

## 1. Foreward

The Linux 2.6 kernel enhanced the management of attached devices through the introduction of udev. Udev provides users with a persistent naming process for all devices across reboots. As Emulex contributed the 8.x driver to the Linux upstream kernel source pool, Emulex was asked to use the existing Linux features to provide persistent naming of devices. Emulex met this challenge by working with a few different applications, such as devlabel, and choosing udev because of its ability to use sysfs and native Linux applications to provide unique lds that make persistent names possible. The major Linux distributions, Red Hat and Novell/SuSE, following the lead of kernel.org, also required the usage of the new mechanisms and the removal of the old mechanisms. While providing kernel-based persistence, udev also provides LUN-level persistence, which resolves the "LUN walking" issue found with WWPN-based solutions.

## 2. Persistent Configuration of sd Devices

There are two steps involved with configuring the system with persistent naming for sd devices.

- Discover the mapping of logical devices (sd's) to physical devices (disks).
- Insert this mapping information into the udev rules file to persist the data between reboots.

### 2.1. Using udev to discover logical to physical mappings for sd devices

The following example demonstrates how to use udev to discover the mapping between sd devices and physical devices. This example has an Emulex dual port HBA with one port connected to a JBOD and the other port connected to a Clariion™ CX300 target array. The topology is fabric.

In this discussion, # is the prompt and root login is assumed. The first step is to make local modifications to system files.

```
[root@localhost ~]# cd /etc/udev/rules.d
```

Examine the files available here. There should be a file called 50-local.rules. You must create a file that is lexically less than 50-local.rules.

```
[root@localhost ~]# touch 20-local.rules
```

Next, write to this file the following line. Note that this line instructs udev to execute a page code 0x83 call to scsi\_id for all sd block devices on the scsi bus and create a symbolic link under /dev/disk-"page code 0x83 return string+kernel instance number". What scsi\_id actually returns is the target-assigned unique id per LUN. Save and close the file. Note that the choice of name in the SYMLINK is chosen to illustrate the process of finding the unique id only.

```
KERNEL="sd*", BUS="scsi", PROGRAM="/sbin/scsi_id", SYMLINK="disk-%c%n"
```

Now, modify the scsi\_id.config files with the following changes. Save and close the file.

```
[root@localhost ~]# vi /etc/scsi_id.config
options=-b ==> ##options=-b
##options=-g ==> options=-g
```

Restart udev to pull in the new naming specifications.

```
[root@localhost ~]# udevstart
```

## 2.2. Updating the udev rules to persist the logical to physical mappings

The first step involves locating the names of the disks gathered as a result of the step above. For this example, the Clariion™ array has 256 luns and the driver only sees 2 disks in the JBOD because of switch zoning.

```
[root@localhost ~]# cd /dev
[root@localhost ~]# ls -al disk*
```

*These names are the JBOD disks attached to one hba port -*

```
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-32000000c5005df61
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-32000000c5005df6d
```

*These names are the luns managed by the Clariion™ Array – Note that this list is not exhaustive.*

```
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-360060160c5101100009fe284a90dd911
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-360060160c5101100019fe284a90dd911
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-360060160c5101100029fe284a90dd911
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-360060160c5101100039fe284a90dd911
lrwxrwxrwx 1 root root 4 May 20 16:41 disk-360060160c5101100049fe284a90dd911
lrwxrwxrwx 1 root root 3 May 20 16:40 disk-360060160c5101100085c38bbd781d911
```

*This mapping is the local scsi disk.*

```
lrwxrwxrwx 1 root root 4 May 20 16:18 disk-SFUJITSU
```

At this point, you need to understand the association of the unique identifiers obtained above to the storage entity that generated them. There are a variety of ways to obtain this association, such as scripts that invoke some cli-base application capable of communicating with the storage entity and then build the rules file automatically, or use the storage entity's management utility.

In this example, the Clariion™ management utility, Navisphere™, provided the unique id to LUN association needed to complete the rules file. For this example, lun 0 was found to have unique id 360060160c5101100085c38bbd781d911.

