

3.4.3 – Paridade

Ruídos e distúrbios elétricos momentâneos podem causar mudanças nos dados quando estão trafegando pelos canais de comunicação. Se o receptor falhar ao detectar isso, a mensagem recebida será incorreta, tornado a detecção de erros uma tarefa importante. Se for possível detectar o erro, o pacote errado pode ser reenviado, ou no mínimo os dados serão reconhecidos como incorretos. Se uma redundância na informação for enviada, 1 ou 2 bits de erros podem ser corrigidos pelo *hardware* no receptor. O bit de paridade é adicionado ao pacote de dados com o propósito de detecção de erro.

Na convenção de paridade-par (*even-parity*), o valor do bit de paridade é escolhido de tal forma que o número total de dígitos ‘1’ dos dados adicionado ao bit de paridade do pacote seja sempre um número par. Na recepção do pacote, a paridade do dado precisa ser recomputada pelo *hardware* local e comparada com o bit de paridade recebido com os dados. Se qualquer bit mudar de estado, a paridade não irá coincidir, e um erro será detectado. Porém, se um número par de bits tiver sido alterado, a paridade coincidirá e o dado com erro será validado. Contudo, uma análise estatística dos erros de comunicação de dados tem mostrado que um erro com bit simples é muito mais provável que erros em múltiplos bits na presença de ruído aleatório aditivo.

Apesar de a paridade permitir a detecção de erros, um único bit de paridade não permite a implementação para que esse erro seja corrigido. Assim, protocolos de comunicação em conexões seriais devem possuir mecanismos de alto nível para garantir a validade dos dados e, se for o caso, requisitar a retransmissão dos dados recebidos incorretamente.

O bit de paridade para cada caractere pode ser ajustado para *none* (N), *odd* (O), *even* (E), *mark* (M) ou *space* (S). *None* significa que nenhum bit de paridade está presente. Quando o bit está presente, mas não é usado, ele pode ser sempre 1 (*mark*) ou 0 (*space*). Já as paridades ímpar (*odd*) e par (*even*) informam o número de bits 1 por caractere. Apesar dessas diversas formas de paridade, o mais comum é que ela não seja usada, deixando que protocolos de comunicação de camadas superiores sejam responsáveis pela detecção de erros.

3.4.4 – Bits de Parada

Os bits de parada enviados no fim de cada caractere permitem que o *hardware* receptor detecte o final do caractere e possa ressincronizar com o fluxo de dados. Normalmente, dispositivos eletrônicos usam 1 bit de parada, apesar de que dispositivos mais antigos podem usar 2 ou até 1,5 bits de parada.

3.4.5 – Notação Convencional

A notação D/P/S especifica as configurações do *frame* de uma conexão serial (D para bits de dados, P para bit de paridade e S para bits de parada). O uso mais comum em microcomputadores é a notação 8/N/1, que significa que a conexão usará 8 bits de dados, nenhum bit de paridade e 1 bit de parada. Já para uma notação 7/E/1, um bit de paridade par é adicionado a sete bits de dados, resultando no total de oito bits entre os bits de início e de parada. Se um receptor configurado para um fluxo de dados 7/E/1 receber um fluxo 8/N/1, metade dos bytes serão interpretados como tendo o bit mais significativo em nível alto.

3.4.6 – Controle de Fluxo

Às vezes é necessário regular o fluxo de dados quando estamos transferindo dados entre duas interfaces seriais. Isso pode ser devido a limitações em uma das interfaces ou em um dispositivo de armazenamento. O fluxo de dados pode ser regulado utilizando um controle por *software* ou um controle por *hardware*.

No controle de fluxo por *software*, são utilizados caracteres especiais para iniciar (XON) ou encerrar (XOFF) o fluxo de dados. Esses caracteres são definidos na codificação ASCII. Nesse tipo de controle de fluxo, o caractere XON avisa o transmissor que o receptor está pronto para enviar dados. Já o XOFF avisa o transmissor para interromper a transmissão dos dados até que o receptor esteja pronto para receber novamente.

No controle de fluxo por *hardware*, são utilizados os sinais CTS (*Clear To Send*) e RTS (*Request To Send*) ao invés dos caracteres especiais. O receptor deixa o CTS em nível lógico baixo quando ele está pronto para receber mais dados e em nível lógico alto quando ele não está pronto. Do mesmo modo, o transmissor deixa o RTS em nível baixo quando está pronto para enviar dados e em nível alto quando não está pronto. Como o controle de fluxo por *hardware* utiliza dois sinais separados, ele é muito mais rápido do que o controle por *software* que precisa enviar ou receber caracteres de controle. O controle por *hardware* não é suportado por todos os dispositivos e sistemas operacionais.

CAPÍTULO 4 – MINI-PC

O termo Mini-PC, na verdade, refere-se a uma ampla variedade de dispositivos que têm as funcionalidades de um PC comum, porém possuem um tamanho reduzido. Existem Mini-PC's de diversos tamanhos, configurações e preços. Há Mini-PC's um pouco maiores que duas ou três caixas de CD até outros cujo gabinete tem até metade do tamanho dos gabinetes convencionais. Os processadores variam desde modelos de 200MHz até modelos de cerca de 3GHz, atendendo a diversas funcionalidades e aplicações. Os preços variam desde US$ 100 até US$ 1000, dependendo da configuração e do tamanho. Como um dos objetivos do projeto também é a redução de custos, quando comparado a um terminal de vídeo dedicado, escolheu-se um modelo com o custo de US$ 100.

O emulador VT100 desenvolvido nesse projeto teve como base o Mini-PC, modelo TU-40, fabricado pela e-Way Technology Systems Corp., uma empresa sediada em Taiwan. A princípio suas especificações pareciam satisfazer os requisitos necessários para que o emulador executasse satisfatoriamente.

As dimensões do Mini-PC utilizado são bem reduzidas: 11,5 x 11,5 x 3,5cm. Ele é um modelo *fanless*, isto é, não possui ventoinha para dissipar o calor, sendo a sua própria carcaça responsável por fazer uma dissipação passiva, tendo em vista que o consumo de energia é bastante baixo: a potência consumida é de cerca de 20 W DC. A Figura 4.1 mostra o Mini-PC utilizado no projeto, tendo o seu tamanho comparado com o de uma caneta.

