FCP for the IBM eServer zSeries systems: Access to distributed storage

by I. Adlung
G. Banzhaf
W. Eckert
G. Kuch
S. Mueller
C. Raisch

The IBM eServer zSeries™ FCP (Fibre Channel Protocol for SCSI) channel provides "Linux™ for zSeries" the capability to access storage devices using SCSI and FCP protocols, thus enabling it to make use of distributed storage. Leveraging the zSeries-unique virtualization approach for industry-standard storage devices as well, the zSeries FCP channel provides unique value in UNIX® and Linux environments. This paper describes the major differences between traditional and distributed storage attachments of zSeries systems. Furthermore, it describes the implementation of the zSeries FCP channel, its unique capabilities and characteristics, and how it is implemented by Linux for zSeries.

1. Introduction

While storage devices such as disks, streaming tapes, and CD-ROMs are often physically integrated into low-end and mid-range servers, high-end systems normally use externally attached and independently managed storage controllers. The z/Architecture* of the IBM eServer z800 and z900 systems defines I/O and storage attachment schemes that are vastly different from those typically used for servers based on Microsoft Windows**, UNIX**, or Linux** operating systems. zSeries*-specific storage controllers support channel programs based on channel command words (CCWs) defined by z/Architecture. Distributed storage controllers, on the other hand, must support the SCSI (Small Computer System Interface [1]) protocol, which is the dominant protocol for storage access in Windows, UNIX, and Linux environments. These two different I/O schemes are explained in more detail in Section 2.

Classical zSeries operating systems such as z/OS* and z/VM* were designed for use only with storage controllers that support the I/O protocols defined by z/Architecture. This changed with the advent of Linux for zSeries, since its storage I/O component is oriented toward SCSI protocols. Lacking the capability to access SCSI-based

0018-8646/02/$5.00 © 2002 IBM

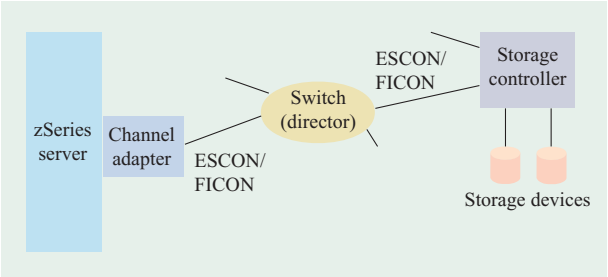

Figure 1

Storage attachment concept for zSeries servers.

storage devices on a zSeries server system, it was necessary to add specific support to Linux for zSeries to enable it to function in a CCW-based zSeries I/O environment. However, this additional layer in Linux for zSeries is unique to zSeries and does not allow it to exploit storage subsystems and applications that are dependent on a SCSI attachment. For this reason, an FCP attachment capability has been added to the z800 and z900 systems, allowing the attachment of SCSI-based storage controllers and enabling Linux for zSeries to access these controllers in the Linux-standard manner.

This paper describes the zSeries FCP channel from both hardware and software interface perspectives. Attention is given to capabilities which, until recently, have been specific to zSeries systems—for example, their capabilities with regard to virtualization and the sharing of I/O attachments and devices. Section 2 describes the storage attachment schemes typical of the zSeries and distributed storage environments, and explores the challenges faced when trying to combine them. An overview of the FCP and storage area networks is given in Section 3. Section 4 describes the key characteristics of the zSeries FCP channel. The unique internal protocol used for communication between host software and the zSeries FCP channel is described in Section 5. In Section 6, some scenarios are provided showing how the most common operations initiated by a host program are executed. Section 7 describes the hardware and firmware concepts used by the FCP channel. An overview of the unique reliability, availability, and serviceability characteristics of the zSeries FCP channel is given in Section 8. Finally, Section 9 describes the support that was added to the I/O stack of the Linux for zSeries operating system in order to exploit the FCP channel.

2. Storage attachment concepts

zSeries-specific storage attachments

The I/O component of z/Architecture, inherited from its predecessors such as the ESA/390 architecture, is based on channels, control units, and devices (see **Figure 1**) [2]. Channels provide a well-defined interface between a server and the attached control units. Originally implemented on parallel copper media, today's channels such as ESCON* and FICON*[1] use optical serial cables. Also, the parallel channel was a multi-drop interface, while both ESCON and FICON are switched point-to-point connections, extended to complex connection infrastructures via ESCON or FICON switches or directors.

ESCON channels use a unique physical interface and transmission protocol. FICON, on the other hand, is based on industry-standard Fibre Channel lower-level protocols. In both cases, however, the higher-level I/O protocol used by software, based on channel programs consisting of channel command words (CCWs), is unique to mainframes adhering to z/Architecture or its predecessors.

For access to disk storage, a specific set of CCWs is used, defined by the extended count key data (ECKD*) protocol. This protocol describes not only the commands and responses exchanged across the channel, but also the format of the data as it is recorded on the disk storage medium. For tape, similar CCW-based command protocols exist, but there are no associated access protocols for media such as DVDs or scanners, because the necessary software device drivers and control units have never been provided.

With z/Architecture, software addresses a storage device using a 16-bit device number, which uniquely identifies the device. Such a device number is mapped to one or more physical paths to the device. This path is typically described by an address quadruple consisting of a channel path identifier (CHPID), a physical link address, a control unit address, and a unit address (UA). The CHPID identifies the channel that provides a path to the device. The link address identifies a route through a switch or director. The control unit address specifies the control unit. Finally, the unit address identifies the device. In the case of a disk, this is often a logical device partitioned out of the disk space provided by an array of physical disks.

