ECE 531 – Advanced Operating Systems
Lecture 7

Vince Weaver

`https://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

17 September 2025

Announcements

- HW#2 was posted
- How is HW#2 going? Don't wait until the last minute!

HW#1 Review

- Put your name on the assignment!
- Short answers OK, but please explain things at least a little (more than a single word).
- Answers
  - Benefits of OS: HW abstraction. Multi-tasking?
  - Downsides of OS: Overhead, Timing, Nontransparency, Single point of failure
  - Why C?: Low-level, historical, faster
  - Why not C?: Security risk. Lack of features?, not

object oriented?

◦ BeOS (Haiku), Plan9, QNX, Solaris, VxWorks, FreeBSD

Note on the Documentation

- Pis have "relatively" good documentation
- It can still be vague, for example GPSET is used to set GPIO bits. It's listed as R/W though. What happens when you read?
- Same, the GPLEV used to read bits. But it's R/W. What happens if write?
- Also the at boot / on reset state might not be true as the firmware runs before we get a chance to access things

Connecting to a Computer

- 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)
  Or a PDP-11 or similar
- USB keyboard / HDMI display quite complex, thousands of lines of code
- Network/Bluetooth/wifi not any better
- What's the simple way to get data in/out?
- Serial ports

Serial Port Background

- “Serial” meaning one-bit at a time
- Extremely easy to make and program
- Back in the day spent lot of your life configuring serial connections
- Still used a lot on embedded systems

Historical Serial Port Uses

- Hooking old machines together (Apple II to printer)
- Modems to connect to BBS or internet
- Programming network switches
- Computer mice
- Multiplayer-gaming
- Most uses now use USB instead (was to save on types of cables, we see how that ended up)

Using Modems to Connect Computers over Phone lines

- By the late 90s this was the primary use of serial ports
- Eventually replaced by “win-modems” that faked everything (often poorly)
- Side story about controlling modems with serial
  - Could send commands to modem inline while connection open (sort of like SW flow control)
  - HAYES modem command set. ATDT, etc.
  - +++ *pause* ATH0

- Hayes patented the pause. What did everyone have to do?

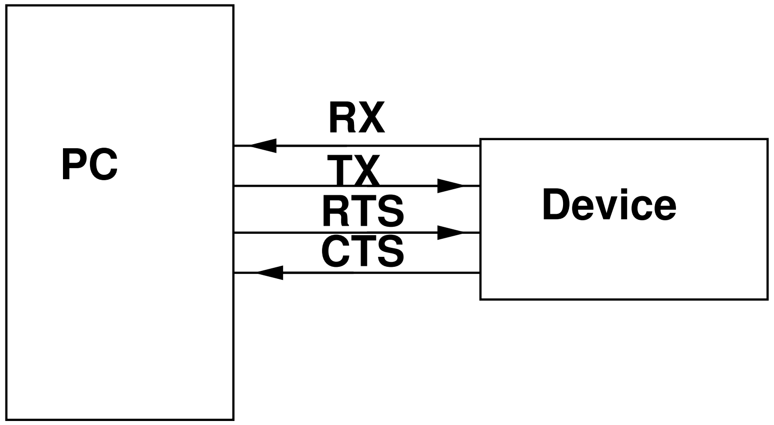
Serial Port Basics

- RS-232 was traditional interface, but modern systems rarely implement it exactly
- Minimum three wires TX, RX, Ground
- Older systems the serial ports were 9pin or 25pin (mostly same signals, just lots more grounds on 25pin)
- /dev/ttyS0, /dev/ttyS1 and similar (Linux)
- COM1, COM2, etc. Windows

Serial Signals

- TX,RX,GND (transmit, receive, ground)
- RTS,CTS (HW flow control, request to send, clear to send)
- DSR,DTR (additional flow control: Data set ready, data terminal ready)
- DCD (data carrier detect – modem there)
- RI (ring indicator)

Historical Serial Cables



- If connecting DCE (data communication equipment) to DTE (data terminal equipment) use “straight through” cable
- If DTE to DTE (connecting two computers) need special loopback/null-modem cable that swaps RTS and CTS

Serial on-the-wire

- RS-232: +12V for 0, -12V for 1 (inverted on transmit/receive, regular on other pins)
- Often -15V to -3V or 15V to 3V
  Might work at -5V/5V
- Modern systems often don't have negative voltages
  Can instead use "TTL" 5V/0V which needs adapter to talk to "real" serial equipment
- Raspberry pi uses 3.3V/0V so be careful

Flow Control

- Ability to start/stop without losing bytes
- Often multiple levels of buffering. FIFO, buffer on device, buffer in OS.

Hardware Flow Control

- Needs extra signal wires
- RTS (Request to Send) and CTS (Clear to Send) Positive or negative, one to the other

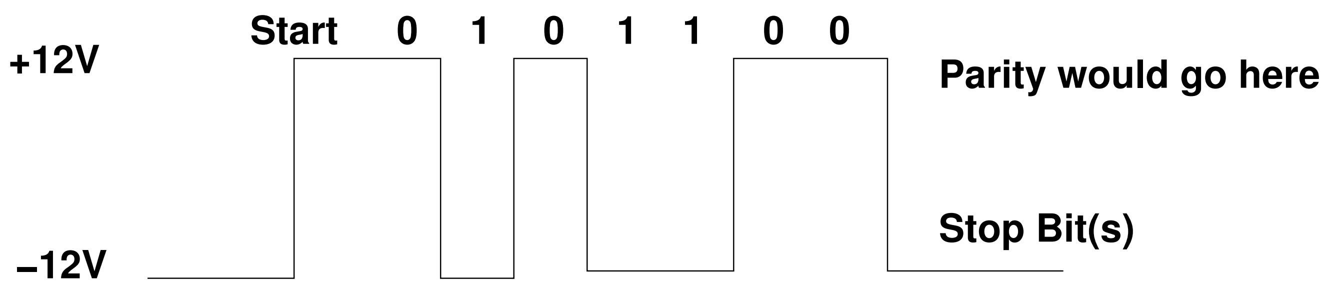
Software Flow Control

- ASCII DC3 (XOFF) (0x13) (control-s) stops transmission
- ASCII DC1 (XON) (0x11) (control-q) restarts transmission
- Requires no extra wires
- What if sending binary data that contains those characters? Must escape those.
- Note: there isn't a standard way of escaping SW flow control, it's up to the application

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
- Parity (simple error detection)
  - (even, odd, stick, or none)
  - Odd, then parity bit is included that makes the number of 1s (including parity) odd.
  - Even, then parity bit included that makes number of

1s even
  - Stick parity (mark = always 1, space = always 0)
  - None, save a bit and just don't include
- Then down to -12V for stop bit(s)
- Since no clock, no way to tell difference between consecutive bits of same value.
  Both ends need to agree to speed before setting up connection.
  Starts a counter based on agreed speed and starts counting at the start bit, samples values from there.



Value sent is 001 1010

Speed (flow rate)

- Most common speed in 9600 8N1
  - 9600 = bits per second (sometimes called baud even though that's more complicated)
  - 8 = bits to transmit
  - N = no parity
  - 1 = 1 stop bit
- 115200 is also popular, is traditionally the top speed supported by serial
- 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)

Serial Hardware: UART

- Universal Asynchronous Receiver Transmitter
- Convert parallel value to serial
- Asynchronous. Why? No clock signal wire.
- Note: some embedded systems have USART (synchronous) that do have a clock signal

UART 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

Terminal Programs

- To communicate over a serial port you need a program that talks to the port and send/receives bytes
- `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.
- "screen" on MacOS and also Linux

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