Debugging Serial Port Operation

Peter D. Hiscocks
Department of Computer and Electrical Engineering
Ryerson Polytechnic University
350 Victoria Street, Toronto, M5B 2K3, Canada
phiscock@ee.ryerson.ca
August 18, 1999

Introduction

These notes are intended to assist anyone getting the serial ports to talk to external devices, which could be a modem or, in my case, a microprocessor development system. These notes are specific to my particular case, which is a 486-66, 16MB RAM, operating Windows 95 in one hard drive partition and an older version of Redhat Linux in the second hard drive partition. The notes are a result of two day's work in trying to get Linux in contact with our 68HC11 microprocessor board.

I assume here that you're familiar with the basic operation of the serial ports, what a UART does, and how the control signals work. If not, check out the references at the end of this note.

Basically, this note documents the approach of first testing the serial ports under DOS, then under Windows 95, and then under Linux. Because the final system was to run under Linux, most of the detail give here applies to that operating system.

Checking for proper operation of the serial port

The simplest method, requiring a minimum of external hardware, is a loopback connection. This ties the data output from the serial port back to its input. Then, using a terminal emulator program, you can send data out the port and see if it is returned to the emulator screen. If the data is returned correctly, then the serial port is doing its job.

You can send data by typing on the keyboard, or you can upload a long ASCII file to keep the port active for a period of time.

Under DOS, `Procomm` is a suitable terminal emulator. Under W95, use `Hyperterminal`. Under Linux, I used `Seyon` in the X Windows environment. (The Minicom program will work in the Linux command line environment, but I haven't had cause to use it yet.)

A loopback connector is a female DB-9 or DB-25 connector that plugs into the COM port connection, or at the end of a serial cable. The data and control output lines are connected back to the inputs.

The usual configuration on the rear panel of the PC is that a DB-9 male connector is COM1 (ttyS0 under Linux), often used for a serial mouse . The DB-25 male connector is COM2 (ttyS1 under Linux), often used to drive a modem. The DB-25 female connector is the printer port.

For a DB-9 connector loopback connection:

Jumper pins:	Function
2-3	RxD (from PC) to TxD (to PC)
1-6-4	DCD to DSR to DTR
7-8	RTS to CTS

So, for example, to make a DB-9 loopback connector you would obtain a DB-9 female connector, connect pin 2 to pin 3, pin 1 to pins 6 and 4, and pin 7 to pin 8. Then plug it into the serial port on the rear of the PC.

For the DB-25 connector loopback connection:

Jumper pins:	Function
2-3	RxD to TxD
6-8-20	DSR to DCD to DTR
4-5	RTS to CTS

Two gizmos are very handy in diagnosing problems with a serial port. The first is a jumper adapter, that allows you to connect the various pins of a serial connection to each other, and can be used to configure a loopback connection. The second useful gizmo is an in-line signal indicator, in which various LEDs indicate the state of the serial port lines. Each of these can be purchased for about $10 from electronic parts stores.

If you have access to an oscilloscope, this is very useful in monitoring the state of the data lines. When a COM port is connected and quiescent (not sending data) the RxD line should be sitting at some negative voltage: the books say -12 volts, but it's more like -10 on my machine. (It must be at least -5 volts.) When the port is actively sending data, the data lines will switch between -12 and +12 volts.

From here on, we'll assume that there is some problem.

Checking that the interrupts are correctly assigned.

The vanilla PC comes with 2 COM or serial ports, COM1 and COM2. In my case, I had added a second serial port board, which should ostensibly provide ports COM3 and COM4.

The root of all the evilness in this particular episode is the limited number of interrupts available on the stock PC. There basically are only 16 of them, and many different devices need them, so there is a strong possibility for conflict. The normal assignments for the COM ports are COM2 and COM4 to IRQ 3, COM1 and COM3 to IRQ 4. Unfortunately, this won't actually work, since the COM ports can't share interrupts: under W95 and Linux each active COM port must have its own interrupt.

DOS

Under DOS, run the diagnostic MSD.EXE . (This should have come with whatever version of DOS you're using.) This shows a complete table of interrupt usage.

W95

Under W95, to view the interrupt used by the first serial port COM1, open

```
My Computer/Properties/Device Manager/Ports (COM&LPT)/COM1/Resources
```

and you can see the base address and interrupt used. Similarily, you can check the interrupt used by COM2 and the parallel printer port LPT.

