Ministério da
Ciência e Tecnologia

INPE-16664-NTC/333

DESCRIÇÃO DE SUB-ROTINAS EM LINGUAGEM C PARA TROCA DE VARIÁVEIS VIA PORTA SERIAL ENTRE PROGRAMAS EM EXECUÇÃO SIMULTÂNEA

Suely Silva
Paulo Giácomo Milani

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.04.13.01>

INPE
São José dos Campos
2010

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-16664-NTC/333

DESCRIÇÃO DE SUB-ROTINAS EM LINGUAGEM C PARA TROCA DE VARIÁVEIS VIA PORTA SERIAL ENTRE PROGRAMAS EM EXECUÇÃO SIMULTÂNEA

Suely Silva
Paulo Giácomo Milani

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.04.13.01>

INPE
São José dos Campos
2010

RESUMO

Este documento descreve um conjunto de sub-rotinas para a realização de transferência de dados via porta serial entre programas em execução simultânea. A motivação para o desenvolvimento destas rotinas foi possibilitar a troca periódica de informações entre programas de simulação de sistemas de controle com processamento distribuído. Os dados a serem transferidos são constituídos de variáveis do tipo real ou inteiro que são ajustadas como uma cadeia de bytes para serem transferidas através de uma porta serial.

ABSTRACT

This document describes a set of routines for data transfer via Serial Port between two programs running at the same time. The motivation for the development of these routines was to provide the periodic change of information between programs for control systems simulation with disributed processing. The data to be transferred consists of integer or floating-point variables that are arranged in a chain of bytes to be transferred through a Serial Port.

SUMÁRIO

1. INTRODUÇÃO **9**

2. ESPECIFICAÇÕES **9**

3. ACESSO ÀS PORTAS SERIAIS EM UM SISTEMA LINUX **9**

4. CONFIGURAÇÃO DA PORTA SERIAL **10**

5. CONTROLE DE FLUXO DE SINAIS **10**

6. DESCRIÇÃO DAS ROTINAS DESENVOLVIDAS **11**

7. DESCRIÇÃO DAS FUNÇÕES DA BIBLIOTECA PADRÃO UTILIZADAS **13**

8. CONFIGURAÇÃO UTILIZADA **17**

9. FORMA DE UTILIZAÇÃO **18**

10. CONCLUSÃO E SUGESTÕES **18**

REFERÊNCIAS **20**

APENDICE A **21**

1. INTRODUÇÃO

As rotinas descritas neste documento foram desenvolvidas com a finalidade principal de serem utilizadas como ferramenta para troca de informações entre os programas que fazem parte do *software* de simulação do Sistema de Controle da Plataforma Multimissão (PMM) sendo desenvolvida pela divisão de Engenharia e Tecnologia Espaciais (ETE) do INPE. As rotinas foram desenvolvidas de acordo com especificações sobre o protocolo de comunicação a ser utilizado, o tipo de variáveis a serem transferidas e o comportamento dos programas envolvidos na transferência de dados.

O conjunto é formado por cinco rotinas: quatro utilizadas para manipulação da porta serial e uma para gerenciamento de um intervalo de tempo de espera utilizado para que as transferências sejam realizadas dentro de um cronograma de tempo definido.

2. ESPECIFICAÇÕES

Foi especificado que a comunicação entre os programas de simulação deveria ser realizada através de porta serial utilizando o protocolo RS232.

O programa de simulação da dinâmica da plataforma deveria interagir com o programa de simulação do controlador de forma completamente passiva, ou seja, o programa de simulação deveria sempre colocar à disposição na porta serial os dados relativos à dinâmica independente do estado do programa de simulação do controle para desta forma manter um paralelo com os dados disponibilizados por sensores no barramento do computador de bordo. Da mesma forma a intervalos de tempo determinados o programa de simulação da dinâmica deveria ler os dados da porta serial independente das ações de atualização desses dados por parte do programa de simulação do controle.

