ECE 598 – Advanced Operating Systems
Lecture 5

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 January 2015

Announcements

- Homework assigned and was due (missed lecture due to storm)
- Trying out new classroom, Barrows 221 (TI Lab)

Cross Compilers

- Some are having difficulty getting this working.
- On ARM Linux, no need at all. Problem is copying to a memory key.
- On x86 Linux can install. Either grab the .tar.gz and setup, or if on Ubuntu might be as simple as `apt-get install gcc-arm-none-eabi`. On Debian you will need to add deb `http://www.emdebian.org/debian unstable main` to your `/etc/apt/sources.list` first.

On 64-bit machines you may also need to install a 32-bit compatibility library

- On OSX people seem to mostly successfully getting Yagato working following the instructions given.
- Windows is a problem. Directions inside a .bz2 file and involves installing MINGWIN first plus then compiling a bunch of files.
- Updating your PATH is an issue. Can get around this by just hardcoding the path in your Makefile CROSS variable as described in HW handout.

Serial Ports

- Want easy I/O.
- Blinky LEDs not enough. Could have O/S communicate by Morse code on the LED (were patches for Linux to do this at one point)

Serial Ports

- Minimum TX, RX, Ground. Older systems 9pin/25pin
- Used for many things (modems, terminals, mouse, GPS, etc)
- RS-232
- Interface usually set of IO-Address and Interrupt (sometimes DMA too)
- ttyS0, ttyS1 and similar (Linux) COM1, COM2 Windows

Serial Port FIFO

- Internal First-in-First-Out structures
- 1 byte (older) to 16 bytes
- Can generate Interrupt. Have to handle in time or else data lost
- Why timeout? Send 1-byte typing, stall if not 15 more.
- Why FIFOs small? flow control. A byte saying to stall sent, if large FIFO a long time before it actually gets

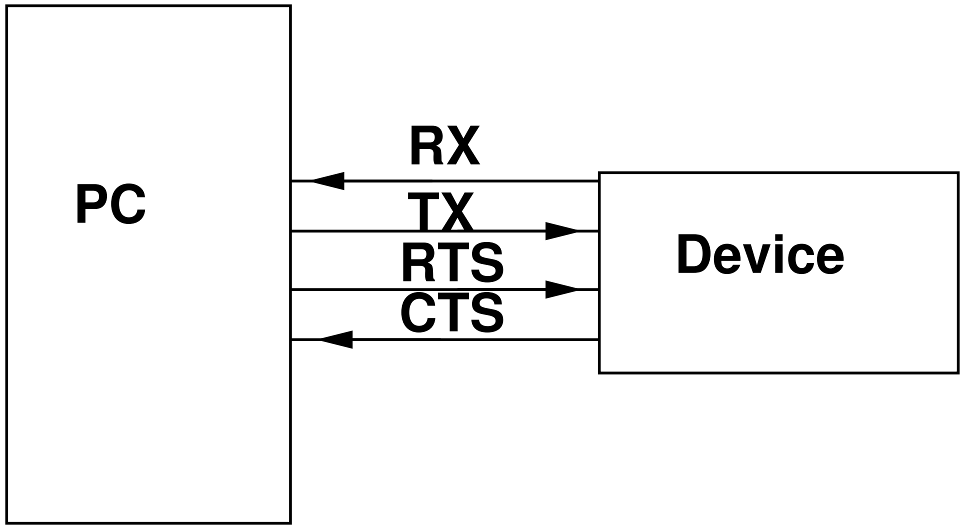
received

- Interrupts: when FIFO reaches a certain size, or if there's a delay. (so if someone typing slowly at the keyboard) also when transmit FIFO empty

Flow Control

- Ability to start/stop without losing bytes
- Often multiple levels of buffering. FIFO, buffer on device, buffer in OS.
- Hardware: RTS (Request to Send) and CTS (Clear to Send) Positive or negative, one to the other. Why in old days needed a crossover cable to connect two PCs together.
- Software: uses the ASCII DC1 (XON) and DC3 (XOFF)

(17 and 19) device control chars over the line. Why bad? Cannot send binary data easily. Why?