In order to provide redundancy and/or increased I/O bandwidth, or to allow load balancing, there is typically more than one physical path to a device. While these paths are specified via different address quadruples, software still uses a single device number to address the device, independent of the path that is chosen for any particular I/O request.

Another characteristic of the z/Architecture I/O scheme and the ECKD architecture for disk is the sophisticated support for sharing channels, control units, and devices

[1] Official standards based on ESCON and FICON, known as SBCON and FC-SB-2, respectively, have been adopted by the InterNational Committee for Information Technology Standards (INCITS).

among multiple operating systems, which may run on the same or different IBM zSeries systems.

Distributed storage attachments

The attachment of storage controllers in the distributed storage environment is predominantly based on the SCSI standard, using a control-block-based command-and-status protocol. Today, there is a clear distinction between the physical level and the command/status level of the SCSI protocol. On the basis of this higher-level SCSI command/status protocol, a new standard has been defined, designated as Fibre Channel Protocol for SCSI (FCP). The standard employs the SCSI command/status protocol above an underlying Fibre Channel transmission protocol. Because of the superiority of the optical Fibre Channel connection regarding speed, distance, and reliability, storage attachments via FCP have the highest growth rate in today's distributed storage market.

Traditionally, distributed storage controllers have been attached via parallel SCSI cabling, which allows only very limited distances between the server and the controller. And although the SCSI architecture has provisions for physical controller and device sharing, the length constraints of parallel SCSI cables impose natural limits on this sharing capability. Also, there has been little need for a capability to share controllers and devices among multiple operating systems running concurrently on the same server, because such server virtualization techniques are just starting to develop in the distributed storage environment.

Storage area networks (SANs)

This situation is changing with the advent of Fibre-Channel-based storage area networks (SANs). The Fibre Channel technology on which current storage area networks are typically based allows customers to considerably increase the distance between server and storage controller and to share controllers among multiple servers—features that have up to now been more commonly associated with mainframe-type systems such as the zSeries, which use ESCON or FICON and ECKD protocols, for example.

With storage area networks, the servers are logically organized around the SAN, which provides access to a large storage pool used to satisfy the storage needs of the connected servers. Such a SAN, often depicted as a storage "cloud" (see **Figure 2**), can be independently managed and serviced, freeing the servers from these chores.

zSeries-specific vs. distributed storage controllers

Two types of storage controllers with Fibre Channel interfaces are currently available: 1) those supporting the z/Architecture-specific FICON interface based on the CCW architecture, with the ECKD command set for disk storage and similar sets of commands for tapes; and 2) those supporting the SCSI-based FCP protocol, with SCSI command sets specific to the device type, such as disk or tape.

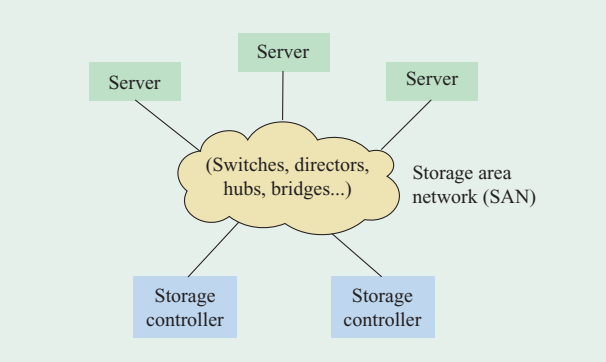

Figure 2

Storage area networks.

Both types of controllers use various internal interfaces to their devices, which may reside in the same enclosures or racks as the controller itself or in different ones. In most cases, these controller-to-device interfaces are standard interfaces, for instance parallel SCSI, electrical Fibre Channel–Arbitrated Loop (FC-AL), a Fibre Channel physical layer variant, or Serial Storage Architecture (SSA) [3], a serial, electrical version of SCSI. This type of internal attachment is independent of the controller "personality," i.e., whether it is a CCW-based or a distributed controller.

Both types of controllers are normally based on the same hardware building blocks. Their individual characteristics are achieved by different firmware handling the host interface. There are even some controllers that support both types of protocols (CCW and FCP) and can be configured to indicate which of their devices and connections are CCW-based and which use FCP.

3. Fibre Channel protocol

Fibre Channel networks consist of servers, storage units (controllers and devices), and interconnects (directors or switches) that enable any-to-any connectivity between the servers and the storage units. The interconnect as well as the communication between the servers and storage units is defined by the suite of Fibre Channel standards approved by the ANSI/INCITS T11 committee and currently being elevated to the ISO/IEC JTC1 committee as a set of international standards [4–8].

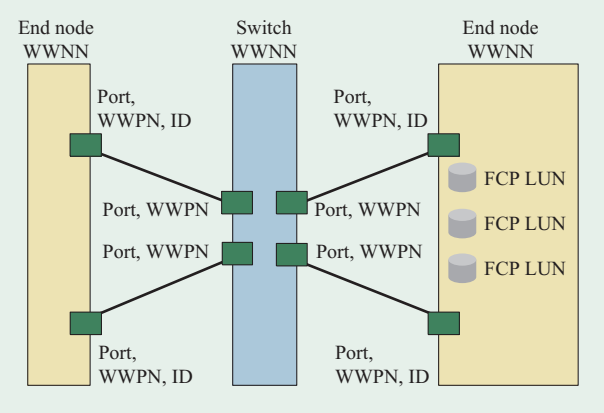

Figure 3

Fibre Channel addresses.

