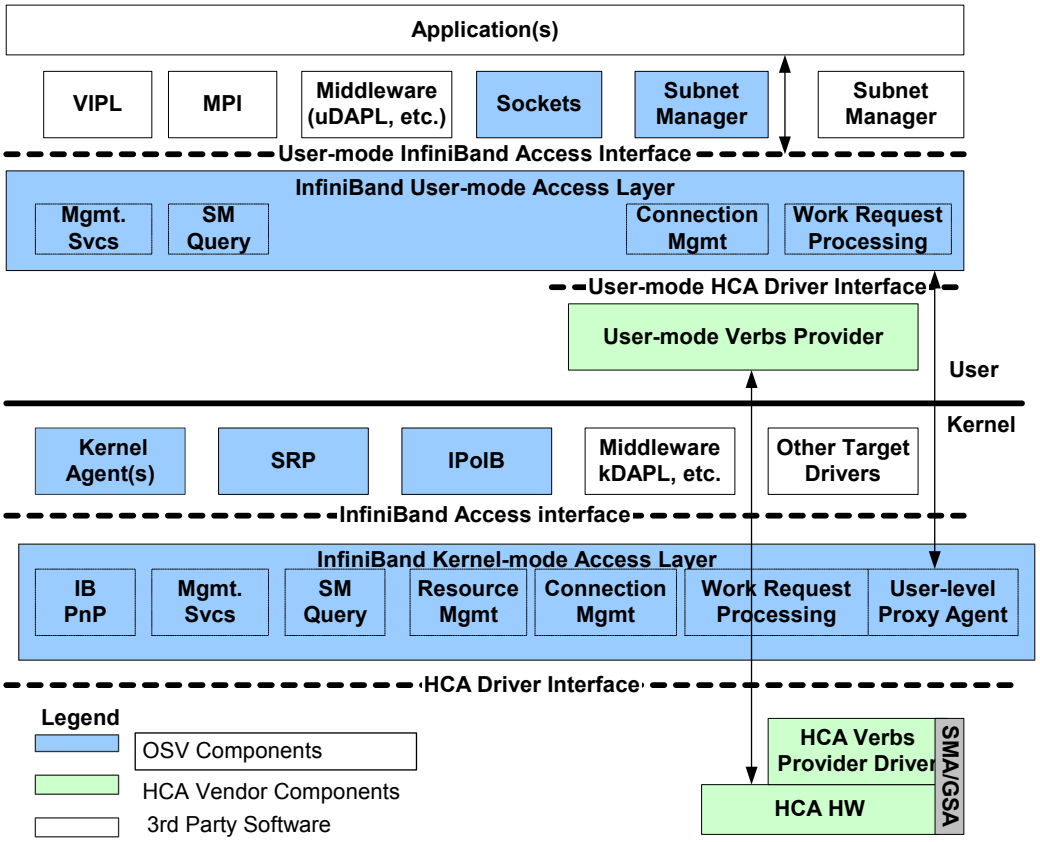
Software Architecture Specification (SAS)

Revision – Draft 2
Last Print Date: 4/19/2002 - 9:01 AM

InfiniBand™ Linux Operating System Software
SCSI RDMA Protocol

1.1 Architecture Overview



HCA – Host Channel Adapter provided by a HW vendor.

HW Verbs Provider Driver – This is a vendor-specific piece of software that together with the vendor's HCA forms a unit capable of implementing the verbs as specified in the InfiniBand™ specification.

HCA Driver Interface – This interface separates vendor-specific HCA code from code that is independent of any particular HCA vendor. The code that sits on top of this interface utilizes the functionality of the verbs without being dependent upon any particular vendor's implementation of the verbs.

InfiniBand Kernel-mode Access Layer– This software exports the full capabilities of the underlying HCA implementations to higher-level software. It exists to provide useful services that would otherwise be duplicated in the independent upper-level components, and to coordinate access to shared InfiniBand™ resources (such as the SMI and GSI) that are needed by multiple upper-level components. The InfiniBand Access component is envisioned to provide several services:

- **Resource Management** – This tracks HCA resource usage by upper level software components. This is useful for cleaning up those resources when/if the upper level components go away, and for having a means of tracking which upper level components should have access to which HCA resources.