Os dados a serem transferidos são compostos de variáveis do tipo inteiro ou real que deverão ser ordenadas em um formato a ser definido.

3. ACESSO ÀS PORTAS SERIAIS EM UM SISTEMA LINUX

O Linux como todo sistema compatível com o padrão POSIX realiza o acesso às portas seriais através de arquivos de dispositivo (*device files*). Cada porta serial tem um arquivo ou mais de dispositivo associado a ela. Estes arquivos encontram-se no diretório /dev. O acesso é feito pela abertura do arquivo de dispositivo correspondente à porta serial que se quer utilizar. As operações de leitura e escrita são então realizadas através das mesmas funções para manipulação de arquivos e suas particularidades são especificadas pelos argumentos de cada função.

A abertura do arquivo para acesso à porta serial é realizada através da função *open( )*; leituras e escritas são realizadas com as funções *read( )* e *write( )* respectivamente e após a finalização das operações desejadas deve-se fechar o arquivo aberto associado à porta com a função *close( ).* Os protótipos destas funções e uma explicação resumida sobre sua utilização encontram-se na seção 7.

4. CONFIGURAÇÃO DA PORTA SERIAL

No Linux a especificação das características da transmissão a ser realizada através da porta serial é feita de acordo com a interface para terminal POSIX. Esta utiliza uma estrutura para definição da configuração a ser utilizada. Esta estrutura, denominada *termios* e as funções de controle sobre a mesma são definidas no arquivo *termios.h*.

A descrição detalhada de toda a interface POSIX pode ser encontrada nos diversos manuais do Sistema UNIX. Sweet, 2004 em seu guia para programação da porta serial para Sistemas compatíveis com o POSIX realiza a descrição das funções de controle da estrutura *termios*.

Abaixo segue uma descrição resumida dos membros mais utilizados da estrutura *termios*.

tcflag_t *c_iflag* : estabelece os modos de entrada. Estes modos descrevem o controle básico de entrada do terminal;

tcflag_t *c_oflag* : define os modos de saída. Estes modos especificam o tratamento do sistema para saída.

tcflag_t *c_cflag* : define os modos de controle. Estes modos descrevem o controle de hardware do terminal.

tcflag_t *c_lflag* : define os modos locais. Estes modos controlam aspectos de mais alto nível do processmento das entradas, tal como sinais de eco e escolha do modo canônico ou não canônico.

cc_t *c_cc[NCCS]* : estabelece caracteres de controle. Estes caracteres especiais de controle são definidos pelos valores armazenados nos elementos do vetor *c_cc[ ]*. Por exemplo, no modo não canônico de processamenteo da entrada, os valores contidos nos elementos *c_cc[VMIN]* e *c_cc[VTIME]*, onde VMIN e VTIME estão definidos no arquivo *termios.h* são utilizados para realizar um controle temporal dos dados de entrada, como tempo de espera por caracteres e controle do número mínimo de caracteres a serem recebidos.

Para realizar a configuração da porta são utilizadas as funções POSIX *tcgetattr*( ) e *tcsetattr*( ). A primeira permite o acesso à configuração em vigor e através da segunda ajusta-se uma nova configuração.

5. CONTROLE DE FLUXO DE SINAIS

Para assegurar que durante a atualização do buffer de transferência de dados da porta serial este não seja acessado para leitura pelo outro computador é necessário realizar o controle do fluxo de sinais. As rotinas desenvolvidas executam este controle através da habilitação e desabilitação das linhas RTS e CTS. Este controle é realizado por meio da função *ioctl*( ).

6. DESCRIÇÃO DAS ROTINAS DESENVOLVIDAS

iIniSerPort( )

Protótipo: **int** iIniSerPort(***unsigned char*** *ucDevNumber*)

Descrição: Esta função abre uma porta serial com uma configuração específica.

Funções que utiliza: sprintf( ), open( ), tcgetattr( ), tcflush( ), tcsetattr( );

Parâmetros:

ucDevNumber é uma variável do tipo caracter para indicar qual porta serial deverá ser aberta. O caracter representado por esta variável será encadeado à string /dev/ttyS para identificar a porta a ser aberta. Os valores podem ser 0,1,2,3,etc. de acordo com a definição das portas na máquina utilizada.