Fibre Channel architecture

The Fibre Channel architecture is a multilayer architecture consisting of five layers, FC-0 through FC-4. FC-0 describes the physical characteristics of a Fibre Channel network, such as the cables and connectors. Different optical and physical interfaces are defined. The zSeries FICON and FICON Express features support the optical Fibre Channel versions. FC-1 defines the transmission protocol, and FC-2 the signaling protocol. The FC-3 layer defines common services. Above these four layers, there are various FC-4 protocols, including HIPPI, IPI3, SB-2, and SCSI. SB-2 is the protocol on which the zSeries FICON channel is based [9]. This section focuses on FCP, the Fibre Channel protocol for SCSI.

Fibre Channel topologies and devices

In a Fibre Channel fabric, nodes are connected by physical point-to-point links, starting and ending at a port. Only in a point-to-point topology are both of these ports tied to end nodes. More often, one port is associated with an end node while the other one belongs to a Fibre Channel switch. Also, in the case of cascaded switches, both ports may be switch ports.

The Fibre Channel architecture defines three distinct topologies as interconnects between Fibre Channel end nodes.

The simplest topology is a direct "point-to-point" connection, typically between a host and an endpoint device such as a disk controller. After a connection has been established between these two nodes, the full bandwidth is available between the host and the device.

The term "arbitrated loop" defines a ring topology in which up to 127 nodes, both hosts and endpoint devices, share the Fibre Channel bandwidth. Arbitrated loops are often implemented using *hub* devices, where the loop is basically implemented within the hub while the end devices are connected to the hub. Hubs can also be cascaded to build more complex configurations.

"Switched fabrics" are implemented as switched connections between hosts and devices. One or more switches or directors are used to interconnect the end nodes in such a Fibre Channel network. Directors represent very reliable high-end switches without single points of failure. For simplicity, the term "switch" is used to refer to both Fibre Channel switches and directors in this paper. All connections in a switched fabric provide the full Fibre Channel bandwidth to each port.

In addition to these three Fibre Channel topologies, Fibre-Channel-to-SCSI bridges can be used to attach parallel SCSI devices.

Switches, hubs, and Fibre-Channel-to-SCSI bridges can all coexist in the same Fibre Channel network, and they are the building blocks for constructing large storage area networks.

The zSeries FCP channel attaches to Fibre Channel switches or directors. Fibre Channel loops are supported only as public loops attached to a switch, in which case the switch handles all loop-specific details of the protocol.

Generally, any kind of end device adhering to the FCP protocol can be attached via the zSeries FCP channel. However, inconsistent Fibre Channel implementations by device manufacturers can make it necessary to verify that no interoperability problems exist.

In the following sections, the emphasis is on switched Fibre Channel fabrics, since they represent the most important building blocks of a SAN.

Addressing

This subsection gives an overview of the different types of addresses used in a Fibre Channel network.

Worldwide names

Nodes and ports are referenced by a worldwide unique node name (WWNN) and a worldwide unique port name (WWPN), respectively. These WWNNs and WWPNs, both 8 bytes in length, are assigned by the manufacturer of a device or Fibre Channel adapter card (**Figure 3**).

Fibre Channel IDs

During the initialization of a link between an end node and a switch, WWNNs and WWPNs are exchanged, and the switch defines a 3-byte identifier (ID) for the port in the end node that is unique within the particular fabric. In all further communication, this ID is the only address qualifier used by the Fibre Channel transport mechanism (FC-2 layer) to route frames from their source to their

destination. The Fibre Channel frame headers carry a source ID (S_ID) identifying the sender of the message and a destination ID (D_ID) addressing the port that is to receive this message. All other address qualifiers are defined by higher-level protocols, such as FICON or FCP.

Normally, these 3-byte Fibre Channel IDs are created using the following scheme:

- The high-order byte specifies the switch to which a source or destination end port is attached.
- The mid-order byte specifies the port on this switch to which the source or destination end port is attached.
- The low-order byte is used to address up to 126 devices if a public loop is attached to this switch port.

The concept of assigning and using 3-byte IDs instead of 8-byte WWPNs after a connection has been established enables the use of simple hardware-based routing algorithms.

If two switches are connected in different SANs, effectively combining the two SANs into one, all switches in this combined new SAN check for duplicate IDs and begin an ID reassignment procedure if required.

A switch or port cannot be individually addressed as a physical Fibre Channel entity. However, switches provide access to certain services such as name (or address) resolution services. These services can be accessed by using so-called *well-known IDs*, which are in the 0xFFxxxx ID range.

FCP addressing

If a Fibre-Channel-to-SCSI bridge attaches parallel SCSI devices that support SCSI logical unit numbers (LUNs), these LUNs are mapped by the bridge to FCP LUNs. The full address of an FCP device consists of a destination ID (D_ID), assigned by the switch at fabric log-in, and the FCP LUN. The D_ID is used to select the target device port, comparable to a parallel SCSI ID, and the LUN selects the logical unit within the target device, for example a specific logical disk volume.

4. FCP attachment in a zSeries environment

This section describes the specific requirements for FCP attachments on a zSeries server system and how the zSeries FCP channel meets these requirements.

Server consolidation

A key asset of the zSeries is its ability to concurrently execute multiple operating systems in up to 15 logical partitions (LPARs). Also, the z/VM operating system, with its integrated hypervisor function, supports multiple guest operating systems. The combination of the two allows a large number of operating systems to execute at the same time on this very reliable platform, limited only by the available system resources. This makes the zSeries a perfect choice for consolidating a large number of Linux servers on a single zSeries system.

