SCSI Transport Class Development

Mike Christie (mchristi@redhat.com)

Simplifying SCSI Transport Class Development and Use.

- Transport Class Issues.
  - Kernel.
  - Userspace.
- Possible Solutions.
  - Merge common SCSI abstractions.
  - Provide userspace apps with simplified API.

How to Make Userspace Maintainers Cry

- Using the SCSI sysfs API is hard.
  - lsscsi provides in depth transport information on a wide range of kernels.
    - lscsi maintainer had to read transport class maintainers minds and dig in the transport class code
  - Multipath-tools can barely figure out one transport.
    - Minimal Fibre Channel support.
    - Tools should be able to take advantage of transport timers, but they don't.

How to Make Kernel Maintainers Cry

- Target blocking API is difficult to use.
  - Some driver's transport class block/unblock use is racey.
  - Some of the drivers have implemented their own work around.
  - The rest have given up.
- Duplicate Functionality, but slightly different implmentations.
  - Port scanning, blocking, locking hacks.

Kernel Solution

- Provide abstraction for SCSI ports and I_T Nexus.
  - Scanning and blocking become standardized.
  - Eliminate need to port fixes between classes.
  - Transport classes can add/override common sysfs attrs for these objects like with scsi_hosts and scsi_devices, or add new objects under the SCSI ones.

SCSI port and I_T Nexus structs

```
struct scsi_transport_template {
        .......
        /* Classes can set these up to export
         * transport specific values for the object.
         */
        struct transport_container local_port_attrs;
        struct transport_container remote_port_attrs;
        struct transport_container i_t_nexus_attrs;
        .......
};

struct scsi_port {
        struct device dev;
        void *transport_data;
        /* this is just the sysfs id. The port values that make up the port
         * id for the transport are added via the the scsi_transport_template
         * attrs.
         */
        int id;
       /* scanning and recovery (dev loss tmo, fail fast tmo, etc) fields */
       ........
};

struct scsi_i_t_nexus {
        struct device dev;
        void *transport_data;
        int id;
};
```

SCSI port and I_T Nexus API example

- iSCSI class creating a target port.

```
iscsi_create_session(struct iscsi_port *initiator_port)
    |-scsi_add_i_t_nexus(struct scsi_port *initiator_port)
      {
            struct scsi_i_t_nexus *i_t_nexus;

            ..........

            i_t_nexus->dev.parent = &initiator_port->dev.
            device_register(&i_t_nexus->dev)
            ......

            scsi_add_target_port(i_t_nexus);
      }
      |-scsi_add_taget_port(struct scsi_i_t_nexus *i_t_nexus)
        {
              struct scsi_port *target_port;

              ..........

              target_port->dev.parent = &i_t_nexus->dev.
              device_register(&target_port->dev)
              ......
        }
```

Userspace Benefits

- Provide userspace with a common starting point to find port, and I_T Nexus values.

```
Tree
.....

scsi_host
.
|—scsi_port
|   |— scsi_i_t_nexus
|   |   |— scsi_port
|   |   |   |— scsi_target
|   |   |   |   |— scsi_device
```

Transport class specific values like iSCSI target name or FC port id would be placed under the scsi port or Nexus structs. Classes Could also add their own objects under the scsi ones.

Userspace Library

- Common kernel objects do not solve all of userspaces's problems.
  - Differences in sysfs layout between kernels.
  - Applications will still need to know something about the transport to get lower level transport values.
- Should there be a userspace library?