Signals

- TX,RX,GND (minimum)
- RTS,CTS (HW flow control)
- DSR,DTR (Data set ready, data terminal ready, other flow control)
- DCD (data carrier detect – modem there)
- RI (ring indicator)

- +12V for 0, -12V for 1 (inverted on transmit/receive, regular on other pins)
  (-15 to -3 or so, 15 to 3 or so).
  Some are -5/5V tolerant.
  Modern TTL 5/0 exist but not back compatible
  3.3V I/O like on Pi troublesome
  (why awkward on small embedded boards)

- DCE (data communication equipment) to DTE (data terminal equipment) straight through. Need loopback if DTE to DTE (swap RTS and CTS)

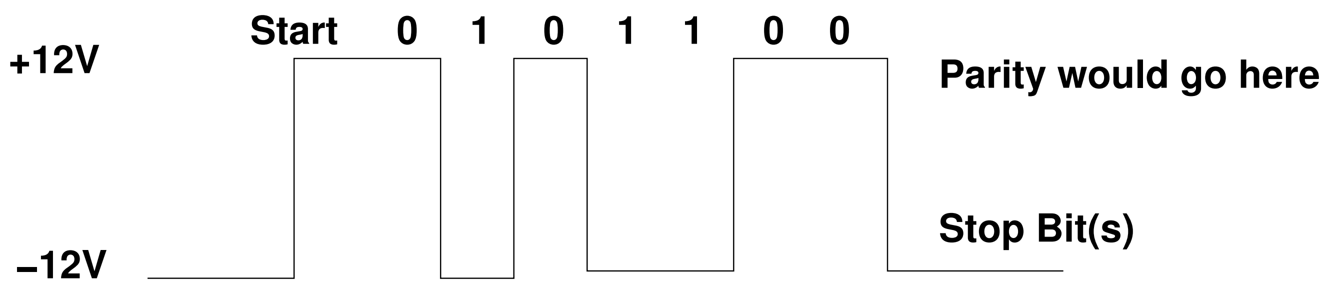
Transmitting a Byte

- TX Held -12V when idle (1)
- Jumps to 12V for start bit (0)
- Bits transmitted. Low bit first. Stays at 12V if 0, -12 if 1
- If parity bit desired, sends that (even, odd, stick, or none)
  If odd, then parity bit is included that makes the number

of 1s (including parity) odd.
If even, then parity bit included that makes number of 1s even.
Stick parity (mark = always 1, space = always 0)

- Then down to -12V for stop bit(s)

- Since no clock, no way to tell difference between consecutive bits of same value. Starts a counter counting at the start bit, samples values from there.


Start
0
1
0
1
1
0
0
+12V
−12V
Parity would go here
Stop Bit(s)

UART

- Universal Asynchronous Receiver Transmitter
- Convert parallel value to serial
- Asynchronous. Why? No clock signal wire.

History

- back in the day spent lot of your life configuring serial connections
- The Linux ioctl interface to this is a pain, old legacy
- Hooking old machines together (Apple II, etc)

Speed (flow rate)

- Clock crystal, programmable divider
- Default old max speed was 115,200; many modern can go much faster, even up to 4MBps. Cables often not up to it as standard did not specify twisted pair
- Common speeds 1 (115.2k), 2 (57.6k), 3 (38.4k), 6 (19.2k), 12 (9.6k), 24 (4.8k), 48 (2.4k), 96 (1.2k) 300, 110 (war games acoustic coupler)

Programs

- `learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/test-and-configure`
- putty is a decent one for Windows
- I use minicom for Linux. A bit of a pain. Have to install it (not by default?) Control-A Control-Z for help. Has similar keybindings to an old DOS program Telix that I used for years.
- OSX? Also Linux?

```
sudo screen /dev/ttyUSB0 115200
```

File Transfer

- Kermit
- Zmodem/Xmodem/Ymodem/etc
- sz / rz on Linux

Things you will need to set

- 115200 8N1 Software Flow
- 115200 Baud
- 8 data bits (7 or 8)
- no parity (even, odd, none)
- 1 stop bit (1, 1.5, or 2)

USB Serial Converters

- PL2303 / FTDI most common
- Often counterfeited, in the news for that recently (and how the companies tried to kill the counterfeits)
- Takes serial in, presents as a serial port to the OS. ttyUSB0 on Linux, COM something really high on windows