

Electricidad a prueba de apagones

VIDA PROLONGADA

Un sistema de alimentación ininterrumpida puede ayudarnos ante un apagón, evitando dañar el hardware y permitiéndonos guardar la información. Un script Nagios escrito en Perl verifica el SAI e inicia un apagado controlado si la unidad se acerca al límite de la capacidad de su batería. **POR MICHAEL SCHILLI**

Mientras husmeaba en mi trastero, lleno de viejo hardware, encontré un antiguo y barato sistema de alimentación ininterrumpida (SAI). Debí haberlo comprado en rebajas, ya que el “Cyberpower 325SL” sólo proporciona 185 W durante unos cinco minutos. Pero supuse que era suficiente para ayudar a mi ordenador, módem ADSL y router a superar indemne un corto apagón.

Comunicación Estilo 1999

Este SAI dispone de una interfaz serie (Figura 1) que puede conectarse a un ordenador mediante un cable serie. ¡A esto es a lo que yo llamo retro! La unidad incluía un software sólo para Windows que recuerdo descarté nada más abrir la caja. Buscando por la Web, sin embargo, encontré el proyecto NUT [1], que ofrece drivers para todo tipo de sistemas SAI, y un demonio para la comunicación con la unidad SAI a través de los drivers.

La instalación de NUT fue instantánea. Tras compilar y ejecutar *make install* se crean tres archivos de configuración. El primero, *ups.conf*, fija los parámetros del SAI utilizado (véase la Figura 2). En mi caso, elegí el driver *genericups*, que funciona con los SAIs baratos y sólo ofrece información básica del estado, sin estadísticas ni información de la capacidad de la batería restante. Un estudio de la documentación del driver reveló que para las unidades CyberPower debía usarse el “type 7”. Denominé al dispositivo “elcheapo”, y es así como se referenciará más tarde a la hora de verificar el estado.

Como conecté el cable serie al segundo puerto serie de mi ordenador, el puerto configurado es */dev/ttyS1*. Si lo hubiese conectado al primero se habría denominado */dev/ttyS0*.

Acceso Limitado

El archivo de configuración del demonio *upsd.conf* define las reglas acerca de

quién podrá acceder a la información del SAI desde el demonio NUT (véase la Figura 3). Abrí el archivo de mi dirección IP estática, y el */32* al final define que sólo puede leerse la información del estado.

Para uso doméstico, probablemente es demasiado engorroso definir el acceso de usuario a través de *upsd.users*, pero el archivo tiene que estar allí, por lo que valdrá con un archivo vacío.

Podríamos definir un nuevo usuario “nut” con un grupo asociado, pero decidí dejar que el demonio se ejecutara con el usuario por defecto *nobody*, para el cual tenemos que crear un directorio de estado:

```
# mkdir /var/state/ups
# chown nobody /var/state/ups
# chmod 700 /var/state/ups
```

A continuación, el demonio del driver y luego el demonio NUT deben iniciarse como root:



Figura 1: El sistema SAI con dos enchufes en la parte superior y un cable serie conectado a la derecha. El medidor de consumo indica que el PC, el módem ADSL y el router combinados están usando unos 114 W.

```
mschilli@mybox:~
# ups.conf
[elcheapo]
driver = genericups
port = /dev/ttyS1
upstype = 7
desc = "el cheapo ups"
2.0-1 All
```

Figura 2: La configuración del SAI en /usr/local/ups/etc/ups.conf.

```
mschilli@mybox:/usr/local/ups/etc
#upsd.conf
ACL all 0.0.0.0/0
ACL localhost 192.168.0.18/32
ACCEPT localhost
REJECT ALL
2.0-1 All
```

Figura 3: La configuración del demonio NUT en /usr/local/ups/etc/upsd.conf.

```
# /usr/local/ups/bin/U
upsdrvctl start
# /usr/local/ups/sbin/upsd
```

Si la salida de estos comandos indica que se realizó con éxito, podemos empezar haciendo una pequeña prueba con la utilidad *upsc* (que se incluye con NUT), la cual revela que el SAI está en línea, se encuentra alimentándose de la toma de corriente:

```
$ upsc elcheapo@localhost
ups.status
OL
```

Al desenchufar el SAI de manera que alimiente el ordenador desde la batería, la llamada anterior a *upsc* devuelve el valor *OB* en lugar de *OL*.