- **Work Request Processing –** Process incoming work requests and dispatch the work request to the appropriate HCA drivers and handles the dispatching of work request completion notification to upper-level components. It also notifies interested upper level components of asynchronous events and errors that are detected by the vendor's HCA during its processing.
- **Connection Management** – Encapsulates the connection management protocol as defined in Chapter 12 of Volume 1 of the InfiniBand™ specification. This significantly simplifies the process of forming connections for those upper-level components that require them, and coordinates access to the GSI amongst those components.
- **SM Query** – Provides an interface to a standard set of subnet administration queries regarding the subnet configuration.
- **Management Services (SMI/GSI QP Access)**– Coordinates access amongst multiple upper-level components to the management queue pairs provided on each port of the vendor's HCA. This also routes incoming SMPs and GMPs to the appropriate upper-level components.
- **User-level Support Services (Proxy Agent)** – Interfaces to the user-mode InfiniBand™ Access Layer component to support all the services appropriate for user mode that need a kernel transition. Utilizes resources provided by both the host operating system and by the HW specific HCA verbs provider driver to accomplish this. The user-mode InfiniBand Access Layer communicates with the proxy agent through a set of IO control calls.
- **IB PnP** – The plug-and-play implementation for InfiniBand™ provided by the host operating system.

InfiniBand Access Interface – This exports the interface that all upper-level components within the kernel use to access the functionality provided by an HCA. The Proxy Agent also uses this interface to support some of the services provided to the user mode access layer.

Subnet Manager – Provides the basic Subnet Manager functionality as defined in Volume 1 of the InfiniBand™ specification.

Kernel Agents – Various kernel components provided by the OS vendor that utilize InfiniBand™.

Middleware – Transport and OS independent software layers that support InfiniBand™ and other RDMA capable transports like iWarp. Also provide an API that is portable across operating systems.

IB Target Drivers – Provides access to TCAs of various types. One example would be a driver provided by the host operating system that utilizes the SCSI RDMA Protocol (SRP) running on top of InfiniBand™ hardware to access InfiniBand™-attached storage. Another example would be a network driver that implements Internet Protocol (IP) over InfiniBand (IPoIB).

User-mode HCA Verbs Provider - Vendor-specific software in user-mode to assist with direct user mode IO (OS bypass) for data transfer operations (DTO).

User-mode HCA Driver Interface - This interface separates user-mode vendor-specific HCA code from code that is independent of any particular HCA vendor.

User-mode Access Layer – This modules exports capabilities of all the underlying HCAs that assist in developing upper layer protocols, like VIPL, uDAPL, SM, HCA Diagnostics etc., providing access to InfiniBand™ pimitives in user-mode. The user-mode InfiniBand Access Layer is a vendor independent shared library that can be dynamically loaded by the higher-level software components.

User-mode InfiniBand Access Interface – This is the interface that all applications running in user-mode use to access the underlying HCA. It attempts to minimize the level of abstraction of the underlying hardware while simultaneously being independent of the implementation of any particular vendor's HCA. It also provides the facilities to allow multiple applications to share access the HCAs.

Sockets, VIPL, MPI, and other messaging interfaces – user-mode implementations of all of these messaging interfaces can be built using the User-mode InfiniBand Access Interface.

Applications – Applications that wish to live "close to the metal" in order to fully exploit the capabilities of the underlying InfiniBand™ hardware and are willing to use a lower-level interface for optimum performance can access the User-mode InfiniBand Access Interface directly.

