NAME

powstatd – a configurable UPS monitor.

SYNOPSIS

powstatd [**–t** | **–k**]

DESCRIPTION

powstatd is a configurable UPS monitoring daemon designed to work with SysVinit (e.g., most Linux distributions).

powstatd monitors a serial connection from a "dumb" or "relay" UPS for power failures and shuts the machine down gracefully if the power remains off for more than a prespecified interval. *powstatd* can also be configured to allow a master machine to control several slave machines connected to the same UPS via a network connection. This allows you to run several machines off the same UPS, with only one of the machines actually reading the UPS status over the serial line. When compiled with appropriate options enabled, *powstatd* also provides security by means of fast encryption of master/slave communication to prevent malicious shutdown of slave systems.

powstatd has two options:

–t Test mode. Used to explore what a given UPS signals under various failure modes as an aid to constructing an appropriate configuration file. When operating in test mode, *powstatd* doesn't actually signal init, so while changes in UPS status are detected, they are not acted upon. See configuration information for more details.

–k Kill mode. Directs *powstatd* to attempt to shut down the UPS. Most UPS systems will ignore the shutdown signal unless the power is actually off. By shutting down the UPS, *powstatd* ensures that once the main power returns the UPS will automatically turn on and cause the machine to reboot.

CAVEATS

powstatd uses the connectionless UDP protocol to communicate UPS status from master to slave. To keep just anyone from generating a UDP packet that will shut your slave machine down when encrypted master/slave communication is not in use, the slave checks the IP address of origin in the UDP packet it receives. If your master has more than one network card, a master with multiple addresses may provide an unexpected IP address, which results in the slave ignoring (legitimate) status reports.

So when you have a machine with multiple network cards, you must either ensure that the master has a unique IP address (try connecting the single-NIC slave to the UPS and demoting the dual-NIC master to slave instead), or use encrypted master/slave communication, which does not need to check IP source addresses thanks to the cryptographic measures used.

When using encrypted master/slave communication, *powstatd* also uses a timestamp to foil replay attacks. Outdated status messages (as defined by a compile-time constant) are simply rejected; thus master and slave clocks should be reasonably synchronized (using, for example, the NTP protocol) or valid status messages may be rejected.

CONFIGURATION

Important Note: This section has been modified from the upstream version in order to describe how the software is configured and installed on **Debian** systems.

powstatd is configured via a configuration file, */etc/powstatd.conf* by default, that specifies the serial line behavior of the UPS under its various failure modes (note that, if compiled with appropriate security option enabled, *powstatd* will require that file permissions on the configuration file deny rwx access to "group" or "other"). Note that the **Debian package powstatd** is compiled with secure cryptography using the public-domain ANSI C implementation of TEA, the *Tiny Encryption Algorithm,* from *ftp://vader.eeng.brad.ac.uk/pub/crypto/xtea.c*

To configure your UPS, follow these steps:

1. Make sure your new UPS is completely charged and is connected to your machine via the serial monitoring line provided by the UPS manufacturer to your machine (if you did not get such a monitoring line with your UPS, see the the UPS-HOWTO for information on how to make one).

2. In order to configure *powstatd,* make sure your machine **is not** powered from the UPS but is instead plugged directly into the wall outlet (your UPS should still, however, be connected to the machine via the serial monitoring line). Instead, plug a desk light or radio into the UPS.

3. Determine what serial line is connected to your UPS and edit it into the configuration file */etc/powstatd.conf* (you can look at the sample powstatd.conf.* files included in the Debian distribution in */usr/share/doc/powstatd/* or */usr/share/doc/powstatd-crypt/*) at the line specifying the serial line, e.g.:

```
watch ttyS0
```

4. As root, run *powstatd* in test mode (you'll have to be root to have rw access to /dev/ttyS0 or whatever device corresponds to your UPS monitoring line).

```
% ./powstatd -t
```

Hit ^C to stop after you see something like:

CTS	DSR	DCD	RNG	RxD	TxD	DTR	RTS	STATUS
1	0	1	1	0	0	0	1	OK
1	0	1	1	0	0	0	1	OK
1	0	1	1	0	0	0	1	OK
1	0	1	1	0	0	0	1	OK

The output describes the state of the serial connection between the UPS and the computer. Since the signaling needs of a UPS/computer connection are really quite low (just a few bits), we aren't really using the serial connection as it was designed to be used for higher-throughput applications. Instead, we'll use the individual control and transmission lines in the nine wire standard connector a bit differently.