Figura 4.1 – Mini-PC modelo TU-40 da e-Way. [4]

A seguir são listadas as especificações do Mini-PC obtidas a partir do manual do equipamento:

- CPU: Compatível com x86, 200MHz.
- Memória RAM principal: 128MB SD RAM
- BIOS: AMI BIOS
- Vídeo: AGP Ver. 2 Compliant (resolução até 1280x1024 em High Color, compartilhando até 8 MB de memória)
- Áudio: AC-97 CODEC, totalmente compatível com AC-97 v.2.1
- Rede: Lan Realtek 8100-B 10/100 Mbps Ethernet interface
- Rede wireless (opcional)
- On-board IDE: Enhanced IDE interface, 44-pin box header x1.
- Portas para periféricos:
- 3 portas USB v.1.1
- 2 portas seriais (opcional)
- Áudio (Mic-in, Line-in)
- Slot para memória CompactFlash Type I/II (suporte à memória Micro Drive)
- Alimentação: 5,0 ~ 5,25 VDC @ 3A max
- Peso: 506 g
- Sistema operacional: Adequado para uso com Windows CE ou XP embarcado e Linux embarcado.

As Figuras 4.2 e 4.3 mostram as vistas frontal e traseira, evidenciando os conectores, indicadores e botões do Mini-PC. O modelo mostrado nas figuras possui as duas portas de comunicação serial e a antena do adaptador *wireless* opcionais.

Figura 4.2 – Vista Frontal do Mini-PC. [4]

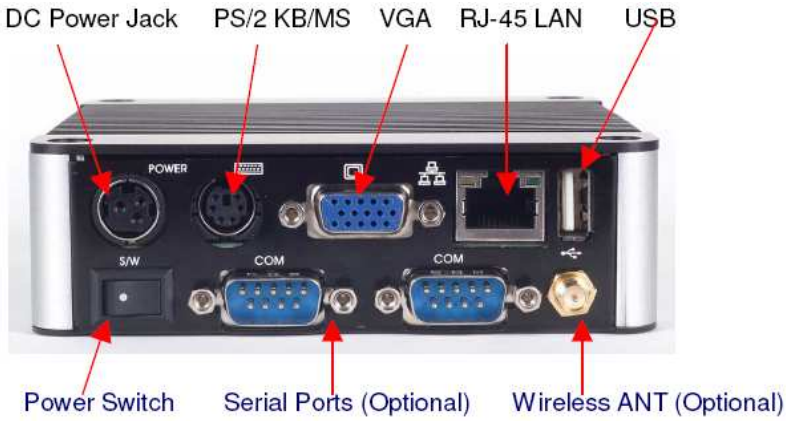

Figura 4.3 – Vista Traseira do Mini-PC. [4]

Apesar das figuras acima mostrarem o Mini-PC contendo as portas de comunicação serial, o modelo adquirido para o projeto não as possuía. Esse contratempo foi resolvido utilizando-se um adaptador USB-Serial, que é conectado à porta USB, disponível no Mini-PC. Porém, essa solução trouxe alguns problemas, conforme explicado nos tópicos seguintes.

CAPÍTULO 5 – SISTEMA OPERACIONAL

Tendo escolhido o Mini-PC como base para a execução do *software* emulador, teve-se a necessidade de se escolher um sistema operacional que consumisse poucos recursos de *hardware*, dada a limitação do Mini-PC nessa área. Dessa forma, o sistema operacional deveria ser leve e funcional, além de ser ágil o suficiente para executar aplicações desenvolvidas em Java.

5.1 – WINDOWS 98

O Windows 98 foi o primeiro sistema operacional escolhido para ser executado no Mini-PC. Tal escolha deveu-se ao fato do Windows ser um sistema com uma interface gráfica bastante amigável, além da maioria dos usuários ter uma maior facilidade em seu manuseio.

A sua instalação ocorreu sem problemas, tendo sido instalado com êxito todos os *drivers* de *hardware*. Após a instalação do Windows, instalou-se a Máquina Virtual Java versão 5.0, já que essa é a última versão compatível com o Windows 98. Apesar da instalação de todos os componentes necessários ter sido finalizada com sucesso, o adaptador USB-Serial não funcionou adequadamente, apesar de o seu *driver* também ter sido instalado adequadamente.

Devido a esses problemas encontrados com o uso do adaptador USB-Serial, além da versão do Java disponível para o Windows 98 não ser a mais atual, decidiu-se abandonar o uso desse sistema operacional, em favor de um sistema mais moderno.

5.2 – PUPPY LINUX

Após a decisão de não utilizar o Windows 98, procurou-se um sistema operacional que pudesse substituí-lo, mantendo uma interface gráfica amigável e uma execução leve, além de possuir suporte a dispositivos mais modernos, devido ao uso do adaptador USB-Serial. Nesse sentido, encontrou-se o Puppy Linux, uma distribuição Linux de código aberto que tem como público alvo a execução diretamente do CD ou, caso instalado, em computadores antigos.

O Puppy Linux é uma distribuição que chama a atenção pelo tamanho. A sua imagem (arquivo .ISO) para execução diretamente pelo CD ou para posterior instalação, possui cerca de 100 MB. Além disso, oferece outros atributos, como:

- *Live booting* a partir de CD's, pen drives e outros tipos de mídia removível;
- Possibilidade de rodar diretamente a partir da RAM;
- Pode-se rodar satisfatoriamente em PC's antigos com 64 MB de RAM.

Apesar de sua única funcionalidade, nesse projeto, ser a de servir como base para a execução de aplicativos Java, o Puppy Linux também oferece suporte a diversos aplicativos de uso comum, como programas de escritório (editores de texto, planilhas e apresentações), de comunicação via Internet, alguns jogos e editores de imagem.