Returning to 20-udev.rules, you write this set of rules. In this example, the name is chosen to illustrate the process only.

```
BUS="SCSI", SYSFS{vendor}="DGC", SYSFS{model}="ST318453FC", PROGRAM="/sbin/scsi_id
-p 0x83 -g -s /block/%k ", RESULT="*360060160c5101100085c38bbd781d911*",
NAME="fc_lun0_%k"
BUS="SCSI", RESULT="*Next_Unique_ID_Here*", NAME="fc_lun1_%k"
BUS="SCSI", RESULT="*Next_Unique_ID_Here*", NAME="fc_lun2_%k"
```

And so on for all luns on this target array.

Note that the rules are context sensitive - that is rule "BUS= ..." sets context for subsequent rules for Next\_Unique\_ID\_Here so that the vendor, model, and program fields don't need to be repeated.

Of course, once the udev rules file is touched, you must restart udev each time. Once the rules file is debugged and ready, reboot your system.

## 3. Creating A Custom Ram Disk For Boot-From-San Using Persistent Names

In this example, lun 0 on the Clariion™ array has the boot-from-san Linux distribution installed. The goal is to get this lun mapped by udev when the initial ram disk runs. The following text builds on the previous example.

The implementation starts by unpacking the existing initrd file in the /boot directory. The initrd file has to be unpacked into a new directory since additional files get inserted into its infrastructure. The following write up is based on RHEL4 Update 0, and kernel revision 2.6.9-5.ELsmp.

On the initiator system as root, change directory into /boot and look for initrd-2.6.9-5.ELsmp.img. This is a gzip compressed Unix data file, with maximum compression.

As a safety step, create a new directory under /boot to hold the contents of the initrd. This effort also requires a tmp directory so just create that now as well.

```
[root@localhost ~]# mkdir working_initrd
[root@localhost ~]# mkdir /working_initrd/tmp
```

Uncompress and unpack the initrd image in /boot to the same prefix name in /root.

```
[root@localhost ~]# cd /boot/working_initrd
[[root@localhost ~]# gunzip -dc /boot/initrd-2.6.9-5.ELsmp.img | cpio -di
2460 blocks
```

Let's see the contents

```
[root@localhost ~]# pwd
/boot/working_initrd

[root@localhost ~]# ls
bin dev etc init lib loopfs proc sbin sys sysroot tmp
```

Now, this boot-from-san approach with udev requires scsi\_id to create the persistent names and scsi\_id needs to be a statically linked executable.

To get the scsi\_id application, install the udev source rpm into some working directory - scsi\_id is part of the udev source rpm. Open the udev Makefile and modify the following:

```
# Set this to create statically linked binaries.
USE_STATIC=false ==> USE_STATIC=true
```

```
# To build any of the extras programs, run with:
# make EXTRAS="extras/a extras/b"
EXTRAS ==> EXTRAS=extras/scsi_id
```

Then execute a top-level make. This step builds a statically linked scsi\_id utility. Now finish building up the infrastructure:

Add files (note: udev\_working\_dir is the path to the install udev057 rpm):

Note that if this system can already boot off the lpfc hba, then scsi\_transport\_fc.o should already be available in /lib.

```
[root@localhost ~]# cd working_initrd
[root@localhost ~]# cp /lib/modules/2.6.9-5.ELsmp/kernel/drivers/scsi/scsi_transport_fc.o ./lib
[root@localhost ~]# cp -Rv /etc/udev ./etc
```

```
[root@localhost ~]# cp /udev_working_dir/extras/scsi_id/scsi_id ./lib
[root@localhost ~]# cp /etc/scsi_id.config ./etc
```

Note that overwriting the udev.conf in the third step won't matter.

Rebuild initrd - Note the boot\_from\_san\_fc tag. Some unique identifier should be substituted here to distinguish this initrd from other initrd files available in grub.conf.

```
find . | cpio -c -o | gzip > ../initrd-boot_from_san_fc.img
```

Change /dev/sdba? and LABEL=xxx lines in /etc/fstab to use udev name (Note that /dev/sdba? is the boot disk - this will vary from machine to machine.)

Change /boot/grub/grub.conf default boot partition to use udev name in kernel command line,

```
kernel /vmlinuz-2.6.9-6.37.ELsmp ro root=/dev/fc_lun0_1
```

This completes the boot-from-san configuration.

## 4. Using udev with st devices

The goal for tape devices is the same as for disk devices. There must be a unique id that persists across initiator reboots and that does not change with discovery order.

