Libsysfs

A programming interface to gather device information in Linux®

Ananth N Mavinakayanahalli <ananth@in.ibm.com>
Linux Technology Center
IBM India Software Lab

Daniel Stekloff <dsteklof@us.ibm.com>
Linux Technology Center
IBM Beaverton

16 January, 2004

Agenda

- Overview of the Driver Model and sysfs
- History and motivation behind Libsysfs
- The basic design of Libsysfs
- Subsystems representation in Libsysfs
- Calling conventions and API usage example
- Future
- Availability

Driver Model overview

- Developed by Patrick Mochel
- Primary intention – make Power Management tasks easy
- Abstracts common elements of different driver models to a standard set of structures
- Lends itself to usage in different scenarios
  - Hotplug
  - Device hierarchy representation
  - Power Management

Driver Model (contd...)

- Basic abstractions in the driver model
  - *Device* – a physical or logical system resource
  - *Driver* – software modules that manage physical or virtual devices
  - *Bus* – a medium to connect a set of similar devices
  - *Class* – an aggregation of similar objects or an aggregation of objects that perform a similar function

Sysfs – a brief overview

- RAM-based filesystem present in Linux Kernel v2.6
- A by-product of the design of the new driver model
  - *Original intention was to debug the new driver model*
- Always *built-in* with 2.6 kernels
- Provides different *views* of system devices
  - Hierarchical – topology tree of devices in the system
  - Bus specific – what devices are connected to what bus?
  - Class based – depending on the functionality of the device

History behind Libsysfs

- What was intended
  - Model specific subsystems to provide diagnostic information from sysfs to applications
  - Work only with device subsystems such as PCI, SCSI and USB
- Drawbacks of this approach
  - Code for individual subsystems was huge
  - Applications that provided diagnostic information on these subsystems already existed (lspci, sg3_utils, lsusb)
  - General feedback: *"The library should not interpret information. That is an apps' job"*

Motivation for Libsysfs

- Make sysfs access easy irrespective of the subsystem
- Provide a programmatic interface to sysfs
  - Provide a C API for applications to access sysfs
- Abstract sysfs structure from users
  - Applications need not know how information is organized
  - Provide a consistent interface to sysfs even though the sysfs structure underneath may change
- Application requirements
  - udev, sysdiag, Event Log Analysis, etc.

The *model the filesystem* approach

- Why?
  - Libsysfs was written to make access to a *filesystem* easy
- How?
  - Use a standard set of data structures that constitute the basic building blocks for the design
- Advantages
  - A subsystem agnostic method to access information
  - Data management and navigation easy

The *basic building blocks*

- Three structures constitute the basic building blocks:
  - sysfs_directory
    - Contains lists of subdirectories, attributes and links under the directory
  - sysfs_attribute
    - Contains the attribute name, value, size and its permissions
  - sysfs_link
    - Contains the name of the link and its target

Subsystems representation in Libsysfs

- The main device subsystems in sysfs have their own representation in Libsysfs
  - *bus*
  - *devices*
  - *class* (the *block* subsystem is considered a class)
- In addition, Libsysfs has a *driver* representation

The *device* representation

- Sysfs structure

```
ananth@...:~>tree /sys/devices/..../0000\:01\:08.0/

/sys/devices/pci0000:00/0000:00:1e.0/0000:01:08.0/

|-- class

|-- config

|-- detach_state

|-- device

|-- host0     --> this is a scsi host bus adapter

|-- irq

|-- resource

|-- subsystem_device

|-- subsystem_vendor

`-- vendor
```

- Libsysfs representation

```
struct sysfs_device {
        unsigned char *name;
        unsigned char *bus_id;
        unsigned char *bus;
        unsigned char *driver_name;
        unsigned char *path;
        /* Private: for internal use only */
        struct sysfs_device *parent;
        struct dlist *children;
        struct sysfs_directory *directory;
};
```

The *driver* representation

- Sysfs structure

```
ananth@...:~> tree /sys/bus/scsi/drivers/sd/

/sys/bus/scsi/drivers/sd/

|-- 0:0:5:0 -> ../../../../devices/pci0000:00/.../host0/0:0:5:0

`-- 1:0:0:0 ->
      ../../../../devices/pseudo_0/adapter0/host1/1:0:0:0
```

- Libsysfs representation

```
struct sysfs_driver {
        unsigned char *name;
        unsigned char *path;
        /* Private: for internal use only */
        struct dlist *devices;
        struct sysfs_directory *directory;
};
```