Durante o desenvolvimento do projeto, já se encontrava disponível a versão 4.0 desse sistema operacional. Porém, optou-se por utilizar a versão 3.01 por ser uma versão mais antiga e, portanto, mais estável. Os procedimentos para a sua correta instalação encontram-se descritos no próximo item.

5.2.1 – Instalação

Um dos motivos para que a distribuição Puppy Linux fosse a escolhida para servir de base no projeto foi a facilidade que ela apresenta na instalação, principalmente em computadores que não possuem um disco rígido, e que utilizam formas alternativas para iniciar o sistema. Como o Mini-PC utiliza uma memória *CompactFlash* no lugar de um disco rígido, a instalação deve ser feita com o auxílio de um outro computador, para que seja possível rodar o CD de instalação. Os seguintes passos devem ser seguidos para instalar o Puppy Linux no Mini-PC:

1) Baixar a imagem ISO do Puppy Linux 3.01 em http://www.puppylinux.org/. O nome do arquivo de imagem é "puppy-3.01-seamonkey.iso".
2) Gravar a imagem em um CD e executar o boot pelo CD em qualquer computador.
3) Já no Puppy Linux, formatar a memória *CompactFlash* com o sistema de arquivos EXT3 através do programa GParted.
4) Abrir o Puppy Universal Installer e escolher a opção "USB CF Flash drive, later move to IDE-adapter", para instalar o Linux na *CompactFlash*, bastando seguir as opções *default*.
5) Após a primeira inicialização do sistema, reiniciar ou desligar o Mini-PC, escolhendo, ao ser perguntado, a opção "Save to hda1".

Conforme dito anteriormente, utilizou-se um adaptador USB-Serial para realizar as transmissões de dados. Para o correto funcionamento do adaptador no Puppy Linux, não é necessária a instalação de nenhum *driver*, visto que eles já se encontram embutidos no sistema operacional. Porém, é necessário habilitá-los, pois eles não são habilitados por padrão. Assim, devem-se executar as seguintes ações:

1) No *Boot Manager* do Puppy Linux, habilitar os módulos “pl2303” e “usbserial”, para que eles sejam carregados no boot do sistema operacional.
2) No prompt de comando, digitar:
   `ln –s /dev/usb/ttyUSB0 /dev/ttyUSB0`

Obviamente que se o computador utilizado na execução do emulador possuir portas seriais nativas, as ações descritas acima são dispensáveis.

CAPÍTULO 6 – LINGUAGEM DE PROGRAMAÇÃO JAVA

6.1 – INTRODUÇÃO

Java é uma linguagem de programação orientada a objeto lançada em 1995 pela empresa *Sun Microsystems*. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é posteriormente interpretado por uma máquina virtual.

O objetivo do projeto que deu origem ao Java não era a criação de uma nova linguagem de programação, mas antecipar e planejar a "revolução" do mundo digital. Eles acreditavam que futuramente haveria uma convergência entre os computadores com os equipamentos e eletrodomésticos comumente usados pelas pessoas no seu cotidiano.

A tecnologia Java tinha sido projetada para se mover por meio de redes de dispositivos heterogêneos. Com a disseminação da Internet, aplicações poderiam ser executadas dentro dos browsers (*Applets* Java) e tudo seria disponibilizado pela Internet instantaneamente. Foi o estático HTML dos browsers que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos seguintes foi assustadora, o número de usuários cresceu rapidamente, grandes fornecedores de tecnologia, como a IBM anunciaram suporte para a tecnologia Java.

Desde seu lançamento, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2004, a linguagem Java atingiu a marca de três milhões de desenvolvedores em todo mundo. Java continuou crescendo e hoje é uma referência no mercado de desenvolvimento de *software*, tornado-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em web browsers, *mainframes*, sistemas operacionais, celulares, palmtops e cartões inteligentes, entre outros.

A linguagem Java foi projetada tendo em vista os seguintes objetivos:

- Orientação a objeto;
- Portabilidade – possibilitar a independência de plataforma ("*write once, run anywhere*");
- Recursos de rede – possuir rotinas que facilitam a cooperação com protocolos TCP/IP;
- Segurança – executar programas via rede com restrições de execução.

Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

- Facilidades de internacionalização – suporta nativamente caracteres Unicode;

- Simplicidade na especificação, tanto da linguagem como do ambiente de execução;
- É distribuída com um vasto conjunto de bibliotecas;
- Possui facilidades para criação de programas distribuídos e multitarefa;
- Desalocação de memória automática por processo de coletor de lixo (*garbage collector*);
- Carga dinâmica de código – programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

6.2 – MÁQUINA VIRTUAL JAVA

6.2.1 – Introdução

Conforme dito anteriormente, na linguagem Java, o código fonte do programa é primeiramente compilado para uma forma intermediária de código, denominada *bytecode*. Posteriormente, o *bytecode* é interpretado pela máquina virtual Java (*Java Virtual Machine* – JVM). Acreditava-se que por causa desse processo, o código interpretado Java tinha baixo desempenho, o que atualmente não é verdade. Apesar de durante muito tempo esta ter sido uma afirmação verdadeira, novos avanços têm tornado a máquina virtual mais eficiente que o compilador estático.

Tais avanços incluem otimizações como a compilação especulativa, que aproveita o tempo ocioso do processador para pré-compilar *bytecode* para código nativo. Outros mecanismos ainda mais elaborados como o *HotSpot* da Sun, que guarda informações disponíveis somente em tempo de execução para otimizar o funcionamento da JVM, possibilitando que a máquina virtual vá "aprendendo" e melhorando seu desempenho.

Essa implementação, no entanto, tem algumas limitações intrínsecas. A pré-compilação exige tempo, o que faz com que programas Java demorem um tempo significativamente maior para iniciarem a sua execução. Soma-se a isso o tempo de carregamento da máquina virtual. Isso não é um grande problema para programas que rodam em servidores e que normalmente são inicializados apenas uma vez. No entanto, em computadores pessoais, o usuário muitas vezes não tem a paciência necessária para aguardar o completo carregamento da aplicação.