In my case, I needed an additional 2 interrupts for COM3 and COM4. The obvious candidate for one of them is IRQ 5, which is assigned by default to the second printer port, LPT2. However, IRQ 5 was being used by the sound board, so I reassigned the sound board to use IRQ 10. The W95 device manager seems to have enough intelligence to be able to decide if this assignment is acceptable to the device or not, because some assignments are locked out. Once I had moved the sound board to IRQ 10, then I could move COM3 over to IRQ 5.

There did not seem to be another available interrupt, so I simply de-installed COM4 using the Windows control panel, since it was not essential.

On the hardware board for COM3 and COM4, there were two jumpers to assign the interrupt lines for the two ports. I left the jumper for COM4 open circuited to disable it, and moved the jumper for COM3 over to IRQ 5.

Linux

Under Linux, you first need to understand the naming conventions of the serial ports. As in any Unix based system, devices are addressed as files in the /dev directory. To make things more confusing, the serial ports have duplicate names. The correspondence between devices and ports is as follows:

DOS	Linux	Linux Alternate
COM1	/dev/ttyS0	cua0
COM2	/dev/ttyS1	cua1
COM3	/dev/ttyS2	cua2
COM4	/dev/ttyS3	cua3

No, I don't know why the serial port has different names, a Guru will have to fill us in on that one. Like much Unix wierdness, probably an artifact of history.

There are zillions of other devices, corresponding to virtual terminals and console devices, but these don't concern us for the direct usage of the serial ports. Other Unix systems use different names for their serial ports.

Checking Interrupts under Linux

To determine the interrupt used by a serial port, use the 'setserial' command, for example, the command:

```
setserial /dev/cua2
```

yeilds the response:

```
/dev/cua2, UART: 16550A, Port: 0x03e8, IRQ: 5
```

To change the interrupt, use the same command, for example:

```
setserial /dev/cua2 irq 9
```

Then you can use the command again to check that the interrupt has actually changed. Notice that these commands have no effect on the serial port hardware: the jumpers must be set to the specified

interrupt lines for the port hardware to work correctly. Setserial simply informs the software driver which interrupt is being used.

In my limited experience, it is advisable to check that the corresponding cua and ttyS devices have the same interrupt specified, because there is no way to predict which name is being addressed by a given software driver.

Enabling and Disabling Flow Control

Flow control uses some of the connections on the serial port to 'throttle' the output so that the receiving device can keep up with the sending device. If the flow control lines are left open-circuited at the connector and flow control is enabled then any software attempting to write to the port will stall and no output signals will appear on the data lines. However, a loopback connector will allow output whether or not flow control is enabled.

Flow control is implemented in software, even if it is designated as 'hardware flow control'. If flow control is disabled, a loopback connector is not required: simply send information to the port and check that line 2 is actuated.

To check on the state of flow control for a port, use the stty command:

```
stty -a < /dev/cua2
```

(Notice the redirection symbol '¡').

This generates several lines of output, among which is the phrase 'crtscts' if flow control is enabled, or '-crtscts' if it is disabled.

You may then disable flow control with the command

```
stty -crtscts < /dev/cua2
```

You should then use 'stty -a ¡ /dev/cua2' to check that flow control is specified as disabled.

Testing a Port

Having checked the interrupt lines and disabled flow control, you are ready to test the port. You'll need a sizeable text file for test purposes. An easy way to generate a lengthy text file for test purposes is to pipe a recursive directory listing into a file, for example:

```
ls -R > dirlisting
```

generates a large file 'dirlisting'. (You can search through this file with a text editor to find files, much easier to use that the unix 'find' command, with its various strangenesses.)

Then, this file may be used to check that a particular serial port is functional, with a command like

```
cat filename > /dev/ttyS2
```

where 'filename' is the text file. See if the data is sent out the serial port, as evidenced by the LED indicators or oscilloscope connection on the data line.

If this does not work, check the following:

Permissions

The permissions must be set correctly for the /dev/cua and /dev/ttyS devices. You can do the testing in superuser mode or change the permissions so that anyone can write to the port in user mode. For example, in superuser mode, 'chmod 777 /dev/ttyS2' makes ttyS2 world readable and writeable.

Device Busy

The device must not be busy: if it complains when you 'cat' a file to it, it may be in use by a terminal emulator program or the serial mouse. For example, when Seyon is using ttyS2, and you attempt to cat a file to cua2, cua2 reports that it is busy. However, 'cat'ing to ttyS2 does work, so we may conclude that Seyon directly addresses cua2.