Retorno:

0 se a porta serial especificada foi aberta com sucesso;
1 se houve falha na abertura da porta;
2 se não foi possível adquirir a configuração da porta;
3 se não foi possível reconfigurar a porta.

Erro: sem especificação.

ireadData( )

Protótipo: **int** ireadData(***unsigned char*** **ucflag*, ***int*** *iVarDesc[]*, ***unsigned char*** *ucVarDesc[]*, ***int*** *irData[]*, ***double*** *drData[]*)

Descrição: Lê dados do buffer da porta serial aberta com iIniSerPort( ).

Funções que utiliza: ioctl( ), read( ).

Parâmetros:

ucflag é uma variável do tipo caracter reservada para *flag* de controle.

iVarDesc[] é um vetor de inteiros que definirá juntamente com o vetor *ucVarDesc[]* quantas variáveis inteiras e reais serão transferidas e qual a disposição dessas variáveis no buffer de transferência.

ucVarDesc[] é um vetor de caracteres 'i' ou 'd' que indica o tipo das variáveis a serem transferidas. Por exemplo se iVarDesc[0]=5, iVarDesc[1]=1, iVarDesc[2]=3, ucVarDesc[0]='d', ucVarDesc[1]='i', ucVarDesc[2]='d', então o *buffer* de transferência conterá a seguinte seqüência de variáveis: uma variável do tipo inteiro para flag de controle; cinco variáveis tipo *double*; uma variável tipo inteira e três variáveis tipo double.

irData[] é um vetor onde serão armazenadas as variáveis inteiras extraídas do buffer de transferência de acordo com os vetores de descrição das variáveis.

drData[] é um vetor onde serão armazenadas as variáveis reais tipo double extraídas do buffer de transferência de acordo com os vetores de descrição das variáveis.

Retorno:

0 em qualquer caso sem verificação de erro.

Erro: sem especificação.

iwriteData( )

Protótipo: **int** iwriteData(***unsigned char*** *ucflag*, ***int*** *iVarDesc[]*, ***unsigned char*** *ucVarDesc[]*, ***int*** *iwData[]*, ***double*** *dwData[])*

Descrição : Escreve dados no buffer da porta serial aberta com iIniSerPort( ).

Funções que utiliza: ioctl( ), write( ).

Parâmetros:

ucflag é um caracter reservado para flag de controle.

iVarDesc[] é um vetor de inteiros que definirá juntamente com o vetor ucVarDesc[] quantas variáveis inteiras e reais serão transferidas e qual a disposição dessas variáveis no buffer de transferência.

ucVarDesc[] é um vetor de caracteres 'i' ou 'd' que indica o tipo das variáveis a serem transferidas. Tem as mesmas características do parâmetro de mesmo nome da função ireadData( ).

irData[] é um vetor onde serão armazenadas as variáveis inteiras que serão montadas na cadeia de bytes que será colocada no buffer de transferência de acordo com os vetores de descrição das variáveis.

drData[] é um vetor onde serão armazenadas as variáveis reais tipo double que serão montadas na cadeia de bytes que será colocada no buffer de transferência de acordo com os vetores de descrição das variáveis.

Retorno:

0 se não houver mensagem de erro de escrita.

1 se a operação de escrita gerou erro.

Erro: sem especificação.

iFinSerPort( )

Protótipo**: int** iFinSerPort ***(void)***

Descrição: Fechar a porta serial aberta.

Funções que utiliza: close( )

Parâmetros: esta função não utiliza nenhum parâmetro. Utiliza o valor de uma variável estática para identificar a porta a ser fechada.

Retorno:

0 se não houver erro na operação.

1 se ocorrer algum erro.

