Character by character TTY input in Unix, then and now

November 10, 2018

In Unix, normally doing a `read()` from a terminal returns full lines, with the kernel taking care of things like people erasing characters and words (and typing control-D); if you run '`cat`' by itself, for example, you get this line at a time input mode. However Unix has an additional input mode, *raw mode*, where you `read()` every character as it's typed (or at least as it becomes available to the kernel). Programs that support readline-style line editing operate in this mode, such as shells like Bash and zsh, as do editors like vi (and emacs if it's in non-windowed mode).

(Not all things that you might think operate in raw mode actually do; for example, `passwd` and `sudo` don't use raw mode when you enter your password, they just turn off echoing characters back to you.)

Unix has pretty much always had these two terminal input modes (kernel support for both goes back to at least Research Unix V4, which seems to be the oldest one that we have good kernel source for through tuhs.org). However, over time the impact on the system of using raw mode has changed significantly, and not just because CPUs have gotten faster. In practice, modern *cooked* (line at a time) terminal input is much closer to raw mode than it was in the days of V7, because **over time we've moved from an environment where input came from real terminals over serial lines to one where input takes much more complicated and expensive paths into Unix**.

In the early days of Unix, what you had was real, physical terminals (sometimes hardcopy ones, such as in famous photos of Bell Labs people working on Unix in machine rooms, and sometimes 'glass ttys' with CRT displays). These terminals were connected to Unix machines by serial lines. In cooked, line at a time mode, what happened when you hit a character on the terminal was that the character was sent over the serial line, the serial port hardware on the Unix machine read the character and raised an interrupt, and the low level Unix interrupt handler read the character from the hardware, perhaps echoed it back out, and immediately handled a few special characters like ^C and CR (which made it wake up the rest of the kernel) and perhaps the basic line editing characters. When you finally typed CR, the interrupt handler would wake up the kernel side of your process, which was waiting in the tty `read()` handler. This higher level would eventually get scheduled, process the input buffer to assemble the actual line, copy it to your user-space memory, and return from the `read()` to user space, at which point your program would actually wake up to handle the new line it got.

(Versions of Research Unix through V7 actually didn't really handle your erase or line-kill characters at interrupt level. Instead they push everything into a 'raw buffer', and only once a CR was typed was this buffer canonicalized by applying the effects of characters to determine the final line that was returned to user level.)

The important thing here is that **in line at a time tty input in V7, the only code that had to run for each character was the low level kernel interrupt handler,**

These are my WanderingThoughts (About the blog)

Full index of entries
Recent comments

This is part of CSpace, and is written by ChrisSiebenmann. Twitter: @thatcks

* * *

Categories: links, linux, programming, python, snark, solaris, spam, sysadmin, tech, unix, web

This is a DWiki. GettingAround (Help)

Search:

and it deliberately did very little work. However, if you turned on raw mode all of this changed and suddenly you had to run a lot more code. In raw mode, the interrupt handler had to wake the higher level kernel at each character, and the higher level kernel had to return to user level, and your user level code had to run. On the comparatively small and slow machines that early Unixes ran on, going all the way to user-level code for every character would have been and probably was a visible performance hit, especially if a bunch of people were doing it at the same time.

Things started changing in BSD Unix with the introduction of *pseudo-ttys* (ptys). BSD Unix needed ptys in order to support network logins over Telnet and rlogin, but network logins and ptys fundamentally change what the character input path looks like in practice. Programs reading from ptys still ran basically the same sort of code in the kernel as before, with a distinction between low level character processing and high level line processing, but now getting characters to the pty wasn't just a matter of a hardware interrupt. For a telnet or rlogin login, the path looked something like this:

- the network card gets a packet and raises an interrupt.
- the kernel interrupt handler reads the packet and passes it to the kernel's TCP state machine, which may not run entirely at interrupt level and is in any case a bunch of code.
- the TCP state machine eventually hands the packet data to the user-level telnet or rlogin daemon, which must wake up to handle it.
- the woken-up telnetd or rlogind injects the new character into the master side of the pty with a `write()` system call, which percolates down through various levels of kernel code.

In other words, **with logins over the network, a bunch of code, including user-level code, had to run for every character even for line at a time input**.

In this new world, having the shell or program that's reading input from the pty operate in line at a time mode remained somewhat more efficient than raw mode but it wasn't anywhere near the difference in the amount of code that it was (and is) for terminals connected over serial lines. You weren't moving from no user level wakeups to one; you were moving from one to two, and the additional wakeup was on a relatively simple code path (compared to TCP packet and state handling).

(It's a good thing Vaxes were more powerful than PDP-11s; they needed to be.)

Things in Unix have only gotten worse for the character input path since then. Modern input over the network is through SSH, which requires user-level decryption and de-multiplexing before you end up with characters that can be written to the master side of the pseudo-tty; the network input may also involve kernel level firewall checks or even another level of decryption from a VPN (either at kernel level or at user level, depending on the VPN technology). Windowing systems such as X or Wayland add at least two processes to the stack, as generally the window server has to read and process the keystroke and then pass it to the terminal window process (as a generalized event). Sometimes there are more processes, and keyboard event handling is generally complicated in general (which means that there's a lot of code that has to run).

I won't say that character at a time input has no extra overhead in Unix today, because that's not quite true. What is true is that the extra overhead it adds is now

only a small percentage of the total cost (in time and CPU instructions) of getting a typed character from the keyboard to the program. And since readline-style line editing and other features that require character at a time input add real value, they've become more and more common as the relative expensive of providing them has fallen, to the point where it's now a bit unusual to find a program that doesn't have readline editing.

The mirror image of this modern state is that back in the old days, avoiding raw mode as much as possible mattered a lot (to the point where it seems that almost nothing in V7 actually uses its raw mode). This persisted even into the days of 4.x BSD on Vaxes, if you wanted to support a lot of people connected to them (especially over serial terminals, which people used for a surprisingly long time). This very likely had a real influence on what sort of programs people developed for early Unix, especially Research Unix on PDP-11s.

PS: In V7, the only uses of RAW mode I could spot were in some UUCP and modem related programs, like the V7 version of cu.

PPS: Even when directly connected serial terminals started going out of style for Unix systems, with sysadmins and other local users switching to workstations, people often still cared about dial-in serial connections over modems. And generally people liked to put all of the dial-in users on one machine, rather than try to somehow distribute them over a bunch.

Written on 10 November 2018.

« Getting CPU utilization breakdowns efficiently in Prometheus

Why Prometheus turns out not be our ideal alerting system »