Flow Control

Ensure that flow control is disabled for the port with the 'stty' command as described above or that a loopback adaptor is plugged into the port.

Using the Seyon Terminal Emulator

The Seyon terminal emulator, like many programs for the X Window system, is very nicely done. However, it takes a bit of detective work to get it up and running the first time. The first requirement is that there be a subdirectory .seyon in your home directory. This should contain the following text files:

```
phonelist  protocols  script  startup
```

Examples of these files are located with the seyon source files, in

```
 usr/src/communications/seyon-2.14c
```

in my machine, and to start with you can simply copy them to your seyon subdirectory.

There are several useful files in the source distribution, including a FAQ and README file.

To startup seyon, you must specify the serial port to use, for example:

```
seyon -modem /dev/ttyS2 &
```

Later, you can modify the .XDefaults file so that the command line can be simplified. Once Seyon is running, there is a 'port' dialogue, where different ports can be selected.

For the Seyon terminal emulator, flow control is referred to as the 'CTS/DTS' switch, and may be turned on or off.

Once you have Seyon up and running, you can use it to check the serial port lines. For example, using the Seyon 'divert' command, you can pipe a large text file to the serial port and see if it appears on the data lines. Or loop it back into the serial port with a loopback connector, and see if it appears correct.

Speed of Transfer

To measure the maximum serial transfer data rate, I transferred a text file of 29912 bytes through /dev/ttyS2 using the Seyon terminal emulator 'divert' command, the maximum baud rate of 38400, hardware flow control (RTS/CTS) enabled and the serial cable terminated by a loopback adaptor.

The data transfer was observed on the oscilloscope to occur in bursts of about 5 milliseconds separated by about 60 milliseconds. The entire file transferred in 149 seconds for an average transfer rate of 200 bytes per second.

The gaps between bursts of data are probably the intervals where the terminal emulator is writing the received characters to the terminal screen, and so one would expect that with a faster processor than this 486/66, the gaps would be reduced.

Next, I used the 'cat' command to transfer data to the port as follows:

First, I changed the 'port' selection in Seyon to ttyS1, to free up ttyS2. Then I executed the command

```
stty -a < /dev/ttyS2
```

to examine the current settings for the port. I found that the 'speed' (baud) rate was 9600, so this was changed to 38400 with the command

```
stty speed 38400 < /dev/ttyS2
```

Also, among the information returned by the stty -a command was the string

```
-crtscts
```

which indicated that flow control was disabled for that port. Flow control was enabled with the command

```
stty crtscts
```

I then found that the 'cat' command produced a short burst of activity on the data lines (2 and 3) and stalled. After some head-scratching, I decided that the receive buffer must be filling, and with nothing to remove it, stalling the serial port. I removed the loopback data connection between pins 2 and 3, which allows data to stream from the port without receiving it. The oscilloscope was connected to pin to 2 to observe the flow of data characters. At this point, the command

```
cat dirlisting > /dev/ttyS2
```

completed correctly, appearing on the scope in a continuous stream, and completing the transfer of 29912 bytes in about 8.6 seconds for an average transfer rate of 3478 bytes per second.

Using the `stty` Command

The parameters of a serial port may be viewed or configured using the Unix `stty` command. To view the current settings of `/dev/ttyS1`, for example:

```
stty -a < /dev/ttyS1
```

spits out a large list of port settings that may be decoded with the help of the `stty man` entry. For example, in the list of parameters is the string `-crtscts`, which indicates that CTS-RTS hardware handshaking is enabled for this port.

In configuring the port, there are zillions of possible settings. Many appear to be historic detritus that was added in to support specific serial devices. For example, the command

```
stty 38400 < ttyS1
```

has the effect of setting the serial port `ttyS1` to receive 8 bit data. To do this from a Tcl program, I used commands like

```
exec stty 38400 < $serial_port
```

where `exec` shells out to a unix command and `$serial_port` is the name of the serial port.

For a complete listing of parameters, check the `man` page for the `stty` command.

References

Sorting this out was greatly aided by information from the following sources, all of which were written by David Lawyer, and are available from the Linux archive site

`http://sunsite.unc.edu/mdw/HOWTO`

- The Linux Serial HOWTO
- The Linux Modem HOWTO
- Text Terminal HOWTO

The following source has much useful information on wiring serial port connectors, handshaking and loopback adaptors:

`http://www.airborn.com.au/rs232.html`