NAME

jpnevulator - Just another serial sniffer

SYNOPSIS

jpnevulator [*OPTION*]... <*FILE*>

DESCRIPTION

jpnevulator is a handy serial sniffer. You can use it to send data on a serial device too. You can read or write from/to one or more serial devices at the same time.

In write (--write) mode data to be sent on the serial device(s) is read from a file or stdin in hexadecimal notation. The input format is FD or 0xFD. Of course all input is treated case-insensitive. Spaces may or may not be included in the input. So DEADBEEF is exactly the same as DE AD BE EF. Data is sent on the serial device(s) line by line.

In read (--read) mode data to be read from the serial device(s) is written to a file or stdout in hexadecimal notation. Skim through the options for several enhancements in the output. It's even possible to pass(--pass) on the data between the several serial devices.

Mandatory arguments to long options are mandatory for short options too.

Generic options:

-l, **--alias-separator**
: Use the given string as the alias separator. See --tty for more information.

-f, **--file**=*NAME*
: In write mode read the contents of the file given and send them on the serial device(s) and in read mode write the contents of the serial device(s) to the file given.

-h, **--help**
: Shows a brief list of options.

-o, **--count**=*BYTES*
: Exit after reading / writing the given amount of bytes.

-r, **--read**
: Put the program in read mode. This way you read the data from the given serial device(s) and write it to the file given or stdout if none given. See the read options section for more read specific options.

-t, **--tty**=*NAME:ALIAS*
: The serial device to read from or write to. Use multiple times to read/write from/to more than one serial device(s). For handy reference you can also separate an alias from the tty name with a collon ':'. If a collon is for some strange reason part of your device name, you can use the --alias-separator option to specify another separation string. If an alias is given it will be used as the name of the serial device.

-v, **--version**
: Output the version information, a small GPL notice and exit.

-w, **--write**
: Put the program in write mode. This way you read data from a given file or stdin if none given and write it to the serial device(s) given. See the write options section for more write specific options.

Read options:

-a, **--ascii**
: Besides the hexadecimal output also display an extra column with the data in the ASCII representation. Non printable characters are displayed as a dot '.'. The ASCII data is displayed after the hexadecimal data.

-b, **--byte-count**
Besides the hexadecimal output also display an extra column with the current index number of the byte in the output. These numbers are displayed in front of the hexadecimal data. When readin from multiple serial devices at the same time the index number will increase per serial device.

-C, **--control**
Monitor modem control bits (line enable, data terminal ready, request to send, secondary TXD, secondary RXD, clear to send, carrier detect, ring and data set ready) too and notify changes. Use the --control-poll option to specify how often to poll for the bits.

-D, **--control-poll**=*MICROSECONDS*
The control poll is the amount of microseconds to wait in between two checks of the modem control bits if nothing else is happening.

-P, **--pass**
This one passes all the data between the serial devices. Handy if you want to put your serial sniffer in between the serial devices you want to sniff.

-q, **--pty**=*:ALIAS*
The pseudo-terminal device to read from. Use multiple times to read from more than one pseudo-terminal device(s). For handy reference you can also use an alias to name the pty. Make sure it starts with a collon ':'. Use the --alias-separator option if you for some reason don't like to use a collon. If an alias is given it will be used as the name of the pseudo-terminal device.

-e, **--timing-delta**=*MICROSECONDS*
The timing delta is the amount of microseconds between two bytes that the latter is considered to be part of a new package. The default is 100 miliseconds. Use this option in conjunction with the --timing-print option.

-g, **--timing-print**
Print a line of timing information before every continues stream of bytes. When multiple serial devices are given also print the name or alias of the device where the data is coming from.

-i, **--width**=*WIDTH*
The number of bytes to display on one line. The default is 16.

Write options:

-c, **--checksum**
Append a single checksum byte to the line of data written to the serial device(s) chosen. This checksum is a simple modulo 256 addition of all input bytes on a line.

-z, **--crc8**=*POLY*
Append a crc8 checksum to the line of data written to the serial device(s) chosen. Use the optionally given poly as the polynomial. Specify the polynomial as hexadecimal value, as in 0x07 (the default).

-y, **--crc16**=*POLY*
Append a crc16 checksum to the line of data written to the serial device(s) chosen. Use the optionally given poly as the polynomial. Specify the polynomial as hexadecimal value, as in 0xA001 (the default).

-k, **--delay-byte**=*MICROSECONDS*
This delay is an optional amount of microseconds to wait in between every input byte is sent on the serial device(s).

-d, **--delay-line**=*MICROSECONDS*
This delay is an optional amount of microseconds to wait in between every input line is sent on the serial device(s).

-j, **--fuck-up**
This is the special fuck up option. When the calculation of a checksum is chosen (see checksum and crc* options) the checkum will be crippled on purpose. Carefully named after the special Jan Arie de Bruin 'fuck up crc' button.

-n, **--no-send**
Do not actually send the bytes on the serial device(s). Rather pointless, but seemed one day long ago to be a rather handy feature.

-p, **--print**
Besided sending the data on the serial device(s) also write the data to stdout.

-s, **--size**=*SIZE*
The maximum number of bytes per line to send on the serial device(s). The default is 22, coming from back in the Cham2 days of the program.

DIAGNOSTICS

Normally, exit status is 0 if the program did run with no problem whatsoever. If the exit status is not equal to 0 an error message is printed on stderr which should help you solve the problem.

BUGS

Order of bytes broke when reading several tty devices at once

The display of incoming bytes can be broke if you use multiple tty devices to read from. At the moment I do not have a solution for this problem. Since I use select() to watch the several tty devices and after the select() I have to read() them one by one, I can not completely 100% display which bytes came after which on different tty devices. Take the example below:

```
$ jpnevulator --ascii --timing-print --tty /dev/ttyS0 --tty /dev/ttyUSB0 --read
2006-05-30 13:23:49.461075: /dev/ttyS0
00 00 05 3B 0D 00 00 05 ...;....
2006-05-30 13:23:49.461113: /dev/ttyUSB0
00 05 3B 0D 00 00 05 3B 0D ..;....;.
2006-05-30 13:23:49.473074: /dev/ttyS0
3B 0D 00 00 05 3B 0D ;....;.
2006-05-30 13:23:49.473105: /dev/ttyUSB0
00 12 05 06 39 00 12 05 06 39 1F 00 22 80 00 0E ....9....9.....
$
```

And now see the order in which things really got sent on the line:

```
/dev/ttyS0:
00 00 05 3B 0D
/dev/ttyUSB0:
00 00 05 3B 0D
/dev/ttyS0:
00 00 05 3B 0D
/dev/ttyUSB0:
00 00 05 3B 0D
/dev/ttyS0:
00 00 05 3B 0D
/dev/ttyUSB0:
00 00 05 3B 0D 00 12 05 06 39 00 12 05 06 39 ...
```

As you can see /dev/ttyUSB0 receives the echo of all things sent by /dev/ttyS0. This is exactly what happens. But since there does exist a small time between the select() who is happy expressing something is available and the read() who does get the available data, some extra data will be available. I have no idea on how I can use high level system call like select() and read() and be still able to put the bytes in the correct order. Anyone an idea?

AUTHOR

Written by Freddy Spierenburg.

REPORTING BUGS

Report bugs to <freddy@snarl.nl>.

COPYRIGHT