O Java ainda possui outra desvantagem considerável em programas que usam bastante processamento numérico. O padrão Java tem uma especificação rígida de como devem funcionar os tipos numéricos. Essa especificação não condiz com a implementação de pontos

flutuantes na maioria dos processadores, o que faz com que o Java seja significativamente mais lento para estas aplicações quando comparado a outras linguagens.

Os *bytecodes* produzidos pelos compiladores Java podem ser usados num processo de engenharia reversa para a recuperação do código fonte original. Esta é uma característica que atinge em menor grau todas as linguagens compiladas. No entanto, atualmente existem tecnologias que "embaralham" e até mesmo criptografam os *bytecodes* praticamente impedindo a engenharia reversa.

6.2.2 – Ambiente de Execução

Programas feitos para serem executados em uma JVM são compilados em um formato binário portável padronizado, gerando um arquivo com a extensão ".class". Tais programas podem consistir de várias classes em diferentes arquivos. Para uma fácil distribuição de programas grandes, múltiplos arquivos ".class" podem ser colocados juntos em um único arquivo com extensão ".jar" (*Java archive* – arquivo Java).

A JVM, que é uma instância da JRE (*Java Runtime Environment* – Ambiente de Tempo de Execução Java), entra em ação quando um programa Java é executado. Quando a sua execução está completa, essa instância é excluída. Um compilador JIT (*just-in-time Compiler*) compila partes do *bytecode*, que possuem funcionalidades similares, ao mesmo tempo, reduzindo a quantidade de tempo necessário para a compilação.

6.3 – SWING

6.3.1 – Introdução

Swing é um *framework* da linguagem Java, utilizada na construção de interfaces gráficas. Assim com a própria linguagem, ele é independente da plataforma. Adicionalmente, ele utiliza o padrão de arquitetura da engenharia de *software* MVC (*Model–view–controller*), que consiste na separação os dados da aplicação e a interface do usuário. Assim, é possível alterar tanto os dados como a interface, sem que um interfira no outro.

Esse *framework* faz parte da JFC (*Java Foundation Classes*), uma API (*Application Programming Interface* – Interface de Programação de Aplicativos) específica para a construção de GUI (*Graphical User Interface* – Interface Gráfica do Usuário) e que além do Swing, contém os *frameworks* AWT (*Abstract Window Toolkit*) e Java 2D.

O Swing foi desenvolvido para prover um conjunto de componentes GUI mais sofisticados que os componentes AWT. Ele possui uma aparência nativa, mas emula a aparência de várias plataformas, inclusive possibilitando que a aparência da aplicação não esteja relacionada com a plataforma em que reside.

Toda interface gráfica do *software* emulador do terminal de vídeo VT100 foi desenvolvida com o *framework* Swing.

6.3.2 – Características

Além de ser independente da plataforma, o Swing possui uma arquitetura altamente particionada, permitindo a customização de interfaces já especificadas. Dessa forma, o programador pode prover suas próprias implementações desses componentes, sobrescrevendo as implementações padrões. De um modo geral, o programador pode estender o *framework*, estendendo classes já existentes, fornecendo implementações alternativas dos componentes.

Dado o modelo de renderização programática do Swing, é possível ter um bom controle sobre os detalhes de renderização dos componentes. Como um padrão geral, a representação visual de um componente do Swing é uma composição de um conjunto de elementos padrões, como "*border*", "*inset*", *decorations*, etc. Tipicamente, o programador pode customizar um componente do Swing, especificando bordas, cores, *backgrounds*, etc., como propriedades desse componente. Assim, o núcleo do componente usará essas propriedades para determinar a correta renderização dos seus vários aspectos. Além disso, também é possível criar controles únicos da interface, com representação visual altamente customizada.

Swing é profundamente dependente de mecanismos do tempo de execução. Padrões indiretos de composição permitem que o programa responda a mudanças em sua configuração mesmo durante a sua execução. Por exemplo, uma aplicação baseada no Swing pode mudar sua aparência (também chamada de *look and feel*) em tempo de execução. Adicionalmente, o programador pode implementar sua própria *look and feel*, permitindo mudanças uniformes na aparência de programas já existentes, sem a necessidade de mudanças profundas no código fonte.

A fácil configuração do Swing é resultado da escolha de usar a API Java 2D para se auto-construir, ao invés de utilizar construtores de interface de usuário nativos do sistema operacional. Dessa forma, um componente Swing não tem um correspondente nativo,

possibilitando que ele mesmo se renderize de qualquer modo possível através das API's gráficas.

Entretanto, no seu núcleo, todo componente Swing depende de um conteiner AWT, pois o JComponent (componente base de todos os outros componentes Swing) estende um conteiner AWT. Isso permite que o Swing se ligue ao framework de gerenciamento de interface gráfica do sistema operacional, possibilitando interações e mapeamentos entre a tela e dispositivos, como movimentos do mouse e pressionamento de teclas do teclado. Assim, por exemplo, todo componente Swing faz sua renderização em resposta a uma chamada do método component.paint(), que é definido no AWT Container. Porém, enquanto um componente AWT delega a construção da interface gráfica ao sistema operacional, um componente Swing é responsável pela sua própria construção.

A biblioteca Swing faz um forte uso do padrão de arquitetura de *software* MVC, que conceitualmente desacopla os dados que são vistos dos controles da interface do usuário, por onde os dados são vistos. Por causo disso, a maioria dos componentes Swing são associados a modelos (que são especificados em termos de interfaces Java), e o programador pode usar diversas implementações padrão ou especificar suas próprias. O *framework* fornece implementações padrão de modelos de interface em todos os seus componentes concretos.

6.3.3 – Relação com AWT

Desde as primeiras versões do Java, somente uma porção do *Abstract Window Toolkit* tem fornecido API's para componentes de interface do usuário, que são independentes da plataforma. No AWT, cada componente é renderizado e controlado por um componente nativo específico do sistema base de controle de janelas. Por isso, componentes AWT são referidos como "pesados" (*heavyweight*). Em contraste, componentes Swing são comumente descritos como "leves" (*lightweight*), pois não requerem alocação de recursos nativos do controle de janelas do sistema operacional.