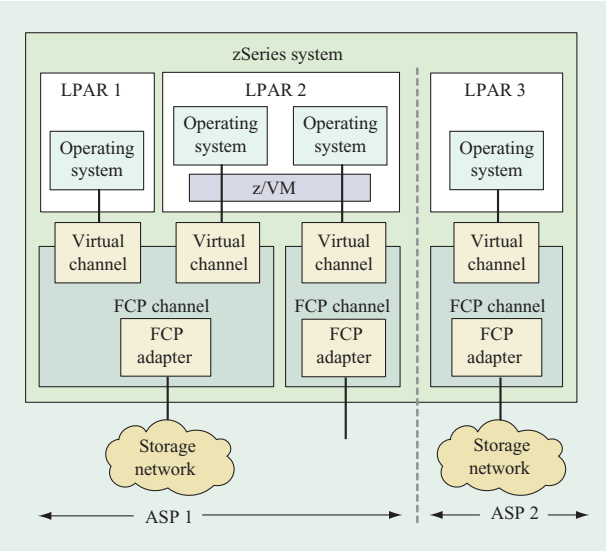

Figure 4

Virtualization of zSeries I/O channel.

This consolidation on a single system has many advantages for a user. It considerably reduces floor space and cabling requirements. The number of outages and repair actions is reduced. And, perhaps most important, the cost for administration, management, and maintenance decreases, considerably reducing the total cost of ownership (TCO) as compared to a server farm, in which the Linux systems run on a large number of separate servers.

Also, in a server farm environment each instance of Linux requires its own Fibre Channel adapter or, commonly, two of them for reliability reasons. The zSeries, on the other hand, allows FCP channels to be shared among multiple operating systems, reducing the overall Fibre Channel attachment cost.

There may be cases in which the sharing of FCP channels should not be allowed. **Figure 4** depicts an example of a zSeries system partitioned into several LPARs. Each LPAR or set of LPARs, together with one or more associated FCP channels, can be assigned, for example, to an application service provider (ASP), who may use it, for example, for Web services, application services, or PC hosting. In such an environment, different service providers must have restricted access to the same zSeries FCP channel. This can be ensured by using standard zSeries system configuration mechanisms, which are generally available for all zSeries channel types. On

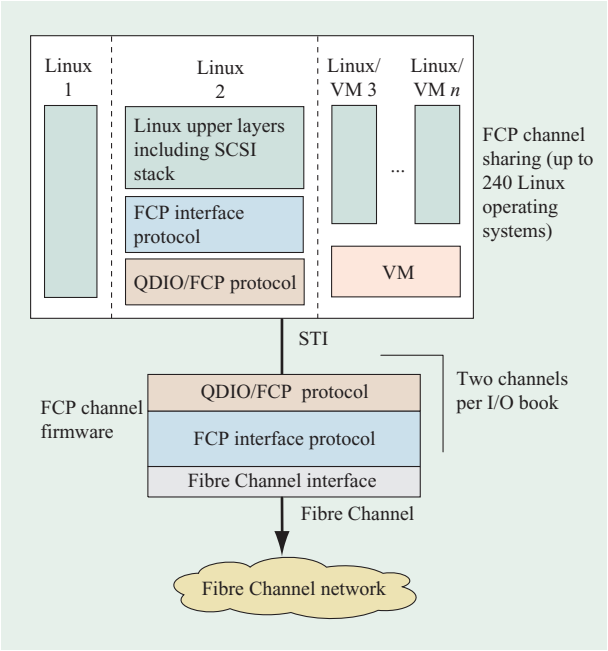

Figure 5

FCP software and firmware overview.

the other hand, each service provider can decide to share one or more of its FCP channels and assign virtual FCP interfaces to LPAR or z/VM guest operating systems as appropriate.

FCP channel sharing

To the operating system, an FCP channel is defined as a new channel type, known as an *FCP channel*. One adapter (with a single Fibre Channel port) is configured as one such FCP channel.

Up to 240 z/Architecture-defined subchannels are available on each physical FCP channel as queued direct I/O (QDIO) devices, as described in Section 5. These devices are internal software constructs and have no relation to physical devices outside the adapter. The host operating systems use these subchannels as vehicles to establish conduits to the FCP environment. Each subchannel represents a virtual FCP adapter that can be assigned to an operating system running either natively in a logical partition or as a guest operating system under VM.

Different host operating systems sharing access to a single FCP channel may access the same Fibre Channel port via this channel. However, only a single host operating system is allowed to establish access to a particular Fibre Channel device (identified by its LUN) at any given time. This restriction has been introduced to eliminate possible conflicts that may occur due to the fact that this FCP channel appears to the Fibre Channel network as a single N_Port, with a single WWPN and port ID, although it is shared by multiple host operating systems.[2]

Since up to 240 virtual adapters can be defined per FCP channel, a connection into a Fibre Channel network can be provided to a fairly large number of operating systems, even with a single FCP channel. And the fact that up to 96 such Fibre Channel interfaces can be installed on a single zSeries 900 (up to 32 on zSeries 800) offers rich configuration capabilities.

Virtualizing a Fibre Channel host bus adapter (HBA) is a unique feature of the zSeries. The zSeries FCP solution requires no changes in endpoint devices such as disk or tape controllers, and it offers the protection and security characteristics required to support server consolidation scenarios as described above.