Of the nine wires, six are of particular interest: four "input" control lines (CTS, DSR, DCD, RNG) and two "output" control lines (DTR, RTS). The remaining three lines consist of an electrical ground and the two transmission lines (TxD and RxD, one running in each direction); the latter are typically used for asynchronous serial information rather than for control. Most "dumb" UPS systems ignore the transmission lines and operate only by toggling combinations of the four "input" wires to indicate the status of the UPS and the two "output" wires to send commands back to the UPS from the computer (for now, we'll assume that the current UPS is one of these).

Since the UPS is fully charged and the power is on, we want to play with the configuration in */etc/powstatd.conf* until STATUS reads OK, and not LOW or FAIL. Provided we are actually watching the correct serial line, getting STATUS to read OK entails setting initial values for the output lines DTR and RTS (of course, it makes no sense to try to set initial values on the input lines). Try adding permutations of init values like:

```
init dtr 1
init rts 1
```

until you get the OK status reading.

5. Repeat step 4, but this time pull the plug on the UPS and then reinsert it after a few seconds. Observe changes in the values for the input lines; it should be easy to determine what line corresponds to power failure. For example:

CTS	DSR	DCD	RNG	DTR	RTS	STATUS	
1	0	1	1	0	1	OK	
1	0	1	1	0	1	OK	
1	0	1	1	0	1	OK	
1	0	1	1	0	1	OK	
0	0	1	1	0	1	FAIL	<- plug pulled
0	0	1	1	0	1	FAIL	
0	0	1	1	0	1	FAIL	
0	0	1	1	0	1	FAIL	
1	0	1	1	0	1	OK	<- plug reinserted
1	0	1	1	0	1	OK	
1	0	1	1	0	1	OK	

would indicate that CTS going to 0 corresponds to a power failure. If you leave the UPS unplugged long enough to discharge (this may take quite a while even with a good size light bulb!), the UPS should signal a battery failure in a similar fashion, e.g.,

CTS	DSR	DCD	RNG	DTR	RTS	STATUS
0	0	0	1	0	1	LOW
0	0	0	1	0	1	LOW
0	0	0	1	0	1	LOW
0	0	0	1	0	1	LOW

6. At this point, you should know the power failure and low battery signals. Now we must determine the appropriate UPS shutdown signal. Fortunately, for most UPS systems, there are only 4 possible simple signals. With the UPS on battery power (e.g., unplugged), try adding each of the following lines into */etc/powstatd.conf* in turn:

```
kill dtr 0
kill dtr 1
kill rts 0
kill rts 1
```

for each trial, try issuing the command:

```
% ./powstatd -k
```

one of the line specifications should cause the desk lamp or radio to turn off; that's the one you want. Important: be aware that many UPS have a "dead time" after the signal is sent before the UPS turns itself off; this "dead time" can be as long as 30-45 seconds! So don't be too impatient here or you won't know which signal is responsible for actually turning the UPS off.

7. At this point configuration should be complete. For example, for the Cyberpower Power99 325/385/450/500VA models, a reasonable */etc/powstatd.conf* configuration file reads:

```
watch ttyS0
fail cts 0
low dcd 0
init rts 1
init dtr 0
```

kill dtr 1

8. If you have other machines running off the same UPS, include one or more slave entries specifying their names in the master's configuration file:

slave slave1.domain.edu
slave slave2.domain.edu

The slave machines' configuration files */etc/powstatd.conf* should contain a single line specifying the name of the master machine that is actually monitoring the UPS that also powers the slave:

watch master.domain.edu

If you intend to run more than 20 slaves off a very large UPS, you will need to adjust the MAXSLAVES parameter in the source code accordingly and then recompile.

If *powstatd* is compiled with the appropriate security option enabled, encryption is used to protect slaves from malicious shutdown messages (This is true on Debian systems). An identical password directive should therefore appear in both master and slave configuration files:

password MyPasswordHere

In addition to encrypting status information, *powstatd* will encrypt generate and check timestamps in order to foil replay attacks.

9. Make sure your new /etc/powstatd.conf doesn't have a line like:
#!Unconfigured! Please customize the file as appropriate and remove this line
(all in one line) because the daemon starter script */etc/init.d/powstatd* inspects */etc/powstatd.conf* for this line to see if the daemon is properly configured and should be started or not.

10. You may also want to edit /etc/init.d/powerfail at the line:
failtime=+5 # shutdown delay from initial power failure

It sets the shutdown delay to 5 minutes after power line failure detection. You may want to increase that if your UPS has lots of power to spare, or reduce it.

11. Reboot, or simply start the deamon with the command:

 # /etc/init.d/powstatd start

How It Works:

powstatd is initiated as a daemon in runlevels 2 to 5.