A maior parte da API Swing é geralmente uma extensão complementar do AWT, ao invés de ser um substituto direto. Na verdade, toda interface "leve" Swing fundamentalmente existe no interior de um componente "pesado" AWT. Isso se deve ao fato de que todo componente *top-level* Swing (como *JApplet*, *JDialog*, *JFrame* e *JWindow*) estende um conteiner *top-leve* AWT. Entretanto, o uso tanto de componentes *heavyweight* como *lightweight* na mesma janela é altamente desencorajado, devido a problemas de incompatibilidade.

A Figura 6.1 apresenta o relacionamento entre as principais classes das API's Swing e AWT, evidenciando a dependência existente entre elas.

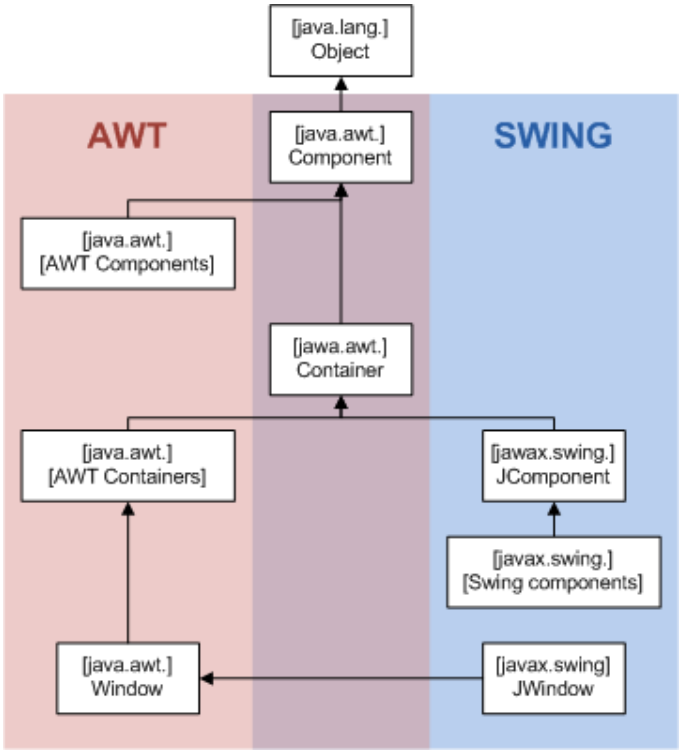

Figura 6.1 – Hierarquia de Classes entre Swing e AWT.

6.4 – INSTALAÇÃO E CONFIGURAÇÃO

A execução de programas feitos em Java é feito através da máquina virtual, conforme dito anteriormente. Como os sistemas operacionais não trazem essa máquina virtual embutida, é necessário que se realize a sua instalação, que é feita instalando a JRE. A versão escolhida foi a 6.0, sendo essa a última versão disponível durante o desenvolvimento do projeto. Os seguintes procedimentos devem ser seguidos para a correta instalação do Java no sistema operacional Puppy Linux:

1) No site http://java.sun.com/, baixar o instalador da JRE e salvá-lo no caminho `/usr/local`. O arquivo terá um nome do tipo "jre-6<version>-linux-i586.bin", onde <version> é a versão do JRE. Por exemplo, se a versão for 1.6.0_10, o arquivo será "jre-6u10-linux-i586.bin".
2) No sistema operacional Puppy Linux, ir até a pasta `/usr/local` no prompt de comando.

3) Digitar:
   `chmod +x jre-6u10-linux-i586.bin`
   Esse comando serve para permitir a execução do instalador.
4) Digitar:
   `./jre-6u10-linux-i586.bin`
   Esse comando descompacta os arquivos e finaliza a instalação.

Para que os programas Java sejam executados de forma adequada, é necessário que se faça as configurações das variáveis de ambiente, seguindo os passos seguintes:

1) Entrar no prompt de comando e digitar:
   `ln -s /usr/local/jre1.6.0_10 /usr/local/java`
   Esse comando cria um link simbólico para facilitar o acesso à pasta onde o Java é instalado.
2) Digitar:
   `touch /etc/profile.d/java.sh`
   Esse comando cria o arquivo java.sh.
3) Digitar:
   `chmod +x /etc/profile.d/java.sh`
   Esse comando serve para permitir a execução do arquivo java.sh.
4) Editar o arquivo recém-criado (java.sh), adicionando as seguinte linhas:

```
#!/bin/sh
JAVA_HOME="/usr/local/java"
CLASSPATH="$JAVA_HOME:$JAVA_HOME/lib"
MANPATH="$MANPATH:$JAVA_HOME/man"
PATH="$PATH:$JAVA_HOME/bin"
export JAVA_HOME CLASSPATH MANPATH PATH
```

5) Reiniciar o computador para confirmar as alterações.

6.5 – RXTX

O Java tem uma grande vantagem frente a outras linguagens de programação que é a possibilidade de um mesmo *software* poder ser executado em diversas plataformas sob uma mesma compilação, não sendo necessário rescrever ou nem mesmo recompilar o *software* para que este esteja disponível em outras plataformas (sejam elas Windows, Linux ou Mac).

Mas toda essa facilidade tem certo custo. Com o Java, a tarefa de chamar API's nativas dos sistemas operacionais, ou ainda fazer comunicação com o *hardware* diretamente, torna-se um pouco mais complexa. Para realizar tais tarefas, a *Sun* e demais empresas envolvidas no desenvolvimento Java disponibilizam diversas API's que possibilitam e facilitam o desenvolvimento de aplicações com a finalidade de comunicação com o *hardware*, como é o caso da API's JavaComm e RXTX, para a comunicação tanto serial quanto paralela (com a RXTX existe também a possibilidade de comunicação via USB).

Originalmente, o suporte à comunicação serial na linguagem Java era feito pela API JavaComm. Porém, o suporte dado a esta API pela *Sun Microsystems* nunca foi completo, sendo inicialmente suportada somente em sistemas Windows, posteriormente sendo suportada em sistemas Linux e finalmente não sendo mais suportada em sistemas Windows na sua última versão, a 3.0.