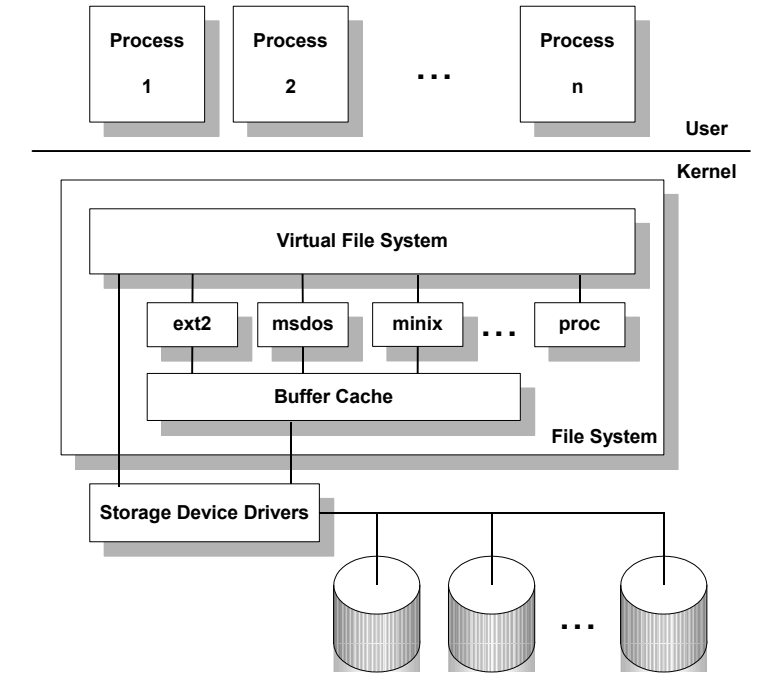
2. SRP Driver

2.1 Introduction

Host systems require access to remote storage devices across an InfiniBand fabric. The method used to access these devices is defined by the IO protocol. The SCSI RDMA Protocol (SRP) developed by the ANSI NCITS T10 working group is designed to take full advantage of the features provided by the InfiniBand Architecture. In addition, the SCSI command set is widely used throughout the industry and is applicable to a wide variety of device types. SRP allows a large body of SCSI software to be readily used on InfiniBand Architecture and is rapidly emerging as the protocol of choice for block-based storage. This section describes the architecture of the Linux SRP device driver.

2.2 Overview

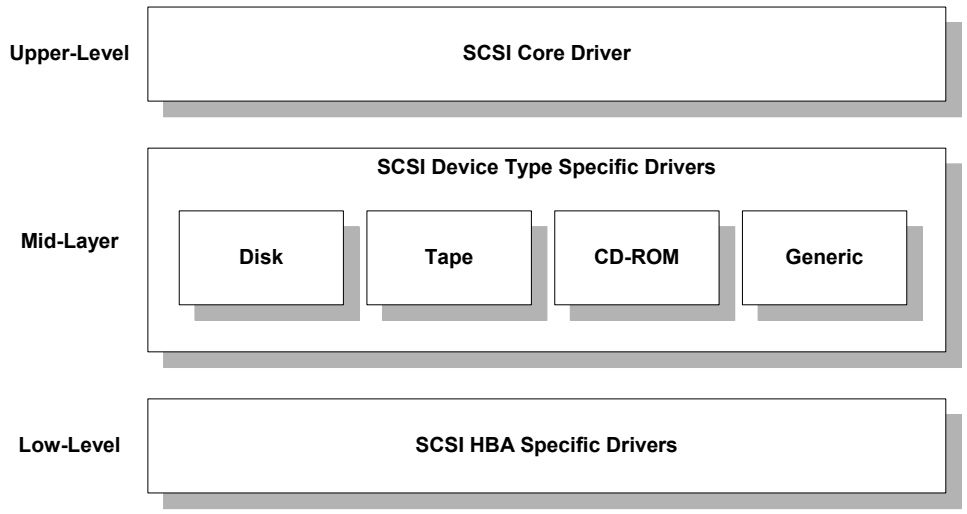
Use level processes typically access storage devices through a file system. The Linux operating system supports a number of file systems. The range of file systems supported is made possible through a unified interface to the Linux kernel known as the Virtual File System (VFS). The VFS interface provides a clearly defined interface between the kernel and the different file systems. VFS maintains internal structures, performs standard actions, and forwards tasks to the appropriate file system driver.