En el Mundo de Nagios

¿Cómo podríamos monitorizar el SAI? Los lectores habituales recordarán que hemos hablado de Nagios en esta sección con anterioridad [2]. Nagios vigila todo tipo de sistemas en mi casa, incluyendo la temperatura del dormitorio, la capacidad del disco duro y el rendimiento del servicio de hosting que uso para mis páginas Web. Por tanto, decidí añadir un vigilante del SAI como otra tarea de Nagios dentro de la configuración existente (véase la Figura 4).

El script del Listado 1, *check_myups*, usa la utilidad *upsc* mencionada anteriormente para consultar el estado del SAI, y añade un contenedor de forma que el script pueda usarse como un plugin de Nagios. Usa algunos módulos Perl

extra, que pueden descargarse desde CPAN. Si el SAI está funcionando y la verificación se realiza con éxito, el script muestra *UPS OK - OL* y devuelve un código de salida de 0. Si el SAI está funcionando con sus baterías, el script devuelve *UPS CRITICAL - OB* y genera un código de salida 2 para indicarle el problema a Nagios.

Y... ¡Acción!

Nagios sigue el criterio de cambios de estado "suaves" y "duros". Si una verificación indica una condición crítica por primera vez pero el parámetro *max_check_attempts* está configurado a 3, Nagios toma nota del problema aunque fija el estado a *SOFT*. Si las siguientes verificaciones, *retry_check_interval* unos segundos después fallan también, y

Listado 1:check_myups

```
01 #!/usr/bin/perl
02 #####
03 # check_myups
04 # Mike Schilli, 2007
05 #####
06 use strict;
07 use Log::Log4perl qw(:easy);
08 use Nagios::Clientstatus;
09
10 my $version = "0.01";
11 my $ncli =
12
13 Nagios::Clientstatus->new(
14     help_subref => sub {
15         print "usage: $0\n";
16     },
17     version => $version,
18     mandatory_args => [],
19 );
20 my $data = `upsc
21     elcheapo@localhost
22     ups.status`;
23
24 chomp $data;
25 my $status = "ok";
26
27 if ($data eq "OB") {
28     $status = "critical";
29 }
30 print "UPS ", uc($status),
31 " - $data\n";
32 exit $ncli->exitvalue(
33     $status);
```

finalmente se alcanza el valor *max_check_attempts*, se cambia el estado a HARD y se disparan los mecanismos de notificación. Es posible configurar correos electrónicos para enviar al administrador de sistemas de manera que tenga un sobresaltado despertar.

Nagios también nos permite iniciar acciones cuando ocurren cambios en el estado. De este modo, los scripts pueden intentar resolver el problema, y si la solución es tan simple como reiniciar un servidor Web, puede ser buena idea

```
mschilli@mybox:~/DEV/articles/nagios/eg
#####
# ups Plugin
#####
define command{
    command_name ups
    command_line $USER1$/check_myups
}

#####
# ups Service
#####
define service{
    use                ez-service
    host_name          mybox
    normal_check_interval 1
    retry_check_interval 1
    event_handler      powerdown
    max_check_attempts 2
    service_description UPS Check
    check_command      ups
}

#####
# powerdown
#####
define command{
    command_name      powerdown
    command_line      $USER1$/powerdo
    un $SERVICESTATE$ $SERVICESTATETYPE$
}

254.1 53%
```

Figura 4: La configuración de Nagios para el verificador del SAI y su manejador de eventos de apagado.