Devido aos problemas encontrados na API JavaComm e ao seu total abandono pela *Sun*, foi desenvolvida a API RXTX por um grupo de programadores independentes. Essa nova API é baseada na API JavaComm, tendo a vantagem de possuir total suporte para ambientes Linux, Windows e até mesmo Mac. Durante a realização desse projeto, a API RXTX encontrava-se na versão 2.1, sendo essa versão a utilizada na escrita do código do emulador VT100. Ela define todos os métodos necessários a uma correta comunicação serial.

6.5.1 – Instalação e Configuração

A instalação da API RXTX no sistema operacional Puppy Linux é bastante simples, sendo muito parecida com outras distribuições do Linux. Para o correto funcionamento do emulador, basta copiar as bibliotecas da API, correspondentes à porta serial, para o local onde o JRE está instalado. Os passos seguintes indicam os procedimentos a serem seguidos:

1) Baixar as bibliotecas da API RXTX no site http://www.rxtx.org/. Durante o desenvolvimento desse projeto, o arquivo contendo as bibliotecas se chamava "rxtx-2.1-7-bins-r2.zip".
2) Extrair os arquivos "librxtxSerial.so" e "RXTXcomm.jar".
3) Copiar o arquivo "librxtxSerial.so" para `/usr/lib`.
4) Copiar o arquivo "RXTXcomm.jar" para `/usr/local/jre1.6.0_10/lib/ext` (o caminho `jre1.6.0_10` pode variar de acordo com a versão do Java instalada).

CAPÍTULO 7 – SOFTWARE EMULADOR

O emulador do terminal VT100 foi desenvolvido em linguagem Java. Conforme dito ao longo do texto, a escolha por essa linguagem deveu-se ao fato de ela ser multiplataforma, não dependendo da arquitetura do computador escolhido para executá-la. Como consequência, o código fonte do programa torna-se portátil, podendo ser executado em um ambiente com diferentes sistemas operacionais, como o Windows ou o Linux.

Do ponto de vista do *software*, na comunicação serial, o envio de dados é feito de forma síncrona, isto é, os dados são enviados no instante de tempo determinado pelo usuário. Porém, o recebimento dos dados acontece de forma assíncrona, podendo esses dados chegar a qualquer momento.

Nesse contexto, existem duas formas de implementação para o recebimento dos dados. A primeira seria a execução de uma *thread*, sendo ela responsável por verificar constantemente se dados estão disponíveis. A segunda é feita através de um evento causado sempre que novos dados chegam à porta serial. Assim, quando um novo dado é detectado, é chamada uma rotina para o recebimento dos dados, não sendo necessária uma verificação constante. Como a segunda forma é claramente mais vantajosa, por utilizar menos recursos da máquina, ela foi a escolhida no desenvolvimento do programa.

7.1 – COMENTÁRIOS SOBRE O CÓDIGO FONTE

O código fonte do programa tem a função de construir a interface gráfica, assim como realizar todas as configurações necessárias a transmissão de dados, através de parâmetros passados pelo usuário.

Durante a inicialização do programa, a classe principal do código, chamada de *Main*, é responsável por construir toda a interface gráfica, através da chamada de métodos da API Swing. Além de construir a GUI, essa classe também faz a identificação de todas as portas seriais presentes no computador, para a posterior escolha de uma dessas portas pelo usuário. Essa identificação é feita chamando métodos da API RXTX.

Além da classe *Main*, o programa possui uma segunda classe, chamada de *ConfigPorta*. Quando o usuário acessa a opção no menu para realizar uma nova conexão, uma instância dessa classe é criada, aparecendo para o usuário uma janela onde são informadas as configurações da porta serial. Essa classe, então, retorna à classe principal (*Main*) os valores

dos parâmetros escolhidos pelo usuário, como taxa de transmissão e número de bits de dados, sendo esses valores utilizados por métodos da API RXTX para realizar a conexão serial.

Dentro da classe *Main*, diversos métodos são responsáveis pelo correto funcionamento do emulador. A seguir, são listados os principais métodos e suas respectivas funcionalidades:

- *serialEvent*: Responsável pela recepção dos dados que chegam na porta serial;
- *actionPerformed*: Tratador dos eventos gerados pela barra de menu;
- *initport*: Método para a inicialização da porta serial (utiliza os parâmetros recebidos através da classe *ConfigPorta*);
- *sendData*: Método para enviar os dados pela porta serial;
- *contarportas*: Método para contar o número de portas seriais disponíveis;
- *enumerarportas*: Método para identificar todas as portas seriais disponíveis;
- *executarComando*: Método para executar o comando recebido pelo do host;
- *keyTyped*: Tratador do evento gerado ao digitar no teclado. Esse método faz uso do método *sendData*, para enviar pela porta serial o código ASCII correspondente a cada tecla.

7.2 – INTERFACE GRÁFICA

A interface gráfica do programa foi feita utilizando a biblioteca “javax.swing”. Essa biblioteca contém componentes gráficos áreas de textos (*textarea*), botões e menus. Ela fornece uma maneira simplificada para a criação de uma GUI. Os componentes são adicionados ao *frame* do programa (a janela propriamente dita) e organizados de acordo com o layout escolhido pelo programador.

Por se tratar de um emulador de um terminal de vídeo, que tem como função principal somente exibir caracteres na tela, o programa tem uma interface gráfica bastante simples, sendo composta por uma área onde os caracteres são exibidos ou manipulados, uma barra de menus, acima da área de texto, onde estão disponíveis opções de configuração, e uma barra de *status*, abaixo da área de texto, onde são mostradas informações acerca da conexão. A Figura 7.2 mostra a janela principal do programa.

