NAME

linecheck, test – debugging tools for term

SYNOPSIS

linecheck [escape ...] < /dev/tty?? >/dev/tty??

test [-d<debug-level>]

DESCRIPTION

You should perform two sets of tests before you ever try to run *term* itself. The *linecheck* program performs a first-order test of the transparency of the link. The *test* program lets you exercise *term* and its clients locally by starting two *term* daemons on the same system.

Linecheck sends packets, once per second, each of which contains one of the 256 possible 8-bit character codes. The diagnostic output helps you determine which characters do not get through. This is important, since almost all of the problems associated with getting *term* running occur because individual characters or character sequences do not get through the serial link or cause other characters to be sent and/or echoed by the link. *Term* can easily be told to avoid sending these characters by listing them in the termrc file, the problem is determining which ones to avoid. *Linecheck* tests all 256 individual characters and helps you determine which are not getting through. It only checks individual characters, not character sequences.

Linecheck must be run on both systems the same way *term* is run. That is, the stdin and stdout should be directed to the serial port while the stderr goes to a log file. Remotely you can type

linecheck 2>/tmp/linecheck.log

to the sh, or

sh -c ’exec linecheck 2> /tmp/linecheck.log’

to the csh. Locally you should escape from your comm program and type something like

linecheck < /dev/modem > /dev/modem 2> /tmp/linecheck.log

to a sh.

You can tell linecheck not to test certain characters by listing their decimal numbers on the command line. For instance, if you know flow-control will get eaten, you can use "linecheck 17 19", and it won’t test those chars. Or, if your link goes through an *rlogin*(1) on the remote end, you will want to put 126 on the command line to escape the ’~’ character. Also, in this case, you’ll want to use

rlogin <system> -8 -L

in a bid to establish a maximally transparent rlogin.

Once *linecheck* has finished running on both ends, you should examine the respective log files on the two systems. These files will contain lines of the form ’<num> sending char’ and ’<num> received valid’, where <num> is the decimal number of the character in the packet. These indicate that the local *linecheck* (the one generating the log file) sent the character or received the character from the other *linecheck.* It’s normal if the ’sending’ and ’received’ numbers are not in sync.

Lines of the form ’Invalid packet: <data>’ are the interesting output. They indicate a packet was received, but was corrupted. This may be due to simple line noise, or to the opacity of the link to a character generated by either the remote or the local *linecheck* program. At least three possible kinds of link opacity can lead to invalid packets. When a character that should be be escaped is sent the link may simply not transmit the character in question, or the link may transmit a different or extra characters. Also, at the same time, the link itself may echo characters back to the system which sent the character in the first place. You must study the log files from both systems to determine which direction of the link causes the problems and which characters should be escaped on which system. Real link problems like these should be repeatable while line noise effects will not.

The bottom of the log file lists characters that linecheck believes should be escaped by the other system in the termrc file. These are characters that the other system sent, but which were not received correctly. If you had no invalid packets then this summary is probably reliable. If there were invalid packets it is possible that *linecheck's* recommendations are wrong. You must examine both logs closely to determine what caused the invalid packets.

If, for some reason, you get stuck out in lala land, and can't kill the remote program, try typing '00000'. That should kill it, and restore your terminal.

Test, when run, establishes two connected instances of *term* on the local system so you may see if they and the clients work. To avoid problems with other programs also called 'test' you should run *test* from the source directory of *term* by typing

./test [-d<debug-level>]

Test opens the files 'local.log' and 'remote.log' in the local directory to record the stderr output of the two term daemons. The –d<debug-level> option sets the debugging level for the test. The termrc file is used to set the other parameters of both daemons. If you need to debug you can use 64 or even 478. Familiarity with the sources is essential if you use debugging.

Once ./test is running you should be able to use the clients from any shell:

trsh

give you a 'remote' shell. You may need to type 'reset ^J' to reset the tty of the new shell (that's Cntl-J).

trsh -s who

will run the who command 'remotely' to list the current users.

tupload -v -f <filename> ... /tmp

will copy the listed files into the /tmp directory and provide CPS statistics.

tredir 4000 23

will map port 4000 to 23 and put itself in the background. Then, typing

telnet 0 4000

should give you a login prompt that works. If you're running X, you should be able to use *txconn* to establish a new display (screen) that maps to the existing screen.

Be sure to run ./test on both the local and remote systems, particularly if the systems use different architectures.

BUGS

Linecheck is too slow. When run with ./test daemons, the *trsh* client may not initialize the 'remote' tty very well.

SEE ALSO

term(1), *term_clients*(1)

AUTHOR

Michael O'Reilly, oreillym@tartarus.uwa.edu.au