The first problem is whether or not the tape device is one of many SCSI tape devices residing behind a Fibre Channel controller, or if it is an FC-Tape device. If it is an FC-Tape device, then the WWPN is unique and can be used to create the persistent name. In fact, the scsi\_id program should return this as the unique identifier with a single digit prefix.

If the FC controller has multiple SCSI tape devices behind it, the WWPN is not unique and the persistent name will have to use multiple information elements to build the unique id.

Here is an example of each. First, an FC-Tape device. This example uses scsi generic (sg) rather than the scsi tape driver.

```
[root@localhost ~]# scsi_id -g -s /class/scsi_generic/sg0
350060b000029b592
```

The value return has a leading prefix of 3. This value is the NAA type and what follows is the controller's WWPN.

Here is the same tape device and a scsi\_id call. The answer is the same.

```
[root@localhost ~]# scsi_id -g -s /class/scsi_tape/nst0
350060b000029b592
```

In both examples, -g was needed because the vendor and model for this tape device were not in /etc/scsi\_id.config

Here is another example to a different FC-Tape Vendor. Note that the answer is equivalent with respect to the leading digit and the WWPN.

```
[root@localhost ~]# /sbin/scsi_id -g -s /class/scsi_tape/nst0
```

35005076300015101

Here is an example of a FC-SCSI Tape device.

When the Emulex driver loads, the scsi midlayer discovers the scsi tape devices as follows

```
scsi scan: INQUIRY to host 14 channel 0 id 0 lun 0
scsi: unknown device type 12
  Vendor: ADIC      Model: SNC 4000      Rev: 42d4
  Type:   RAID      ANSI SCSI revision: 03
Attached scsi generic sg5 at scsi14, channel 0, id 0, lun 0, type 12
scsi scan: INQUIRY to host 14 channel 0 id 0 lun 1
  Vendor: ADIC      Model: Scalar 24      Rev: 227A
  Type:   Medium Changer      ANSI SCSI revision: 02
Attached scsi generic sg6 at scsi14, channel 0, id 0, lun 1, type 8
scsi scan: INQUIRY to host 14 channel 0 id 0 lun 2
  Vendor: IBM       Model: ULTRIUM-TD2    Rev: 38D0
  Type:   Sequential-Access      ANSI SCSI revision: 03
Attached scsi tape st0 at scsi14, channel 0, id 0, lun 2
st0: try direct i/o: yes (alignment 512 B), max page reachable by HBA 4503599627370495
Attached scsi generic sg7 at scsi14, channel 0, id 0, lun 2, type 1
scsi scan: INQUIRY to host 14 channel 0 id 0 lun 3
  Vendor: IBM       Model: ULTRIUM-TD2    Rev: 38D0
  Type:   Sequential-Access      ANSI SCSI revision: 03
Attached scsi tape st1 at scsi14, channel 0, id 0, lun 3
st1: try direct i/o: yes (alignment 512 B), max page reachable by HBA 4503599627370495
Attached scsi generic sg8 at scsi14, channel 0, id 0, lun 3, type 1
```

This log output shows a controller at lun 0, the medium changer at lun 1 and two scsi tape devices at luns 2 and 3.

Here is what the `scsi_id` call returns:

```
[root@localhost ~]# scsi_id -g -s /class/scsi_tape/nst0
1IBM      ULTRIUM-TD2      1110133831
[[root@localhost ~]# scsi_id -g -s /class/scsi_tape/nst1
1IBM      ULTRIUM-TD2      1110133994
```

Note that the unique Id is actually comprised of three value with space delimiters.

A udev rule needs to have a unique id for the device meaning all three parts of this returned string are required. To do this, use the following command.

```
[root@localhost ~]# scsi_id -u -g -s /class/scsi_tape/nst0
1IBM_____ULTRIUM-TD2_____1110133831
[root@localhost ~]# scsi_id -u -g -s /class/scsi_tape/nst1
1IBM_____ULTRIUM-TD2_____1110133994
```

Creating the udev persistent name for scsi tape has the same process as scsi disk once the scsi id call needed to extract a unique id is known.