5. The link between channel and software

The zSeries FCP channel concept

The zSeries FCP channel is based on the common I/O platform, as described in [10]. This I/O platform provides two zSeries FCP channels, each employing a Fibre Channel chip set, a PowerPC processor and memory, and associated firmware. The STI interface (as described in [11]) provides the physical connection between such a channel and the zSeries central electronic complex (CEC), i.e., the complex consisting of zSeries processors, main memory, and I/O subsystem. For more details about the FCP channel implementation, refer to Section 7.

On top of this physical connection between the CEC and the FCP channel, a communications protocol is required to exchange requests, responses, and data between software running on the zSeries processor and the firmware in the FCP channel; this communications protocol has two layers. The lower layer is the QDIO/FCP protocol, which provides a communication path between the two communicating entities. The upper layer is the FCP interface protocol. This is a command/response protocol used for activities such as establishing communication paths to Fibre Channel ports and devices, sending Fibre Channel commands to these entities, and similar functions. **Figure 5** shows both the host operating system and the FCP channel firmware containing specific layers for handling these protocols. It also depicts the layer on the channel that provides support to deal with multiple counterparts in different host operating system images, in order to support the virtualization concept

[2] Proposals for Fibre Channel architecture enhancements exist which would allow a single physical N_Port to behave as a set of logical N_Ports, each with its own unique WWPN and Fibre Channel ID. This may permit this restriction to be dropped in the future.

described in Section 4. These two protocols are described in more detail in the following subsections.

QDIO/FCP protocol

The queued direct I/O (QDIO) protocol was initially introduced as an extension of z/Architecture for high-speed zSeries communications adapters, such as the OSA-Express Gigabit Ethernet adapter. It largely bypasses the zSeries channel subsystem and lower I/O layers of operating systems that typically deal with classical zSeries I/O constructs such as channel programs for the initiation or completion of requests. This scheme provides considerable performance improvement compared with using I/O constructs based on classical zSeries I/O architecture.

With QDIO/FCP, traditional z/Architecture CCWs are used only to initialize these data paths. Sense-ID CCWs are issued to verify that the accessed "devices" are in fact QDIO devices. These are followed by other CCWs to create and activate the QDIO queues. Once these queues are successfully established, the subchannel remains in the architectural state "subchannel and device active." All subsequent control and data traffic flows through the QDIO queue structures.

An overview of the QDIO protocol is given in [12]. However, since QDIO was initially defined for networking traffic, it did not fully meet all of the needs of storage attachments and had to be extended accordingly. The specific requirements originating from these types of protocols are described in the next subsection.

Characteristics of storage access protocols

The industry-standard protocols SCSI and FCP basically consist of requests, associated responses, and unsolicited status presentations. Requests consist of command blocks, describing the action to be executed on the device, optionally with output data, pointers to buffers for storing input data, or both. Responses consist of request-completion status indications specifying the results of command execution and optionally input data and/or sense data. The latter can provide additional information in case an error occurs while the command is being executed. Unsolicited status indications provide information which may be associated with a command, but in which the status is detected by the device after it has signaled command completion. More often, unsolicited status is not associated with a command at all, for example in the case of incoming Extended Link Services initiated by another Fibre Channel node.

QDIO protocol with extensions for FCP

QDIO is based on queues and associated control blocks in main memory. Host software accesses these queues and control blocks to issue I/O requests or to determine that an I/O operation has completed. The actual data transfer is done by the FCP channel, which can access these queues to determine the work that is to be done, perform the required data transfers, and store the pertinent completion status.

Subchannels as defined in the z/Architecture are used to address queues. For communications, both control subchannels and data subchannels are defined. Only the latter are used with the zSeries FCP channel. For the FCP channel, each subchannel is associated with a single pair of queues, a request queue and a response queue.

An operating system requires just one subchannel with the associated queue pair to communicate with an FCP channel and to access all of the Fibre Channel devices that can be reached via that FCP channel. As described above, up to 240 such QDIO subchannels per FCP channel are currently supported, and these may be assigned to different operating system instances so that they all can share the FCP channel.

The device address (as defined by the z/Architecture) used to address this single subchannel is associated only with this particular communication path between the host operating system image and the FCP adapter. It is not related in any way to the addresses of the devices and controllers in the Fibre Channel network. Rather, the addresses associated with these Fibre Channel target devices, like their WWPNs and LUNs, are conveyed by software to the FCP channel as part of the data it provides to the adapters via the QDIO queues.

Host software puts the data to be transferred to the FCP channel into data buffers in main memory, or allocates buffers for data to be retrieved from the FCP channel. Each QDIO queue has up to 128 entries, and therefore can specify up to 128 buffers. The buffers do not have to be contiguous but can consist of a sequence of 4KB entities. The addresses of these entities are maintained in storage buffer access lists (SBALs), where each entry in a list, called a storage buffer address list entry (SBALE), references a single 4KB buffer.

For a more detailed description of the QDIO protocol, refer to [12].

FCP interface protocol

The QDIO queues in the context of the FCP channel are used for three different purposes: to send requests from the host program to the FCP channel, to send responses to these requests from the FCP channel back to the host software, and to present unsolicited status information from the FCP channel to the host program.