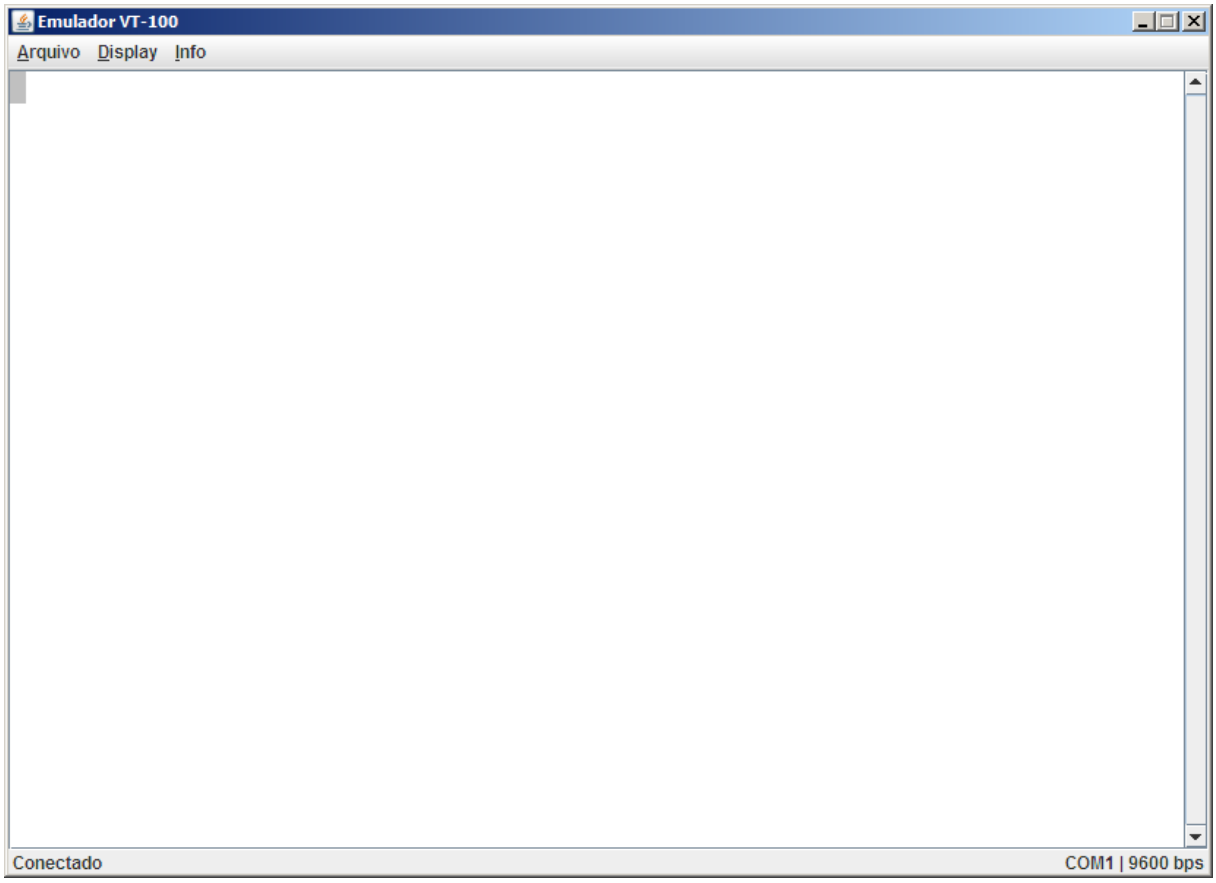

Figura 7.2 – Interface Gráfica do Emulador.

Acessando o menu “Arquivo → Nova Conexão...”, abre-se uma janela onde é possível configurar os parâmetros da comunicação serial, podendo-se escolher a porta a ser utilizada, a taxa de transmissão (em bps), o número de bits de dados, o número de bits de parada e a paridade. Essa janela de configurações é ilustrada na Figura 7.3.

Figura 7.3 – Janela para configurar a conexão serial.

Após a escolha dos parâmetros da comunicação serial, são exibidas na barra de *status*, na parte inferior da interface do programa, a porta e a taxa de transmissão escolhidas. Na figura mostrada anteriormente, é possível visualizar que o programa estava se comunicando através da porta “COM1”, com uma taxa de 9600 bps.

Além de poder realizar configurações sobre a transmissão serial, a barra de menus possibilita configurar a janela de acordo com as preferências do usuário. Acessando o item “Display → Cor”, são disponibilizadas duas opções relacionadas à área de textos, podendo configurá-la com um fundo branco, onde os caracteres são escuros, ou inverter a tela, isto é, configurá-la com um fundo escuro, onde os caracteres são brancos. Já o item “Display → Tamanho”, disponibiliza duas opções relacionadas ao número de linhas e colunas mostradas na tela, sendo elas: 24x80 e 14x132.

Por último, acessando o item “Info → Sobre...” é apresentada uma tela com informações sobre o programa, sobre a versão do Java instalado e sobre o sistema operacional em execução. Essa tela é mostrada na Figura 7.4, onde o emulador estava sendo executado em um sistema com Windows XP.

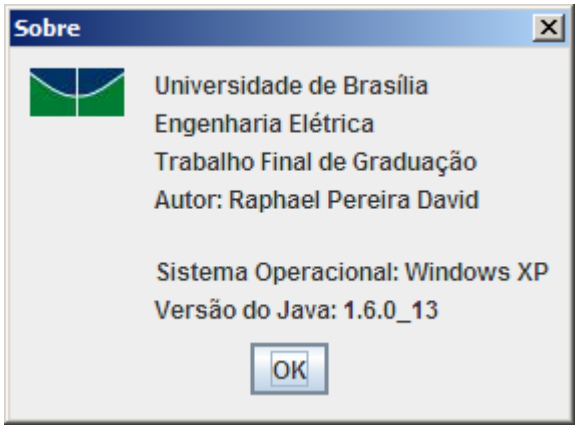

Figura 7.4 – Janela com informações do programa e do sistema.

7.3 – RESULTADOS OBTIDOS

A realização desse projeto visava desenvolver um *software* capaz de emular o terminal de vídeo VT100, implementando todos os comandos que podem ser interpretados pelo terminal, e executando tal emulador em um computador de baixo custo, com especificações de *hardware* limitadas. Assim, para organizar e simplificar a execução desse projeto, o trabalho foi dividido em algumas etapas.