Slimserver	OK	04-21-2007 15:04:14	27d 2h 39m 36s	1/1	HTTP OK HTTP/1.1 200 OK - 289 bytes in 0.002 seconds
UPS Check	CRITICAL	04-21-2007 15:50:01	0d 0h 15m 0s	3/3	UPS CRITICAL - OB
Webserver80	OK	04-21-2007 15:45:58	27d 3h 39m 17s	1/1	HTTP OK HTTP/1.1 200 OK - 289 bytes in 0.001 seconds
Wireless	OK	04-21-2007 14:52:29	1d 2h 52m 42s	1/4	WL OK - wl is off

Figura 5: Nagios muestra que el SAI está quedándose sin batería.

hacer esto en lugar de molestar al administrador del sistema. En el caso de un SAI que tengamos en casa, que se quede sin batería, simplemente necesitamos que el sistema Linux se apague correctamente y evitar así un aterrizaje abrupto. El manejador de eventos del Listado 2, *powerdown*, hace exactamente esto.

Pero debemos tener cuidado, Nagios llama al script definido con la directiva del manejador de eventos cada vez que cambia el estado: primero, cuando el estado cambia de OK a CRITICAL/SOFT, luego cuando cambia de CRITICAL/SOFT a CRITICAL/HARD y finalmente tras recuperarse de CRITICAL/HARD a OK.

El script *powerdown*, que toma la información del estado que se le pasa, asegura que sólo se apague el equipo si ocurre un estado crítico HARD e ignora el resto de peticiones.

Según la configuración de la Figura 4, Nagios pasa las dos variables *SERVICESTATE* (ok/ critical) y *SERVICESTATE* (soft/ hard) de forma que *powerdown* pueda tomar una decisión acertada.

Si nos estamos quedando sin batería, el manejador de eventos intenta ejecutar el script *poweroff*, pero sólo el usuario root puede hacerlo. Para asegurarnos de que el usuario *nagios* que ejecuta la aplicación Nagios tiene permiso para ello, se

añade la siguiente entrada al archivo *sudoers*:

```
# /etc/sudoers
nagios ALL= NOPASSWD: /usr/bin/poweroff
```

Cuando se configura esto, el usuario *nagios* puede ejecutar *sudo /usr/bin/poweroff* como root sin tener que teclear una contraseña. Ambos scripts, *check_myups* y *powerdown*, han de instalarse en el directorio de plugins Nagios, */usr/local/nagios/libexec*. Una vez realizados los cambios en la configuración de Nagios, éste necesita reiniciarse.

¡Ahora ya estamos preparados para el próximo apagón!

RECURSOS

- [1] Network UPS Tools (NUT): <http://www.networkupstools.org/>
- [2] "El Vigilante" por Michael Schilli. Linux Magazine – Edición en Castellano, Número 20: <http://www.linux-magazine.es/issue/20/PerlNagios.pdf>
- [3] Turnbull, James. "Pro Nagios 2.0". Apress, 2006.
- [4] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/32/Perl>

Listado 2: powerdown

```
01 #!/usr/bin/perl -w
02 #####
03 # powerdown event handler
04 # Mike Schilli, 2007
05 #####
06 use strict;
07 use Sysadm::Install qw(:all);
08 use Log::Log4perl qw(:easy);
09
10 Log::Log4perl->easy_init({
11     file =>
12
13     ">>/tmp/powerdown.log",
14     level => $DEBUG
15
16     my ($state, $softhard) =
17         @ARGV;
18
19     LOGDIE
20         "usage: $0 state
21         SOFT|HARD"
22         if !$softhard
23         or $softhard !~
24         /SOFT|HARD/;
25     DEBUG
26     "Called $0 $state $softhard";
27     if ($state eq "OK") {
28         DEBUG "Ignoring OK";
29         exit 0;
30     }
31
32     if ($softhard eq
33         "SOFT") {
34         DEBUG "Ignoring soft
35         mode";
36         exit 0;
37     }
38     # Shut PC off
39     tap("sudo",
40         "/usr/bin/poweroff");
```