Linux File System Components

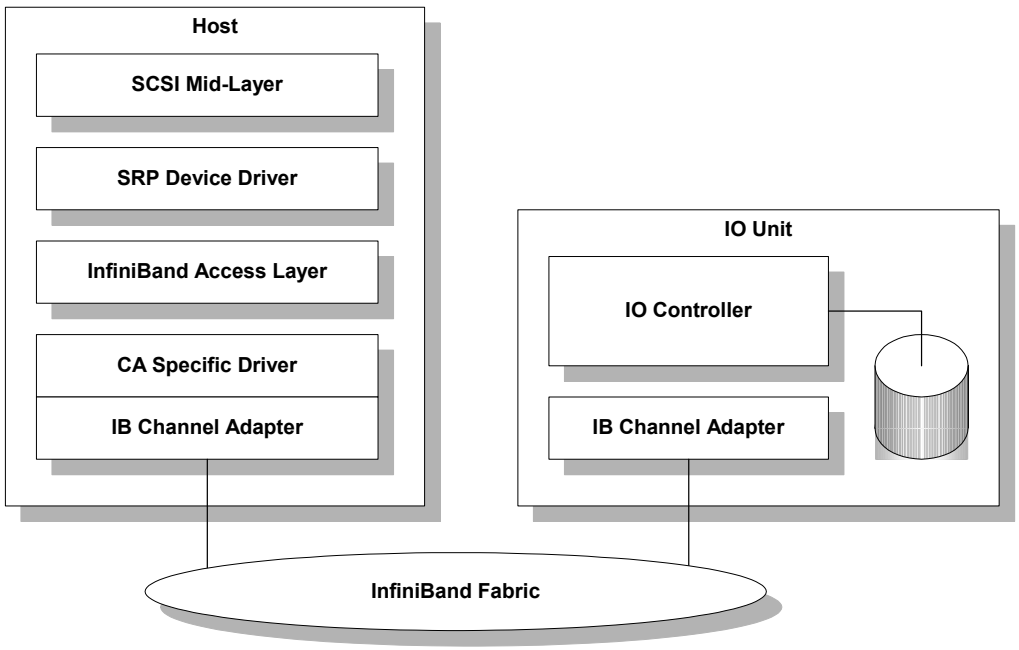
The central demand made of a file system is the purposeful structuring of data to allow high performance and randomized access. Linux employs a dynamic buffer cache to increase the performance of block devices accessed through the file system. In addition to storage device access through a file system, a process may also access a raw or character storage device directly, bypassing the file system and buffer cache components.

The storage device drivers provide access to physical devices by abstracting the details of the underlying hardware interface, for example IDE or SCSI. The SCSI device driver stack provides access to devices supporting SCSI protocols. The Linux SCSI framework contains an abstraction layer called the SCSI mid-layer. This layer provides upper-level drivers and applications with a device-independent set of interfaces to access devices. The SCSI mid-layer routes IO requests to low-level SCSI Host Bus Adapter (HBA) specific drivers. Low-level SCSI drivers provide access to devices through particular HBA hardware interfaces. Multiple low-level SCSI drivers may be loaded at the same time as required by the underlying hardware. This model simplifies the implementation of both the hardware-specific HBA drivers (the low-level SCSI drivers) and applications. The SRP device driver is implemented as a Linux low-level SCSI device driver.