Here is the rule for the FC-Tape device:

```
BUS="scsi", SYSFS{vendor}="HP", SYSFS{model}="ULTRIUM 3-SCSI", PROGRAM="/sbin/scsi_id -p 0x83 -u -g -s /class/scsi_tape/nst%n",RESULT="350060b000029b592", SYMLINK="fc_lun_st%n"
```

And here is the rule for the FC-SCSI Tape device:

```
BUS="scsi", SYSFS{vendor}="IBM", SYSFS{model}="ULTRIUM-TD2", PROGRAM="/sbin/scsi_id -p 0x83 -u -g -s /class/scsi_tape/nst%n",RESULT="1IBM_____ULTRIUM-TD2_____1110133831", SYMLINK="fc_lun_st%n"
BUS="scsi", RESULT="1IBM_____ULTRIUM-TD2_____1110133994", SYMLINK="fc_lun_st%n"
```

And finally, here is the output of the rule:

```
[root@localhost ~]# udevstart
[root@localhost ~]# ls -al /dev/fc*
lrwxrwxrwx 1 root root 3 Apr  7 15:03 fc_lun_st0 -> st0
lrwxrwxrwx 1 root root 3 Apr  7 15:03 fc_lun_st1 -> st1
```

## 5. Avenues Outside of SCSI\_ID

A look at `/etc/udev/rules.d/50-udev.rules` provides another mechanism to provide persistent names. Here is an example:

```
KERNEL="sr[0-9]*", BUS="scsi", PROGRAM="/etc/udev/scripts/check-cdrom.sh %k DVD-R", SYMLINK="dvdwriter%e"
```

In this example, the udev rule calls a script named “check-cdrom.sh” located in `patch /etc/udev/scripts`. The second avenue, therefore, is a custom script installed into the `udev/scripts` directory that, for example, walks the `sysfs` directory tree finding FC targets and devices by some value matching criteria.

Using the FC-SCSI tape example above, the script would:

1. cd to `/sys/class/fc_transport/`
2. for each target, cd to the device and cat the port name.
3. if the port name (WWPN) matches some criteria
4. cat the `hh:cc:tt:ll` tuple and parse it for the symlink names.

## 6. Other Considerations

As the system administrator, the mechanism used to derive the name needs some consideration. The name also requires some thought especially if many devices are addressed. The first decision is whether or not the udev rule uses the `scsi_id` program to communicate with the device and derive the unique id or whether the udev rule uses a script to parse static data.

Using `scsi_id` to communicate with the device has the advantage of providing actual device data in the form of model, vendor, and unique id derived from a page x83 read of the device. The other advantage of this approach is a FC target with multiple removable devices can have a device between lun 0 and lun n that fails, but the rule doesn't break as it is specific to the lun. For example, if lun 3 fails, the persistent name isn't impacted since it was written to address unique id data on each particular lun. Since it is likely the system administrator wants to bind a persistent name to a specific device, this would appear to be a decent solution.

However, if the solution doesn't require such specific bindings, then the script solution works well. The system administrator only needs to know what WWPN the target answers to and is not concerned with the Luns on the backend – perhaps because the target makes a guarantee about the unique id not explicitly bound to any particular device serial number.

Finally, all of the steps listed above are required on every server that needs a persistent name to one or more attached devices. Although the first thought might be that a custom rules file is necessary for every server, it is possible to write the 20-local.rules file to contain every vendor name and model supported in your SAN and use the scsi id program along with some standardized naming convention to provide a single file that is written once and loaded into every server.

## 7. Summary

The introduction of udev in 2.6 kernel-base Linux distributions provided a mechanism for system administrators to persistently name devices found internally or externally to the local system. This paper explores the effort and a-priori knowledge required to write udev rules that fulfill the notion of a persistent name.

Given a bit of effort with the storage device tools and other system tools discussed in this paper, udev does provide a useful mechanism for creating persistent names.

## References

1. [http://www.kroah.com/linux/talks/ols\\_2003\\_udev\\_paper/Reprint-Kroah-Hartman-OLS2003.pdf](http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf)
2. <http://www.reactivated.net/udevrules.php> by Daniel Drake (dsd)
3. [http://kernel.org/pub/linux/utils/kernel/hotplug/udev\\_vs\\_devfs](http://kernel.org/pub/linux/utils/kernel/hotplug/udev_vs_devfs) by Greg Kroah-Hartman
4. <http://linux.dell.com/devlabel/devlabel.html>
5. Conversations with James Smart.