Serial port on linux in VirtualBox filters out 0x11 from data stream

I don't know if this problem is because of VirtualBox or because of the serial port configuration. I'm testing a program that interacts with a particular piece of hardware over a serial port. Everything is generally working fine, but in the data stream received from the device, some bytes are not received properly. Specifically 0x11, 0x13 are missing, and 0x0d is received as 0x0a. I have a serial sniffer program watching the serial port and the byte is seen there, but it's not seen in my program that's running in a VirtualBox linux machine. I need to see all binary bytes with no flow control. Here's how I'm opening the serial port:

```
int openPort( char* portname )
{
  int fd;

  fd = open( portname, O_RDWR | O_NOCTTY | O_NDELAY );
  if ( fd == -1 )
  {
      perror( "openPort():" );
      printf( "Unable to open %s\n", portname );
  }
  else
  {
      fcntl( fd, F_SETFL, FASYNC );

      struct termios my_termios;
      struct termios new_termios;

      tcgetattr( fd, &termios_save );
      tcgetattr( fd, &my_termios );
      cfsetispeed( &my_termios, B9600 );
      cfsetospeed( &my_termios, B9600 );
      my_termios.c_cflag &= ~( PARENB | CSTOPB | CSIZE );
      my_termios.c_cflag |= ( CS8 | CREAD );
      my_termios.c_lflag &= ~( ICANON | ECHO | ECHOE | ISIG );
      my_termios.c_oflag &= ~OPOST;
      my_termios.c_cc[VMIN] = 1;
      my_termios.c_cc[VTIME] = 100;   // wait 10 seconds for a character
      tcsetattr( fd, TCSANOW, &my_termios );
      tcgetattr( fd, &new_termios );
      if ( memcmp( &my_termios, &new_termios, sizeof( my_termios ) ) != 0 )
      {
          printf( "Error setting serial port options, aborting\n" );
          close( fd );
          fd = -1;
      }
  }

  return fd;
}
```

I've already looked at various guides to using the serial port, including Serial Programming Guide for POSIX Operating Systems and this is what I've come up with.

c linux serial-port virtualbox

asked Mar 2 '15 at 20:50

JonS
184 13

Where is the reading code? Is the sniffer running inside the VM or on the host? – Eugene Sh. Mar 2 '15 at 20:54

The sniffer is running on the host. The reading code is just a read() call, and that part works fine and thus isn't relevant, so I didn't include it in the question. – JonS Mar 2 '15 at 21:17

No, it doesn't work fine, since it does not read what you expect. Try running some serial sniffer or terminal on the guest machine and see if the data is received fine through the VM layer. – Eugene Sh. Mar 2 '15 at 21:18

Instead of tcsetattr(), then tcgetattr() and a memcmp(), the proper way to do this is to simply check the return code from tcsetattr(). – sawdust Mar 3 '15 at 1:58

2 @sawdust: No, it's not. `tcsetattr()` followed by `tcgetattr()` and `memcmp()` **is** the proper way, because `tcsetattr()` will return success if *any* (instead of *all*) of the settings were successfully applied. See man termios(3) for details. – Nominal Animal Mar 3 '15 at 19:01

2 Answers

`0x11` and `0x13` are the hex values of ASCII `XON (DC1, Ctrl-Q)` and `XOFF (DC3, Ctrl-S)` and are used to provide pacing of the data stream in applications with limited performance.

Either the host serial port, or VirtualBox, is interpreting these (or just filtering them to avoid subsequent interpretation). Set the ports to raw mode in the host machine. You don't say, but if it is some flavour of *nix, then `stty` could help.

answered Mar 2 '15 at 21:18

Pekka
2,766 17 38

Apparently I missed a section in the guide. I had originally copied my init code from some example program which I can't find now, so I don't know if they missed it also, or if I missed it in my copy. Anyways, the problem is the setting of c_iflag, which I didn't change originally. Adding the line:

```
my_termios.c_iflag &= ~( IXON | IXOFF | IXANY | ICRNL | INLCR | IGNCR );
```

Takes care of the issue. Apparently software flow control and CR->NL mapping is enabled by default.

answered Mar 2 '15 at 21:40

JonS
184 13

"Apparently software flow control and CR->NL mapping is enabled by default." -- Regardless of what is enabled by default (or currently configured), you are expected to specify every flag that is necessary for the mode you want. There is a **cfmakeraw()** syscall for assistance in setting up non-canonical mode. – sawdust Mar 3 '15 at 2:03

Sawdust is right. `cfmakeraw()` is BSD-specific (meaning you need to `#define _BSD_SOURCE` before `#include` s). The man termios(3)` man page shows how to open-code it, in the *Raw mode* section. – Nominal Animal Mar 3 '15 at 19:07

I'm not using BSD, so that's not helpful to me. – JonS Mar 5 '15 at 3:23