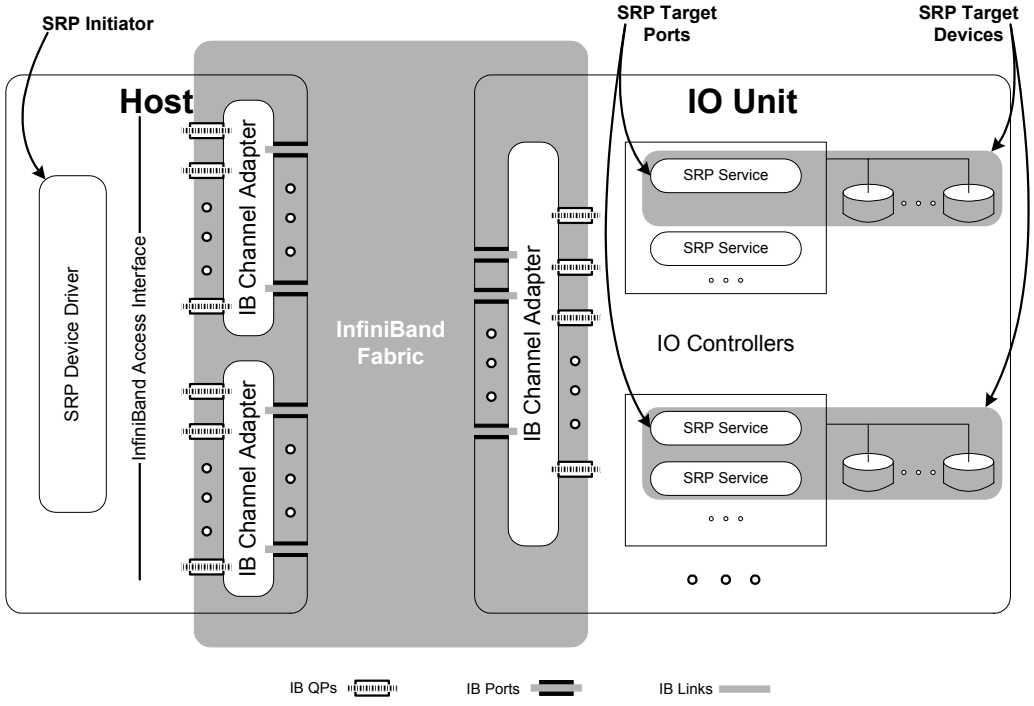
Linux SCSI Driver Stack

The SRP device driver differs from traditional low-level SCSI drivers in Linux in that the SRP driver does not control a local HBA. Instead, it controls a connection to an IO controller to provide access to remote storage devices across an InfiniBand fabric. The IO controller resides in an IO unit and provides storage services.



SRP Device Driver / IO Controller Relationship

The SRP device driver is known as the SRP initiator whereas the service on the IO controller is known as the SRP target. The SRP Specification created by the ANSI NCITS T10 working group defines the communication protocol used between the initiator and the target.



SRP Architecture Mapping

The SRP protocol provides transport services to enable a basic client-server model where an initiator presents SCSI tasks to a target for execution. All operations in this model use reliable service connections across the InfiniBand fabric. SRP defines the message format and behavior required to transfer commands and data between an initiator and a target. SCSI commands and completion status are exchanged asynchronously using message send operations.

SRP defines a message flow control mechanism that allows a target to limit the number of requests that may be queued on the target for execution. A target may use this mechanism to manage internal resources, for example, to dynamically allocate message buffers among multiple initiators. This allows the target to provide optimal use of limited resources and improve overall system performance.

In SRP, the target performs all device data transfers to or from initiator memory using RDMA operations. An initiator allows RDMA access from the target by registering its data buffer memory. Information describing the registered data buffer memory is included in the SRP command.

A typical SRP IO transaction is as follows:

1. The initiator builds an SRP request message that contains a SCSI command, a device logical unit number, and a data buffer memory descriptor, and sends the request to the target.
2. The target receives the SRP request and performs an RDMA operation to transfer the initiator data buffer memory contents to or from the device.
3. The target builds an SRP response message indicating the completion status of the request and sends the response to the initiator.

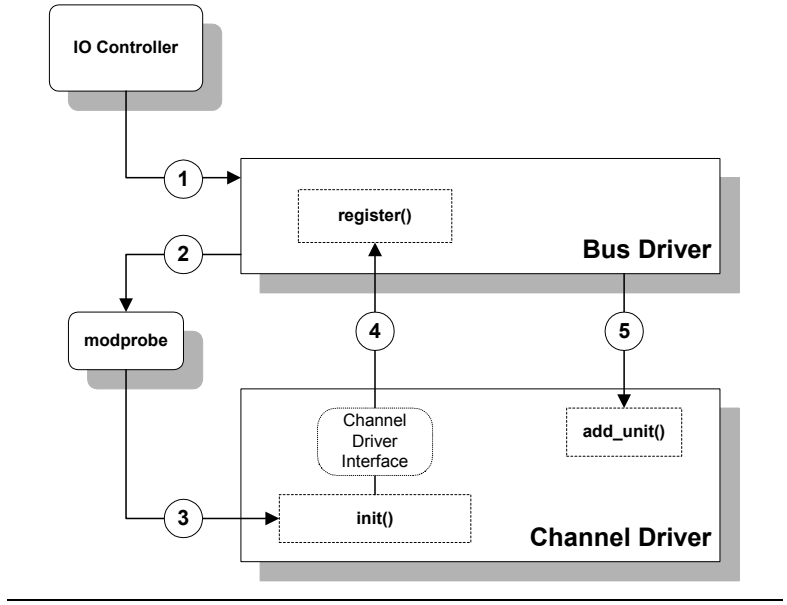
The initiator can also perform SRP task management operations, for example, to abort a task on the target. In addition, the target can send messages to the initiator describing asynchronous events, such as the insertion of new media into a removable device.