Erro: sem especificação.

itimeMan( )

Protótipo: **int** itimeMan(**double** **dinicTime,* **double** *dWaitInterval*)

Descrição: Comanda uma espera de determinado período de tempo desde um tempo inicial especificado. Atualiza a variável *dinicTime* depois de transcorrido o intervalo de tempo determinado em *dWaitInterval*.

Funções que utiliza: gettimeofday( ).

Parâmetros:

dinicTime é um valor em milisegundos que representa o instante inicial a partir do qual será feita a contagem de tempo até transcorrer o intervalo de espera determinado pelo parâmetro *dWaitInterval*. *dinicTime* é calculado através da chamada da função gettimeofday( ) e assume valores de tempo que têm como referência o dia primeiro de Janeiro de 1970 à 0:0 hora. O usuário poderá iniciar *dinicTime* com o valor do tempo transcorrido em milisegundos desde a data mencionada ou iniciá-lo através da chamada de itimeMan( ) passando valor nulo para *dinicTime*. Se *dinicTime* for 0 a rotina retornará imediatamente e o intervalo de tempo será ignorado. Neste caso esta chamada é utilizada apenas para atribuir a este parâmetro o valor do tempo de origem da contagem para a seqüência de intervalos de espera.

dWaitInterval é o intervalo de tempo de espera.

Retorno:

0 ao final do intervalo de espera.

Erro: sem especificação de erro.

7. DESCRIÇÃO DAS FUNÇÕES DA BIBLIOTECA PADRÃO UTILIZADAS

As decrições nesta seção foram apresentadas de forma resumida e foram compiladas de diferentes versões de Manuais de Referência do Sistema UNIX. A intenção nesta seção é esclarecer a utilização destas funções dentro das rotinas desenvolvidas no trabalho aqui apresentado. Por esta razão na descrição das funções não serão comentados parâmetros sem efeito na cahmada das funções pelas rotinas desenvolvidas. Detalhes sobre as funções podem ser obtidos em qualquer Manual de Referência UNIX. Por exemplo a Sun MicroSystems mantém em sua página documentação *online* de referência sobre este Sistema. Na seção Referências são listados alguns encdreços que mantêm manuais para consulta.

sprintf( )

#include <stdio.h>

Protótipo: **int** sprintf(***char ****str****, const char ****format****,* ...)

Descrição: escreve a saída seguida pelo byte "\0"na variável *str* de acordo com um formato determinado por *format*.

Parâmetros:
str: variável reservada para acomodar uma cadeia de caracteres.
format especifica como os argumentos subseqüentes são convertidos para saída.

Retorno:
O número de caracteres escritos.
Um valor negativo se houver erro.

open( )

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

Protótipo: **int** open(***const char*** **path*, ***int*** *flags*);

Descrição: Promove a abertura de um arquivo e associa a este um descritor para ser usado em operações de outras funções sobre o mesmo.

Parâmetros:
path é uma variável do tipo ponteiro que aponta para o nome do arquivo aberto.
flags estabelece os modos de acesso ao arquivo. É formado por operações de "ou" lógico entre uma série de *flags* descritos em *fcntl.h*.

Retorno:
Se bem sucedida a chamada da função retornará o descritor do arquivo.
Se houver falha será rertornado o valor –1 e a variável *errno* será ajustada com a indicação de erro apropriada.

tcgetattr( )

#include <termios.h>
#include <unistd.h>

Protótipo: **int** tcgetattr**(*int fd, struct termios ****termios_p* **);**

Descrição: Obtém os parâmetros associados ao arquivo descrito por *fd* e armazena estes parâmetros em uma estrutura *termios*

Parâmetros:
fd é o descritor do arquivo.
termios_p é um ponteiro para a estrutura *termios*. Esta estrutura contém informações que descrevem uma interface geral de controle para portas de comunicação assíncrona.

Retorno:
0 em caso de sucesso.
-1 em caso de falha e ajuste da variável *errno* para indicar o erro

tcflush( )

#include <termios.h>
#include <unistd.h>

Protótipo: **int** tcflush **(*int* *fd*, *int* *queue_selector*)**

Descrição: Descarta dados existentes no objeto referenciado pelo descritor fd.