Request handling

The host operating system uses a special control block known as a queue transfer control block (QTCB) to specify a request that should be handled by the FCP channel. A QTCB is used for FCP commands to be sent to the Fibre Channel network, as well as for requests to

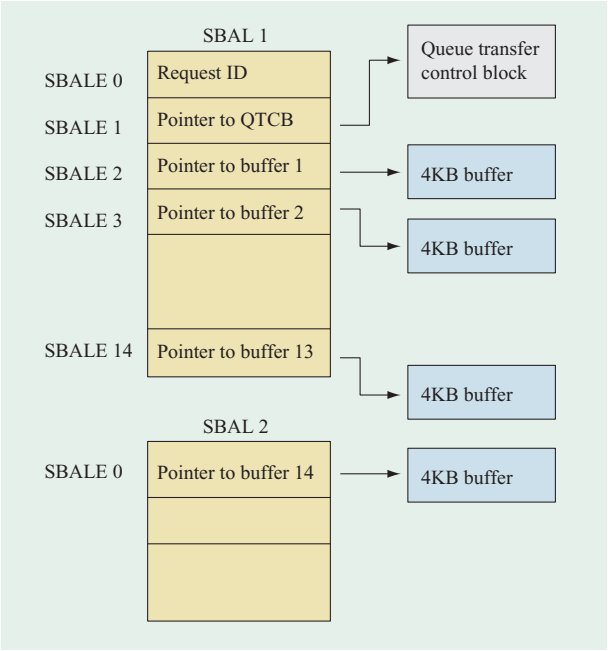

Figure 6

QDIO control structure.

be handled internally by the FCP adapter. These requests may further specify that data be written, read, or both.

The QTCB contains command code specifying the operation to be performed by the FCP channel. One specific command code, the most important one, is the request for the FCP channel to send an FCP command to a remote device on the Fibre Channel network. For this type of operation, the host program places the entire FCP command descriptor block, as defined by the FCP standard, into the QTCB. This FCP command descriptor block specifies both the addresses of the target FCP device (such as its D_ID and LUN) and the command to be executed.

Furthermore, the QTCB contains space for response data and status information to be filled in by the FCP channel when it processes the associated request.

Since each request issued in the context of the storage access protocol has a corresponding response, requests and responses must be correlated. This is done via a request identifier generated by the host program. The host program must ensure the uniqueness of this request identifier within its scope, i.e., for the particular subchannel it uses to communicate with the FCP channel. When this identifier is forwarded to the FCP channel as part of a request, the FCP channel qualifies the identifier with the number of the QDIO subchannel on which the request was received, and the logical partition (LPAR) identifier. To have a request processed by the FCP channel, the host program places both the request identifier and a pointer to the QTCB into the first two SBALEs of an SBAL (see **Figure 6**). If the request is associated with a data transfer, additional SBALEs are filled with pointers to the data in the case of write requests, or pointers to data buffers to be filled in the case of read requests.

With the QDIO/FCP protocol, up to 15 SBALEs per SBAL are used. Since the first two SBALEs are used for the request identifier and the pointer to the QTCB, 13 entries are left for data pointers. If this is not sufficient, up to three additional SBALs can be linked to the first one. In these linked SBALs, all 15 SBALEs can be filled with data buffer pointers. The maximum amount of data that can be transferred in this way between main memory and the FCP channel with a single request is more than 2 MB.

The FCP channel firmware processes the QTCB, "unwraps" the included FCP request, and processes the command (see **Figure 7**). In many cases, this requires the FCP channel to send an I/O request to an external device, such as a disk controller, or a device that is part of the Fibre Channel infrastructure (for instance, a Fibre Channel switch). Other commands may be executed completely within the FCP channel, without requiring an external I/O operation. Also, some commands require the FCP channel to transfer data to an external I/O device or receive data from such a device.

Response handling

If the FCP channel has processed a request, which may include retrieving data from zSeries main memory and sending it to the Fibre Channel network, or receiving data from the network and uploading it into zSeries main memory, it also updates the QTCB with the corresponding completion indication and potentially additional status information. It then transfers the QTCB back to the same main-memory location from which it was originally fetched (see Figure 7). Finally, the FCP channel places the corresponding request identifier into an SBAL of the response queue and uses basic QDIO signaling mechanisms to alert the host program to an update to this queue. The host program uses this request identifier to correlate the response to the original request and retrieve the completion status from the associated QTCB again.

It is important to note that if the FCP channel has to perform an asynchronous I/O or data transfer operation in order to execute the request, the execution of such a request will always appear to be asynchronous from the point of view of the host program.

Handling of unsolicited status information

It is the responsibility of the host program to supply status-read request buffers via status-read requests. These are similar to read requests, except that they do not

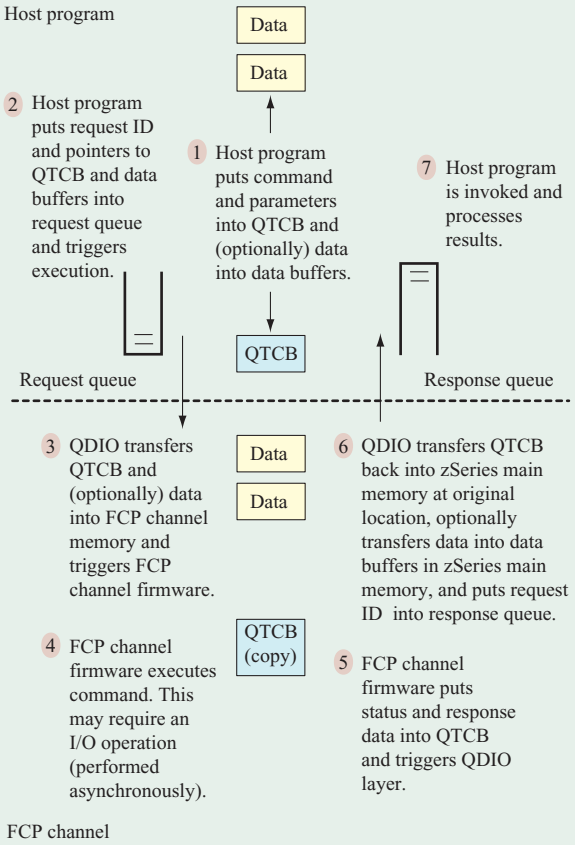

Figure 7

High-level control and dataflow.