2.3 Theory of Operation

The following subsections describe the operation of the Linux SRP device driver.

2.3.1 Load / Unload

A bus driver is present on each host. The purpose of the bus driver is to inform the host of an InfiniBand attached IO controller that has become available or unavailable. The Linux bus driver maintains a mapping of IO controller profile class, subclass, and protocol identifier values to channel drivers. If a channel driver is not present at the time the bus driver needs to notify it of a new IO controller, then the bus driver will load the channel driver. The bus driver delivers add and remove notifications to a channel driver asynchronously through a function call interface registered with the bus driver when the channel driver is loaded.



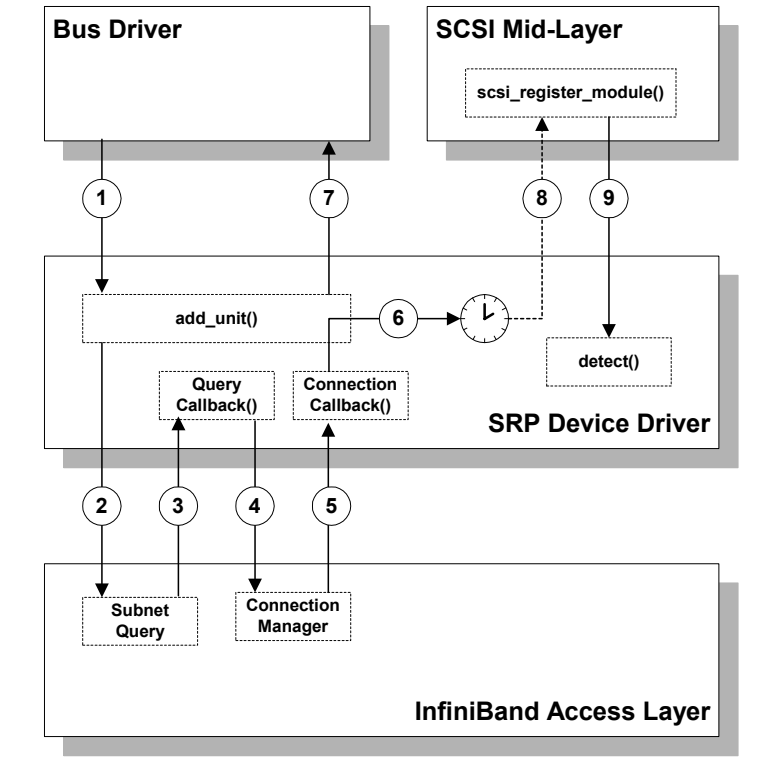
Channel Driver Load Sequence

1. The bus driver receives notification of a new IO controller.
2. The bus driver invokes a user-level application to load the appropriate channel driver.
3. The channel driver module is loaded and the driver initialization function is called.
4. The channel driver registers an interface with the bus driver.
5. The bus driver notifies the channel driver of a new IO controller instance.