Parâmetros:
fd é o descritor do objeto onde será realizada a operação.
queue_selector indica o tipo de dado a ser descartado dependendo de seu valor. Se o valor for **TCIFLUSH** serão descartados dados recebidos porém não lidos. Se for **TCOFLUSH** serão descartdos dados escritos no buffer de saída mas não transmitidos. E se for **TCIOFLUSH** ambos os tipos serão descartados.

Retorno:
0 em caso de sucesso.
-1 em caso de falha e ajuste da variável *errno* com valor apropriado indicando o tipo de erro.

tcsetattr( )

#include <termios.h>
#include <unistd.h>

Protótipo: **int** tcsetattr *(**int** fd, **int** optional_actions, **struct termios** *termios_p*)*

Descrição: Ajusta os parâmetros associados com o terminal descrito por *fd* de acordo com a estrutura *termios_p.*

Parâmetros:
fd é o descritor do terminal sobre o qual será executada a operação.
optional_actions especifica quando será executada a operação.
termios_p é a estrututura que contém a nova configuração a ser usada para o terminal.

Retorno:
0 em caso de sucesso.
-1 em caso de falha e ajuste da variável *errno* com valor apropriado indicando o tipo de erro.

ioctl( )

#include <sys/ioctl.h>

Protótipo: **int** ioctl(***int*** *fd*, ***int*** *request*, **...**/* *arg* */)

Descrição: Executa o comando de operação E/S declarado pelo parâmetro request sobre o dispositivo descrito por fd. O significado do terceiro argumento e o efeito da função dependem do valor de *request*.

Parâmetros:
fd é o descritor para o dispositivo sobre o qual deve atuar a função.
request determina o comando a ser execcutado sobre o dispositivo.
arg deve ser um inteiro ou um ponteiro para uma estrutura de dados específica relacionada ao dispositivo descrito por *fd*.

Retorno:
0 em caso de sucesso.
-1 em caso de falha e ajuste da variável *errno* com valor apropriado indicando o tipo de erro.

read( )

#include <unistd.h>

Protótipo: **ssize_t** read*(**int** fd, **void *** buf, **size_t** count*)

Descrição: Lê determinado número de bytes do arquivo descrito por *fd* e armazena no buffer identificado pelo segundo argumento da função.

Parâmetros:
fd é o descritor do arquivo onde será feita a leitura.
buf é o *buffer* onde será armazenado o conteúdo da leitura realizada.
count define a quantidade de bytes a ser lida.

Retorno:
Número de bytes lidos em caso de sucesso.
-1 se ocorrer falha e ajuste da variável *errno* com o valor adequado de indicação de erro.

write( )

#include <unistd.h>

Protótipo: **ssize_t** write*(**int** fd, **const void *** buf, **size_t** count*)

Descrição: Escreve determinado número de bytes em um arquivo associado ao descritor *fd*. Os bytes escritos são retirados de um *buffer* que começa em *buf*.

Parâmetros:
fd é o descritor do arquivo onde os bytes serão escritos.
buf é o ponteiro para o *buffer* onde estão os dados que deverão ser escritos no arquivo de destino.
count define a quantidade de bytes a ser escrita.

Retorno:
Número de bytes escritos em caso de sucesso.
-1 se ocorrer falha e ajuste da variável *errno* com o valor adequado de indicação de erro.

close( )

#include <unistd.h>

Protótipo: **int** close(***int*** *fd*)

Descrição: Fecha um descritor de arquivo rompendo sua associação com o mesmo e liberando-o para identificação de outro arquivo.

Parâmetros:
fd é o descritor a ser dissociado.

Retorno:
0 se a operação for bem sucedida.
-1 se houver erro.

gettimeofday( )

#include <sys/time.h>

Protótipo: **int** gettimeofday(***struct timeval** *tv, **struct timezone** *tz* **);**

Descrição: Obtém do sistema o tempo corrente expresso em segundos e microsegundos passados desde 1º de Janeiro de 1970 à 00:00 UCT (Universal Coordinated Time).