Primeiramente, objetivou-se a escolha do computador de baixo custo. Pensou-se no uso de um computador antigo, porém essa solução traria o inconveniente do tamanho dos gabinetes. Assim, decidiu-se pelo uso de um Mini-PC, com especificações de *hardware* razoáveis e com a vantagem de possuir um tamanho diminuto.

Em uma segunda etapa, focou-se a escolha do sistema operacional que pudesse ser executado no Mini-PC e que, além disso, fosse capaz de prover suporte à linguagem de programação Java. Dessa forma, foi adotado o sistema de código aberto Puppy Linux, que possui todos os atributos necessários já citados.

Após a escolha de todos os elementos básicos do projeto, realizaram-se testes para verificar a capacidade de processamento do Mini-PC, quando executando programas de uso comum, como processadores de texto, no sistema Puppy Linux. A princípio, apesar de ter sido detectada certa lentidão nas respostas do computador, tal lentidão não foi caracterizada como sendo prejudicial ao projeto.

Assim, iniciou-se a escrita do código do programa emulador propriamente dito. Primeiramente, implementou-se uma interface gráfica simples com o intuito de se realizar as primeira conexões seriais. Nesse ponto, encontrou-se alguma dificuldade, pois a API RXTX, conforme comentado anteriormente, não possui uma documentação apropriada, dado o seu desenvolvimento independente. Porém, com a ajuda de fóruns na internet, foi adquirido conhecimento suficiente para se implementar uma correta conexão serial.

Com o andamento do projeto, e o aprendizado das características do terminal VT100, foram sendo implementados os comandos que seriam interpretados pelo programa. O funcionamento dos comandos foi interpretado a partir do manual do usuário do VT100. Nesse ponto do desenvolvimento do emulador, decidiu-se priorizar a implementação dos comandos principais, como os que manipulam os dados na tela do terminal, em detrimento de comandos que, a princípio, não fazem sentido serem implementados, como os que mandam um sinal sonoro ou invocam um auto-teste de *hardware* no terminal.

Alguns comandos implementados pelo programa tem a sua execução um pouco demorada no Mini-PC, sobretudo comandos que alteram significativamente o conteúdo da tela, onde se pode citar o comando que apaga todos os caracteres. Porém, no geral, a maioria dos comandos é executada em um tempo adequado.

7.4 – DESENVOLVIMENTOS FUTUROS

Findado o trabalho, pode-se considerar que o objetivo do projeto foi atingido, mesmo que parcialmente, pois, apesar de não terem sido implementados todos os comandos possíveis, os comando principais foram implementados com sucesso. Por isso, ficam registradas aqui as possíveis implementações adicionais ao projeto que, ou por falta de tempo ou de recursos, não puderam ser concretizadas.

Primeiramente, será necessário o término da implementação dos comandos restantes, além de pequenas correções ou adaptações nos comandos já implementados. Como exemplo de uma correção a ser feita, pode-se citar os comandos *Index* e *Reverse Index*, que no programa emulador não realizam a rolagem da tela quando o cursor se encontra na última ou na primeira linha, respectivamente.

Opcionalmente, pode-se substituir o Mini-PC por um computador com processamento mais rápido, caso o usuário veja essa necessidade. Além disso, é possível efetuar a troca do sistema operacional, devido à portabilidade intrínseca do *software* emulador.

CAPÍTULO 8 – CONCLUSÃO

O *software* emulador do terminal de vídeo VT100 foi desenvolvido em linguagem de programação Java e executado em um computador de baixo custo, denominado Mini-PC. Ao término do projeto, verificou-se que as principais funções do terminal de vídeo VT100 foram implementadas.

A linguagem Java foi escolhida visando a total portabilidade do programa relacionada ao sistema operacional. Assim, o *software* não fica restrito a um único sistema operacional, tendo o usuário a liberdade de executar o emulador no sistema de sua preferência.

O uso da API RXTX para a realização das conexões seriais permitiu a configuração de tais conexões de forma adequada, apesar da dificuldade encontrada de se utilizar os métodos contidos nessa API, devido a pouca documentação disponível sobre ela. Porém, essa dificuldade foi contornada, pesquisando-se em fóruns especializados na internet.

Os principais comandos do terminal VT100 foram programados no emulador. Apesar de esses comandos executarem funções aparentemente simples, como apagar ou manipular dados na tela, a implementação de cada um desses comandos demandou um grande aprendizado da linguagem de programação Java. Tal aprendizado também foi necessário no momento de se construir a interface gráfica do programa, feita com o uso de métodos da API Swing. Essa interface possui menus e janelas para a configuração do terminal que, apesar de não serem elementos presentes nos terminais antigos, foram usados para facilitar o seu manuseio. Ainda assim, a interface foi desenvolvida de modo que o usuário habituado com os antigos terminais não sentisse grandes diferenças no seu uso.

Além das dificuldades encontradas na programação do emulador, houve grande dificuldade na obtenção de referências bibliográficas sobre os terminais de vídeo, tendo sido encontrado apenas o manual do usuário do terminal VT100.

Apesar de o emulador ter sido executado satisfatoriamente no Mini-PC, pode haver a necessidade de se utilizar um computador com especificações mais modernas, de modo a tornar a execução do programa mais ágil, no caso de haver grandes volumes de dados a serem tratados pelo *software*.

O resultado final do projeto foi satisfatório, tendo em vista a implementação de várias funções do terminal VT100, além do grande aprendizado da linguagem Java, e levando em consideração as dificuldades encontradas durante a sua execução.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] DEC. **VT100 User Guide.** 3. ed. Estados Unidos da América: Digital Equipment Corporation, 1981.

[2] DEITEL, H. M.; DEITEL, P. J. **Java How to Program.** 6. ed. Estados Unidos da América: Prentice-Hall, 2004.

[3] ZELENOVSKY, Ricardo; MENDONÇA, Alexandre. **PC: um Guia Prático de Hardware e Interfaceamento.** 3. ed. Rio de Janeiro: MZ Editora Ltda., 2002.

[4] E-WAY. **TU System Manual.** v2. Taiwan: e-Way Technology Systems Corp., 2006.