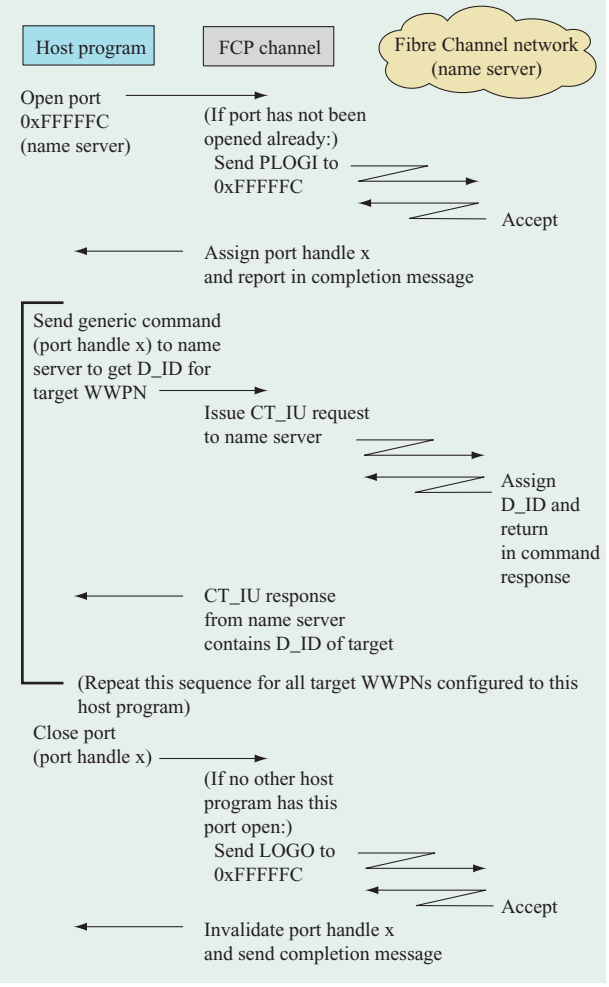

Figure 8

Name server accesses to determine D_IDs of target ports.

trigger a read operation. Instead, they are left pending until unsolicited status arrives. When the FCP channel receives unsolicited status information, this status is put into one of the pending buffers, and the associated request identifier is posted in the response queue.

6. A command flow scenario

This section presents a sample scenario for the cooperation of a host program and the FCP channel in the execution of I/O commands.

The first action that must occur before any other communication with the Fibre Channel network can occur is a fabric log-in (FLOGI), which is done by sending a FLOGI command to the Fibre Channel switch to which the FCP channel is directly connected. This fabric log-in is done by the FCP channel when it initializes, as soon as it is able to activate the Fibre Channel link to this switch. No host program interaction is required for this purpose.

When the FCP channel initialization is complete, a host program intending to use this channel must establish a QDIO connection to the channel, as described in Section 5, before it can begin communicating with the Fibre Channel network. The host program must have been configured with the addresses of the Fibre Channel devices it wishes to access. The configuration of WWPNs and LUNs for these devices and the mapping to standard Linux device addressing constructs are described in more detail in Section 9. To communicate with a device, the host program must determine the Fibre Channel destination identifiers (D_IDs) corresponding to the configured WWPNs. This is done by accessing the name server of the Fibre Channel fabric, as shown in **Figure 8**. The host program must first send an Open_Port request to the FCP channel, specifying the well-known destination ID 0xFFFFFC of the name server. The FCP channel then establishes a connection to the name server by performing

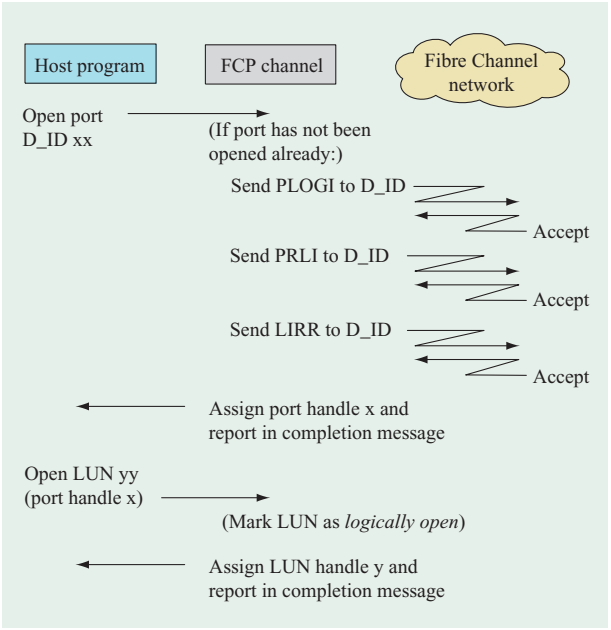

Figure 9

Open target port and LUN.