Parâmetros:
tv é uma estrutura **timeval** especificada em *time.h* que inclui os seguintes membros:
long *tv_sec*, segundos desde 1º de Janeiro de 1970;
long *tv*_usec, *microsegundos*.
tv é uma estrutura para região de tempo (*timezone*) atualmente sem utilização.

Retorno:
0 em caso de sucesso.
-1 se ocorrer falha e ajuste da variável *errno* com o valor adequado de indicação de erro.

8. CONFIGURAÇÃO UTILIZADA

Na abertura da porta serial o modo de acesso ao arquivo associado foi definido na função *open( )* por *oflag* = O_RDWR | O_NOTTY indicando por meio de **O_RDWR** que o arquivo aberto será para leitura e escrita e por meio de **O_NOCTTY** que o terminal correspondente ao arquivo aberto não será controlador do processo.

Através do ajuste dos valores nos membros da estrutura *termios* a seguinte configuração foi utilizada na transferência dos dados entre os dois programas de sinulação:

.c_cflag = B115200 | CS8 |CREAD | CLOCAL | CRTSCTS;

B115200: *baud rate* de 115 200;

CS8: 8 *bits* de dados;

CREAD: habilitação para recepção

CLOCAL: linhas locais

CRTSCTS: habilitação de controle de fluxo via *hardware*

.c_oflag=0 indica dados de saída não processados;

.c_lflag=0 indica o modo não canônico ou dados de entrada não processados;

.c_cc[VMIN] = 0 e .c_cc[VTIME] = 0 indicam retorno imediato ao usuário. O mínimo entre o número de caracteres disponíveis e o número requisitado é retornado ao usuário.

9. FORMA DE UTILIZAÇÃO

As rotinas de transferência de dados descritas neste trabalho foram reunidas em uma biblioteca de vínculo estático denominada *libst*.

Para utilização das rotinas deve-se fazer o *link* desta bibiloteca com o programa de chamada das mesmas e colocar os dados a serem transferidos no formato especificado na descrição feita da seção 6. Deve-se ainda incluir o arquivo *st.h* na área de cabeçallho do programa fonte que irá chamar as rotinas. A linha de comando mostrada abaixo gera o executável de nome ***teste*** de um programa fonte ***teste.c*** que faz chamada às rotinas da biblioteca *libst*. O compilador utilizado é o *gcc* e considera-se que os arquivos *st.h* e *libst.a* se encontrem no diretório local.

```
$ gcc -o teste teste.c -L. -lst –lm
```

A ligação entre os dois computadores envolvidos na operação de transferência de dados deve ser realizada por um cabo de comunicão serial do tipo Null Modem. O apêndice A mostra um esquema para o cabo serial utilizado na comunicação entre os dois computadores que executam os programas que utilizam as rotinas da biblioteca *libst*.

10. CONCLUSÃO E SUGESTÕES

Para testar as rotinas de transferência de dados foram criados dois programas que fazem iterativamente várias chamadas às mesmas e realizam trocas de informações entre dois computadores utilizando um intervalo de espera entre as trocas. A execução destes programas e a posterior comparação entre os dados enviados e recebidos por cada programa mostrou que as rotinas desenvolvidas executaram as transferências de dados corretamente dentro do cronograma imposto pela utilização da rotina de gerenciamento dos intervalos de tempo de espera.

Uma vez testado o funcionamento das rotinas com estes dois programas elas foram utilizadas para transferência de dados entre dois programas que realizam a simulação do controle da Plataforma SubOrbital 1 durante seu período de estabilização. Esta simulação foi anteriormente realizada em programa único para simulação da dinâmica e da lei de controle e foi descrita em Milani e Silva, 2006. As rotinas de transferência de dados da biblioteca *libst* foram utilizadas para separar a simulação entre duas máquinas diferentes aproximando a simulação de um caso mais real. Neste caso a simulação é realizada por dois programas, um que sinula a dinâmica da Plataforma e o outro que simula o controle. As velocidades angulares atualizadas no primeiro programa e os torques de controle

calculados no segundo são transferidos utilizando as rotinas de transferência da biblioteca *libst*. Este último trabalho é descrito em Silva, 2007.