2.3.2 Initialization

The Linux SCSI mid-layer provides a generic module initialization routine for low-level drivers to use. This routine is the first entry point called by Linux to initialize the driver. The generic init_module() function normally calls the Linux routine scsi_register_module(), which is a low-level SCSI driver routine to detect and initialize the hardware controlled by the driver. A Linux low-level SCSI driver has one chance to call scsi_register_module() to probe for devices and create device objects. Linux will not properly handle the addition of InfiniBand-attached IO controllers after the call to scsi_register_module() has been performed. For this reason, the SRP driver initialization routine is different from most low-level SCSI driver initializations. The scsi_register_module() is only called when the driver is certain that it has been informed of all SRP IO controllers assigned to the host. To achieve this certainty, the SRP driver queues a timer call to scsi_register_module() a small amount of time in the future, which sets a grace period after the SRP driver is notified of an IO controller. If another IO controller is added to the SRP driver during the grace period, the grace period starts over. If the grace period ends without a new call to add an IO controller,

scsi_register_module() is called allowing the mid-layer to detect devices, and complete the initialization process.



SRP Device Driver Initialization Sequence

1. The bus driver notifies the SRP device driver of a new IO controller instance.
2. The SRP driver issues a subnet query for connection path records.
3. The SRP driver query callback function is invoked.
4. The query callback function issues a connect request to the connection manager.
5. The SRP driver connection callback is invoked.
6. The SRP driver add_unit() function is signaled that the connection is established which starts the grace period timer.
7. The SRP driver add_unit() function returns to the bus driver.
8. The grace period timer expires without a restart and calls the scsi_register_module() routine.
9. The scsi_register_module() routine calls the SRP driver detect() function to determine the number of HBAs.

2.3.3 Connection Management

2.3.3.1 Connection Establishment

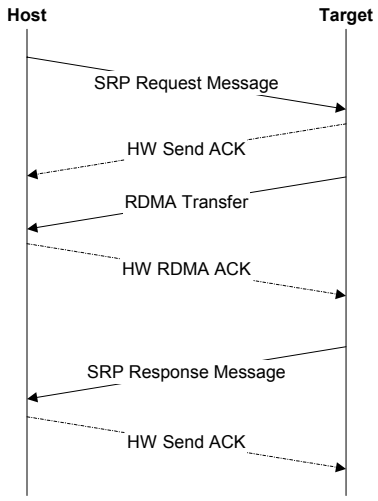
A switched fabric may allow several connection paths between a host and an IO controller. The SRP device driver obtains a list of one or more path records from the InfiniBand Access Layer and maintains an index into the path record list to establish a connection to the IO controller. If a path record query times out, the driver may retry the query. If a connection request times out, the driver may retry the request on the current path before advancing to the next path record in the list. If none of the paths is successfully connected, the driver will attempt to acquire a new list of path records from the access layer from which to restart the connection process. Once established, a connection will allow normal IO transaction processing.

2.3.3.2 Connection Failover

If the underlying channel adapter supports automatic path migration (APM) and multiple paths exist between the host and the IO controller, the SRP device driver will establish a connection with primary and alternate paths. Otherwise, the SRP device driver will perform connection error handling and failover in software. If an unexpected error occurs on an established connection, the SRP device driver will advance to the next path record in the list if possible and re-initialize the connection from that point, wrapping back to the beginning of the path list if necessary. Re-initializing a connection may include loading and enabling a new alternate path for APM or establishing a new connection through the connection manager.

2.3.4 Command Processing

SCSI commands are delivered to the SRP device driver through entry points provided when the driver registers with the Linux SCSI mid-layer. When the mid-layer has a SCSI command for a device controlled by the SRP driver, the queue_command() entry point is called with a pointer to a SCSI request structure as the argument. The SRP driver registers the host buffer memory, formats an SRP request, and sends it in a message to the target. Included in the request is the SCSI command and any necessary information describing the host buffer memory to or from which data will be transferred. The target interprets the request and executes the command using its local storage resources. Data movement between host memory and the target is facilitated by RDMA. The RDMA must be complete before the target sends a reply to the host. The target replies to the host with a message containing an SRP response. The SRP response notifies the host that the IO transaction has completed and the completion status. The host buffer memory is deregistered when the IO transaction completes.