A UPS can only be in one of three states; OK, FAIL, or LOW. Usually, when the main power is on, the UPS is operating in the OK state; when power fails, the UPS changes to FAIL mode, from which it can either recover and return to OK (if the power is restored) or can move to LOW (if the battery starts running out of juice before the power returns).

powstatd monitors the UPS condition. When the state changes, *powstatd* writes the new state of the UPS in */etc/powerstatus* and then signals init (the mother of all processes) of the change in the UPS condition. The init process receives this notice (a SIGPWR interrupt) and checks */etc/powerstatus* to see if it contains "OK", "FAIL" or "LOW".

The contents of */etc/powerstatus* tells init (which is configured by the */etc/inittab* file) to run the script

/etc/init.d/powerfail with one of the arguments "start" (shutdown due to power line failure), "now" (shutdown NOW due to eminent UPS battery failure) or "stop" (cancel shutdown before power is back online). The init process then removes */etc/powerstatus* so as not to be confused on subsequent interrupts.

/etc/init.d/powerfail start
: initiates a timed *shutdown -h* (halt) in background, on the assumption that if power is restored the shutdown can be cancelled.

/etc/init.d/powerfail stop
: cancels the running shutdown and notifies all users that power is restored and no shutdown is imminent.

/etc/init.d/powerfail now
: cancels the running shutdown and initiates an immediate *shutdown -h* in foreground; this means once the UPS tells you the battery is low, you will indeed shutdown (there is no recovery).

Note that as you halt the machine, the shutdown sequence (/etc/init.d/halt) invokes *powstatd* one last time, but this time with the kill flag (-k), forcing the UPS to turn off, but only if the UPS is indeed in either the FAIL or LOW state (in any case, most supplies will ignore the kill signal if power is still available). In this fashion, once the power eventually returns (even after a day or two), the system should automatically restart without intervention.

Troubleshooting:

1. If your machine keeps shutting down even when the power is on, you're probably watching the wrong serial line. To recover, try rebooting in single user mode (issue "linux 1" at the LILO prompt) and disable *powstatd* by renaming */etc/powstatd.conf* to something else. Reboot and you should be able to fix the configuration.

2. Some older UPS systems as well as some homebuilt cable connections (see the UPS HowTo) may require that UPS shutdown signals be sent on the transmission line rather than on one of the signaling lines. Usually, this implies that a serial break signal (e.g., a long series of zeros) is required for the UPS to shut down. For these situations, you can use the special configuration file command

```
kill break
```

to obtain the appropriate behavior.

4. If your slave machines keep rejecting seemingly valid status messages from the master when using encrypted master/slave communication, first make sure they are running the same version of the software. Otherwise, try changing the definition of "outdated" at compile time (MAXCLOCKDRIFT) and recompiling. If you are not using NTP or some other mechanism to ensure that master and slave clocks are reasonably synchronized, you may well be better off running powstatd compiled without -DSECURE.

5. Some older versions of init are not capable of invoking one of several different scripts (e.g., powstatd.ok, powstatd.fail, powstatd.low) on receipt of SIGPWR under different circumstances. Instead, they always invoke the same script, which must then handle the control logic (i.e., deciding whether to shutdown gracefully, abort a shutdown, or shutdown immediately) internally (SPARC/Solaris and SGI/Irix apparently fall in this catagory). If you have such a version of init, look at the sample script *powstatd.dumb* included in the distribution for an example of how to go about handling this case. (Debian's init is fine in this respect.)

ACKNOWLEDGEMENTS

I learned a lot from reading the publically-available source code for other UPS monitoring packages, like *genpower, powerd* and *upsd.* I wrote *powstatd* primarily because the exact combination of features or

configuration options I required were not available with these other packages.

Peter Galbraith (galbraithp@dfo-mpo.gc.ca), the *powstatd* Debian package maintainer, provided numerous suggestions and bug fixes, as did Philippe Troin (phil@fifi.org), who was the first to suggest using, e.g., MD5-based digital signatures to avoid malicious shutdowns. Nick Holgate (holgate@debian.org) suggested an extension to *powstatd* signaling capabilities in order to support UPS systems that require a break signal for shutdown. Nicolas Simonds (nic@bridge.com) provided information about changes to the code for both SPARC/Solaris and SGI/Irix.

TEA, or the *Tiny Encription Algorithm*, is due to David Wheeler and Roger Needham of the Cambridge Computer Laboratory. Their implementation, used here, is in the public domain.

Alberto Maria Segre
segre@cs.uiowa.edu
S378 Pappajohn Building
The University of Iowa
Iowa City, IA 52242-1000.