A comparação dos resultados obtidos com a simulação realizada em programa único e a obtida com a utilização dos programas divididos em dois computadores demonstrou a adequação das rotinas de transferência desenvolvidas para a troca de informações necessárias.

Não foram detectados erros de transferência ou de perda de dados nas simulações realizadas, no entanto como as rotinas da *libst* não realizam a verificação da validade dos dados recebidos, deixando esta tarefa para a aplicação que faz as chamadas das rotinas, sugere-se uma posterior implementação, se necessário, de código específico para este fim, bem como o desenvolvimento de uma rotina para gerenciamento dos erros.

REFERÊNCIAS

SWEET, M. R.. **Serial Programming Guide for POSIX Operating Systems**. Copyright 1994-2004 by Michael R. Sweet, 44 p..

MILANI, P. G.; SILVA, S.. **Simulação em Tempo Real do Comportamento Dinâmico da Plataforma Suborbital 1 Durante Período de Estabilização**. São José Campos: INPE, 2006. 26 p..

SILVA, S.. **Simulação em Tempo Real do Controle da Plataforma Suborbital 1 com Implementação do Controle e dinâmica em Máquinas Distintas Interagindo Via Interface RS232**. São José dos Campos: INPE, 2007. 27 p..

Manuais *online*:

http://docs.sun.com/app/docs/coll/40.10

http://ou800doc.caldera.com/en/Navpages/index.html

http://www.rt.com/man/

http://www.opengroup.org/onlinepubs/009695399/

APENDICE A

Cabo Null Modem com suporte a controle de fluxo por meio de *hardware*:

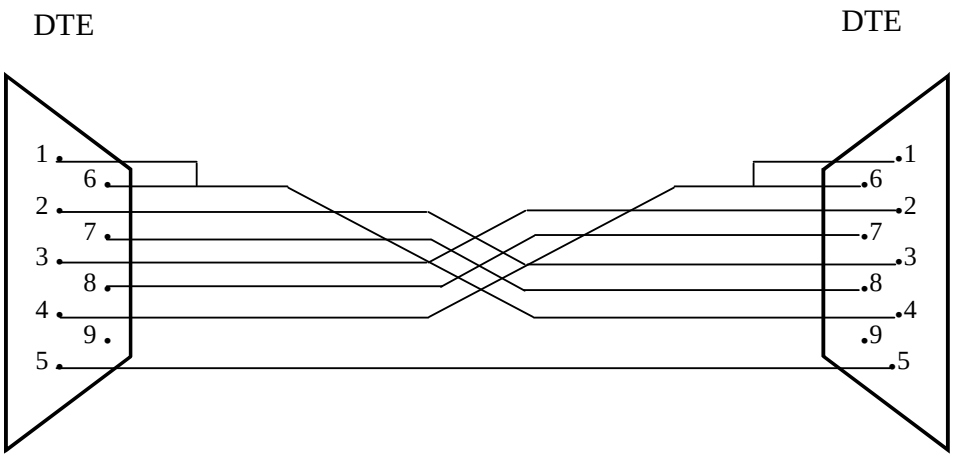

Figura 1. Pinagem do Cabo Null Modem com Suporte a Controle de Fluxo

Pinos de dados:

TXD (pino 3): Saída de dados serial
RXD (pino 2): Entrada de dados serial

Pinos para Handshake:

RTS (pino 7): Request to send
CTS (pino 8): Clear to send
DSR (pino 6): Data Set Ready
DCD (pino 1): Data Carrier Detect
DTR (pino 4): Data Terminal Ready

Terra:

GND (pin 5): Ground

Pino para indicação de campanhia

RI (pino 9): Ring Indicator