SRP IO Transaction Diagram

The SRP device driver is event driven. These events include the following:

- Arrival of a SCSI command from the mid-layer
- Completion of a message send operation from the InfiniBand Access Layer (send messages are SRP requests or responses destined for the target)
- Completion of a message receive operation from the InfiniBand Access Layer (received messages are SRP responses or requests from target)
- Detection of error conditions

Note that because send and receive operations are independent, involve other software layers, and may utilize different completion queues, it is possible to receive an SRP response message from the target before the corresponding SRP send message request has completed. The SRP driver design must allow for this scenario.

2.3.5 Asynchronous IO

The SRP device driver has the ability to process multiple IO requests simultaneously. For each IO request from the SCSI mid-layer, the SRP driver maintains a data structure for the duration of the request's life – that is, from the time the request arrives until the time the completion response message arrives from the target. The data structure contains details about the request and its state. The address of this structure serves as the SRP driver context value. Asynchronous completion events use this context to associate the completion with one of perhaps many outstanding tasks.

2.3.6 Transaction Ordering

The SRP device driver provides an interface to random and sequential access devices. The SRP device driver does not maintain a different transaction ordering policy based on the class of device, it simply passes IO requests and responses through between the host client and device server. Therefore, to provide a uniform internal design and mode of operation, order is strictly maintained for all requests

and responses processed through the SRP driver. Multiple IO requests may be in process simultaneously, allowing the target device, adapter, or SCSI device class driver to reorder requests based on the policy of that layer to maintain data consistency.

2.3.7 Multithreading

The SRP device driver has the ability to process multiple IO requests simultaneously; therefore, critical data structures must be accessed using thread-safe mechanisms. The design of the SRP driver will minimize the extent of any locks to increase parallelism on SMP systems and to achieve maximum CPU effectiveness (IOps / % CPU utilization).

2.3.8 Interrupt Processing

The SRP device driver differs from traditional low-level SCSI drivers in Linux in that the SRP driver does not control a local HBA. Instead, the SRP driver controls a connection through an InfiniBand host channel adapter. Hardware interrupts from the channel adapter are handled by the HCA specific driver and delivered to the InfiniBand Access Layer. The access layer notifies the SRP driver of these events through a registered set of callbacks. The execution priority level of the callback is determined by the specific event and notification mechanism used by the InfiniBand Access Layer.

2.3.9 Error Handling

The SRP device driver can encounter errors from several sources. For example, errors may originate from the InfiniBand access layer, the IO protocol, the operating and IO subsystem interfaces, and the underlying IO device.

2.3.9.1 Transport Errors

The SRP device driver establishes connection-oriented, acknowledged channels that provide a reliable interface to remote IO controllers. Any transport errors occurring on the channels will result in the migration or destruction of that connection. When a transport error occurs, the SRP driver will attempt to failover to a new connection and resume IO processing.

2.3.9.2 Invalid Requests

It is assumed that the target is properly designed and implemented such that invalid messages are not received. If an invalid request or response message is received, the protocol engine will respond with the corresponding protocol error notification message to the Linux SCSI mid-layer.

2.3.9.3 Device Errors

Errors from the underlying devices are considered a natural part of IO processing. Such errors are converted into the appropriate error condition status and delivered to the Linux SCSI mid-layer for further processing. The SRP device driver makes no assumptions about the underlying device or the intended use of that device. The SRP device driver does not retry failed IO transactions. The underlying device, or the SCSI mid-layer may perform retries.

2.3.10 OS Interfaces

The SRP device driver is implemented as a Linux low-level SCSI driver. As such, the SRP device driver conforms to the standard OS low-level SCSI driver interfaces.

2.3.11 InfiniBand Access Interfaces

The SRP device driver resides above the InfiniBand Access layer and requires capabilities:

- Dynamically loaded on demand.
- Notification of IO controller addition or removal.
- Obtain an IO controller profile.
- Obtain the service entries for an IO controller.
- Obtain a list of path records to an IO controller.
- Create a reliable connected channel to an SRP target port.
- Send and receive SRP messages asynchronously.
- Separate or combined send and receive completion queues.
- Completion processing using callbacks or polling.
- Register / Deregister physical memory for remote access.
- Notification of channel errors.
- Notification of port state changes.