The *class_device* representation

- Sysfs structure

```
ananth@...:~> tree /sys/class/net/eth0/
/sys/class/net/eth0/
|-- addr_len
|-- address
|-- broadcast
|-- device -> ../../../devices/pci0000:00/.../0000:01:0a.0
|-- driver -> ../../../bus/pci/drivers/3c59x
|-- flags
|-- mtu
|-- statistics
|-- tx_queue_len
`-- type
```

- Libsysfs representation

```
struct sysfs_class_device {
        unsigned char *name;
        unsigned char *classname;
        unsigned char *path;
        /* Private: for internal use only */
        struct sysfs_class_device *parent;
        struct sysfs_device *sysdevice;
        struct sysfs_driver *driver;
        struct sysfs_directory *directory;
};
```

The *bus* representation

- Sysfs structure

```
ananth@...:~> tree -d /sys/bus/usb/
/sys/bus/usb/
|-- devices
|   |-- 1-0:1.0 -> ../../../devices/pci0000:00/.../usb1/1-0:1.0
|   `-- usb1 -> ../../../devices/pci0000:00/0000:00:1f.2/usb1
`-- drivers
    |-- hub
    |-- usb
    `-- usbfs
```

- Libsysfs representation

```
struct sysfs_bus {
        unsigned char *name;
        unsigned char *path;
      /* Private: for internal use only */
        struct dlist *drivers;
        struct dlist *devices;
        struct sysfs_directory *directory;
};
```

Calling conventions in Libsysfs

- API names are self explanatory
  - All ***sysfs_open_xxx*** functions have a corresponding ***sysfs_close_xxx*** function
  - All ***opened*** structures must be closed with a call to their corresponding ***close*** function
  - ***sysfs_get_xxx*** functions must be used to obtain handles to elements of opened structures that are lists or handles to other structures
- Refer *libsysfs.txt* (shipped with sysfsutils and udev packages) for the complete list of functions and their explanation

API usage example

- To obtain information about the network interface eth0, the sequence of calls would be:
  - *Get a handle to the class device*
    - struct sysfs_class_device *class_device = sysfs_open_class_device("net", "eth0");
  - *Get a handle to the list of attributes for this class device*
    - struct dlist *attrlist = sysfs_get_classdev_attributes(class_device);
  - *Get the physical device reference for eth0*
    - struct sysfs_device *device = sysfs_get_classdev_device(class_device);
  - *Get the driver reference that is used by the device*
    - struct sysfs_driver *driver = sysfs_get_classdev_driver(class_device);
  - *Close the class device*
    - void sysfs_close_class_device(class_device);

API usage example – udev

- One of the environment variables on a hotplug event will be a string of type “/block/sdb/sdb1”
- udev uses the following code snippet to get the sysfs_class_device for this class device

```
strcpy(dev_path, sysfs_path);
strcat(dev_path, device_name);
dbg("looking at '%s'", dev_path);
/* open up the sysfs class device for this thing... */
class_dev = sysfs_open_class_device_path(dev_path);
if (class_dev == NULL) {
        dbg ("sysfs_open_class_device_path failed");
        goto exit;
}
dbg("class_dev->name='%s'", class_dev->name);
```

Contd...

API usage example – udev (contd..)

- To get the *dev* attribute for this class device

```
struct sysfs_attribute *attr = NULL;
attr = sysfs_get_classdev_attr(class_dev, "dev");
if (attr == NULL)
        goto exit;
dbg("dev='%s'", attr->value);
```

- To get the *parent* class device reference

```
class_dev_parent = sysfs_get_classdev_parent(class_dev);
if (class_dev_parent)
                dbg("really a partition");
```

Applications that use Libsysfs

- Greg Kroah-Hartman's ***udev***
- IBM LTC's ***systool, lsbus*** – utilities shipped with sysfsutils package
- IBM LTC's ***sysdiag*** – diagnostics command line utility
- Patrick Mansfield's ***scsi_id*** utility, shipped as part of the udev package
- Christophe Varoqui's ***multipath*** utility, also shipped with udev
- The ***OpenHPI*** project

Resources

- Libsysfs is shipped as part of the sysfsutils package
- Latest version of sysfsutils can always be found at

  http://linux-diag.sourceforge.net
- Mailing list

  linux-diag-devel@lists.sourceforge.net

Future

- Solidify API
- Keep pace with changes in sysfs
- Add new interfaces as and when necessary

Disclaimers and Trademarks

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries
- Linux is a registered trademark of Linus Torvalds
- Other company, product or service names may be trademarks or service marks of others.

Thank You!