Open Computing ‘‘Hands-On’’: ‘‘PC-Unix Connection’’ Column: June 94

Getting Getty Gone

The author finally parts with his beloved `uugetty`, and the mysterious SVR4 `ttymon` moves in. Does the new relationship stand a chance?

By Tom Yager

I’m not afraid of change. In my lab, I can’t afford to get too attached to anything because, chances are, the next project will require that I yank and reconfigure some part of the shop. But I have set aside one system--my Mobius Unixware server--that I am determined to hold safe from the unsteadying winds of constant reconfiguration. Anything I do to that system is purely voluntary and carefully planned.

I recently had occasion to install Novell’s first major Unixware update, version 1.1. This version incorporates all the vendor’s bug fixes to date, plus new features like SLIP and PPP. I set aside a full day to back up my server and install the update. Things went quite smoothly, with the exception of one buggy post-install script that was simple to repair. I had some time left over, so I planned the demise of my time-honored friend, `uugetty`.

Its Honorable Past

If you didn’t know, `uugetty` is a part of recent Unix UUCP implementations. It replaces `getty`, the system’s doorman for serial-line connections. The `getty` utility sets port parameters (speed, parity, and so on) and waits for a connection, either on a modem or a direct serial line to a terminal or another system. Once it sees evidence of a connection--carrier Detect (CD) turning on or a connected-user pressing the Return key--`getty` prompts the user for a log-in name, then turns control over to the `login` program.

The reason `uugetty` exists is that older `getty`s always assumed they had exclusive control of the serial line. On a system that runs UUCP and makes outgoing calls to other systems, this assumption doesn’t wash. UUCP would try to make an outgoing call and fail because `getty` was parked on the line. The activity of `uugetty` allowed UUCP to take over the line and hold it until its transfers were complete. Then `uugetty` would return to passively listening for incoming connections.

That Unixware includes `uugetty` at all speaks to the general unwillingness of us Unix types to let go of something once we’ve learned it. In my case, I’ve endured broken `getty` and `uugetty` implementations on every PC Unix system I’ve ever configured. It was only when I got to Unixware that I was pleasantly surprised by a version of `uugetty` that truly supported bidirectional modem lines.

I left `uugetty` running (through entries in `/etc/inittab`) even with the knowledge that it was obsolete. Unixware, like other System V Release 4 versions, replaced the whole kludgy serial-port access scheme with a clean, lean access monitor called `ttymon`. I put off using it because its configuration is atypical of traditional Unix and more typical of the new breed: instead of filling a text file with meticulously ordered parameters, the best way to configure ttymon is through Unixware's Sysadm administrative shell interface. It looked involved and not sufficiently documented, so I let it slide.

Its Obvious Decline

Even on systems that implement it well, like Unixware, uugetty is far from perfect. Most notably, it cannot set the full range of available serial-line attributes. You tell uugetty how to set serial parameters by defining them in /etc/gettydefs. Each line in this file defines the speed, parity, XON/XOFF support, and a limited number of other settings. It's that ''limited number'' part that drove me to dump uugetty. In setting up my system for dial-up access, I knew I needed to support a wide range of connection speeds, anywhere from 1,200 baud all the way up past 20-kilobyte baud. The speed doesn't matter, but uugetty wouldn't allow me to set the line for hardware (CTS/RTS) flow control. So a user connected with a slower modem would get overruns unless I made sure every log-in script contained an stty command enabling hardware flow control.

Another flaw in uugetty is that it does not permit passing control to anything but the login program. It always prompts for a log-in name, then hands that name off to login for a password prompt. If I wanted to set a line up with a special program other than login, say, an invention of my own I named bbslogin, I'd have to follow a convoluted path.

The new ttymon solves both of these thorny problems. Its serial-port parameter file, /etc/ttydefs, lets you use any parameter keyword compatible with stty. So you can set not only speed and parity, but also hardware flow control, interrupt and erase characters, and other attributes not allowed in /etc/gettydefs.

The ttymon command is also considerably more configurable than uugetty. A single invocation of ttymon can watch any number of ports; one instance of uugetty was required for each serial line being handled. When a connection is spotted, ttymon can fire up login or any program you specify. If you select something other than login, as with a terminal in the stockroom, ttymon lets you specify not only the program name, but also the user ID under which it will run. The user will never be prompted for a log-in name. The moment the terminal is turned on, the specified program will launch. When it is turned off, the user will be logged out.

Finally, ttymon lets you be more expressive, informative, or obnoxious as your character requires. You can enter an arbitrary string that ttymon will use as a log-in prompt. You can also define a string that will be displayed in lieu of a connection when you have disabled a port. Instead of simply not answering the modem call or sitting unresponsive as a terminal user bangs on the Return key, ttymon will send a message informing them that the port is unavailable.

Its Quiet Exit

I decided to snuff out uugetty's light when my homework revealed the above advantages available with ttymon. The first step is the killing itself, changing the entries in /etc/inittab from ''respawn'' to ''off'' for each uugetty. Then copy /etc/inittab to the file /etc/conf/init.d/kernel so uugetty won't mysteriously restart on you the next time you relink your Unix kernel. Enter **init q** to force init to reread /etc/inittab, and shut down your serial-port log-in attempts.

Then, from a good full-screen terminal--preferably the console, but you can use an xterm X Window

System terminal as well, which allows you to point and click with your mouse--type **`sysadm ports`**. It's from here that you set up the Service Access Facility (SAF), of which `ttymon` is a part.

First, tell SAF that you want `ttymon` listed among its supported services. From the top-level menu, navigate the following path through the menus: ''port_monitors,'' then ''add.'' You'll see the dialog box shown in Figure 1 (40K GIF). Filling this box in is easy: just select an arbitrary port monitor tag (I chose the tag ''modems'') and fill in the rest as shown.

As soon as you complete the form and press the ''save'' button (or F3), `ttymon` is registered as a port-monitor service and launched. Now you need to configure `ttymon` for each of the ports you plan to monitor for connections. Get back to the top-level menu (hit F6 or ''cancel'' several times) and follow the flow: ''port_services,'' ''add,'' ''add to one,'' and then pick the port monitor tag you defined in the previous step. You'll then get two dialog boxes in sequence that prompt you for the settings associated with a single port. Figure 2 (40K GIF) has an example of the settings you should use. You'll need to modify the port monitor tag, service tag, tty device name, and prompts to match your system. The ''ttylabel'' field refers to an entry in `/etc/ttydefs` that needs to exist before you can complete this step. Pull up the man page on `stty` to get a list of the attributes you can define in `/etc/ttydefs`. Use the `sttydefs -l` command to check the validity of the `/etc/ttydefs` file if you encounter problems.

This simple tutorial will get your ports running, but it won't help you understand how or give you much to go on if things don't work. The *System Setup and Configuration* manual, included with your operating system, contains excellent descriptions of how `ttymon` works and how the SAF in general is structured. If you use modems with your SVR4 system, I recommend you make the switch even if you have to put off a thorough understanding of `ttymon` and the SAF.

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

HTML markup by Tim "The Kid" Cooley
Last Modified: Tuesday, 22-Aug-95 15:48:32 PDT