Linux RS232 drivers

User space drivers for TTY devices

- Heather Lomond

Simplest way

By far the simplest way to get an RS232 device hooked up to a linux machine is via a USB to RS232 converter. For some reason these are usually blue cables that have the circuitry built into the connectors. Assuming you have the right drivers installed they usually come up as:

```
/dev/ttyUSB0
```

To be Canonical or not

On most older systems and and, indeed modern ones like the Arduino IDE, input and output to a serial terminal is only performed when a Carriage Return (CR or CR/LF) is sent.

You may have met this when doing a serial.print(xxx) which doesn't actually appear on the screen until either the max number of characters for aline is reached or you send a CR (usually with a serial.println(xxx) command).
When you type into a console on Linux, nothing happens until you press return.

This is known as Canonical form for tty data transfer.

If things happen immediately, eq you have a menu option and you press "1" and it does something straight away, then this is non-canonical form.

Access from a terminal

In order to get simple things going you can access the device file from a normal linux terminal.
To set up a BAUD rate you can use:

```
stty -F /dev/ttyUSB0 speed 9600
```

To let the terminal wait and see what is coming over the RS232 line, you can use

```
cat /dev/ttyUSB0
```

And to send stuff to it you can use

```
echo “stuff” > /dev/ttyUSB
```

RPI SOC Hardware

On the Raspberry PI, the RS232 is available as 3.3V logic on the big connector. (26 way connector show here but same pinouts for the 40 way).

Usually you will want pins 6,8 and 10. Since RS232 does not have a master/slave configuration you will need to connect the RPI TX to the RX on the device you are attaching (and the RX on the PI to the TX on the device).

3.3V	1	2	5V
I2C0 SDA	3	4	DNC
I2C0 SCL	5	6	GROUND
GPIO4	7	8	UART TXD
DNC	9	10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 21	13	14	DNC
GPIO 22	15	16	GPIO 23
DNC	17	18	GPIO 24
SP10 MOSI	19	20	DNC
SP10 MISO	21	22	GPIO 25
SP10 SCLK	23	24	SP10 CE0 N
DNC	25	26	SP10 CE1 N

RPI Console

Most linux systems have a serial console associated with them. For most desktop systems we never use this, going instead to the GUI or screen based console instead, but for embedded things it is often the only way of talking to a linux environment. Most Android phones have this available internally as do most single board computers like the Odroid and RPI.

Since the Pi has the build in SOC RS232 set up as the console which means by default it will send data out on it if you have a serial terminal attached (remember VT100s?). This needs disabled if we are going to use it for our own purposes.

`sudo rpi-config` and select `advanced options/A8`

The built in serial hardware on the Pi is registered as the device

`/dev/ttyAMA0`

This is the file you will need to open to do driver related things

Driver Part 1

I have a sensor that just outputs a 3 digit floating point number (e.g. 16.4), as ascii text every 0.2 seconds followed by a CR/LF pair. I want to read the number into my c program so that I can display it.

First some headers and defines

```
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>

/* Most of the control parameters are set in <termios.h> */

#define BAUDRATE B9600
#define MODEMDEVICE "/dev/ttyUSB0"
```

Next comes the setup code …

Driver Part 2

```
main() {
        int fd, res;
        struct termios UEGO_tio;   /* this is the control structure  for the tty */
        char buf[255];

        /* Open with read/write and ignore Ctrl-C */
        fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );

        /* clear our control structure for new port settings */
        bzero(&UEGO_tio, sizeof(UEGO_tio));

        /* Set the baudrate, hardware flow control, 8 bits, no parity, 1 stop bit,
           direct control of the RS232 (i.e Local control) and enable receiving) */
        UEGO_tio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

        /* this tells it to ignore errors and exchange CR for NL */
        UEGO_tio.c_iflag = IGNPAR | ICRNL;

        /* This says to use raw output */
        UEGO_tio.c_oflag = 0;

        /* This tells it to be a canonical input */
        UEGO_tio.c_lflag = ICANON;

        /* now clean the modem line and activate the settings for the port */
        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&UEGO_tio);
```

Driver Part 3

That is all the setup required. All we need to do now is go read lines from the file and output them

```
        while (1) { /* loop until we are terminated */
                  res = read(fd,buf,255);
                  buf[res]=0; /* set end of string, so we can use printf */
                  printf("UEGO = %s\n", buf);
        }

}
```

Nan-Canonical Drivers

If we want to do Non-canonical work, then we just use the c_lflags to tell the driver to use this mode and set the number of characters to block the read on:

```
newtio.c_lflag = 0;

/* This tells the read to block until just 1 char is available
   if we set VTIME to >0 this will act as a timeout and the read will return if
   the one character is available or the read blocks untill the timeout occurs
   in which case no characters are returned */
newtio.c_cc[VTIME] = 0;
newtio.c_cc[VMIN] = 1;
```

And then we can have a loop something like:

```
while (1){
    res = read(fd,buf,255);    /* blocks untill 1 char is received */
    buf[res]=0;
    printf("%s\n", buf);
}
```

Questions