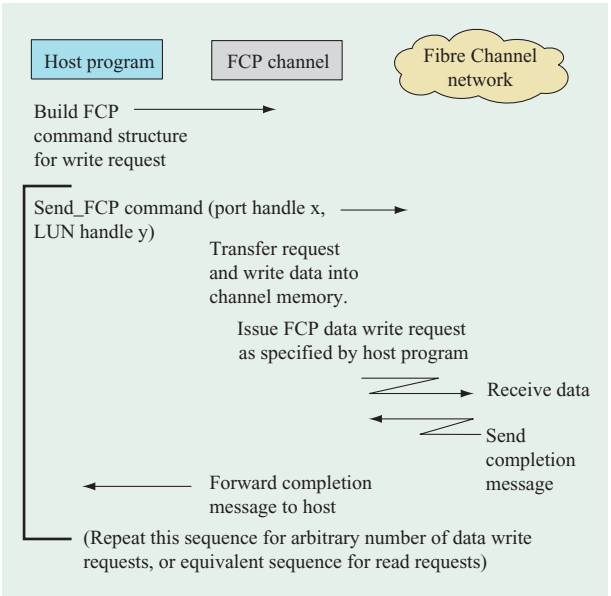

Figure 10

Issue write (and read) requests.

a port log-in (PLOGI) to that D_ID. If this completes successfully, it assigns a port handle x to the logical connection to this port and returns this handle in the completion message to the host program. The host program must specify this port handle in all subsequent requests to this target port.

Next, the host program issues the command Send_Generic to the FCP channel in order to send so-called Fibre Channel generic requests (of type CT_IU) to the name server to interrogate the D_IDs of all of the target ports (identified by their WWPNs) it wishes to access. When it has finished, it finally sends a Close_Port request for this name server port (identified by its port handle) to the FCP channel. This causes the FCP channel to send a port log-out (LOGO) request to the name server port 0xFFFFFC. When this request has completed, the FCP channel invalidates the port handle x again and returns a completion indication to the host program.

The host program now recognizes all of the D_IDs of the target ports it wishes to access and is ready to set up connections to those ports and the attached devices. A connection to a target port is established by sending an Open_Port request to the FCP channel, specifying the D_ID of that port (see **Figure 9**). In the same manner as for an Open_Port to the name server, the FCP channel again sends a PLOGI request to that port. However, upon successful completion of this request, it also sends a process log-in (PRLI) and, finally, a link incident report registration (LIRR) to this port. These two latter requests are suppressed for log-ins to ports with well-known Fibre Channel addresses starting with 0xFF . . . (for instance, the address of the name server). When all of these sequences have completed successfully, the FCP channel assigns a port handle x to this connection and returns it with the completion message to the host program.

If a second host program now issues an Open_Port request to this same port, the PLOGI, PRLI, and LIRR requests are not sent. Instead, the FCP channel just opens the port logically and assigns a different port handle x1 to this connection. Both host programs now have access to the same physical Fibre Channel port, each of them using its own unique port handle.

After the host program has created a connection to a Fibre Channel port, it can establish connections to all of the LUNs accessible via this port that have been configured to this host program. This is done by sending an Open_LUN request for a particular LUN, specifying the port handle of the associated Fibre Channel port and the 8-byte Fibre Channel LUN of the device. Opening a LUN does not cause any requests to be sent to the Fibre Channel network; rather, it is handled internally by the FCP channel. It assigns a LUN handle y to this connection to the device and returns it to the host program.

The host program is now able to send read and write requests to such a device, as shown in **Figure 10** for the case of a write. The host program builds the FCP request

to write data to its own host memory and issues a Send_FCP command to the FCP channel to trigger the execution of this command. Port handle x and LUN handle y are used to identify the target device for this operation. The FCP channel now transfers both the request and the data to be written from host memory into FCP-channel internal memory, and sends the request to the Fibre Channel network. When the operation is complete, a completion indication is returned to the host program, together with any sense data that might have been received for this request.

It must be noted that the host program may have any number of concurrent requests to the same or different devices outstanding at any given time, limited only by the amount of resources available in the FCP channel.

When the host program is finished with a certain device, it can issue the Close_LUN command for that device, which causes the FCP channel to invalidate the LUN handle y that had been assigned to this LUN (see **Figure 11**). No requests are sent to the Fibre Channel network in this situation.

Finally, when the host program has closed all LUNs that are accessed via a particular target port and wishes to close that target port, it issues a Close_Port command, specifying the pertinent port handle x. The FCP channel now determines whether any other host program still has an open connection to this port. If this is the case, it just invalidates port handle x and sends a completion indication back to the host program. Otherwise, if this was the last or only host program to talk to this port, it first sends a port log-out (LOGO) request to the Fibre Channel port before returning the completion indication to the host program.

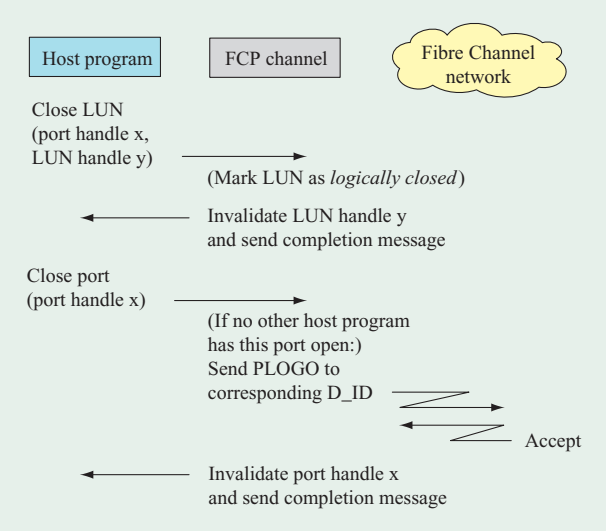

Figure 11

Close LUN and target port.

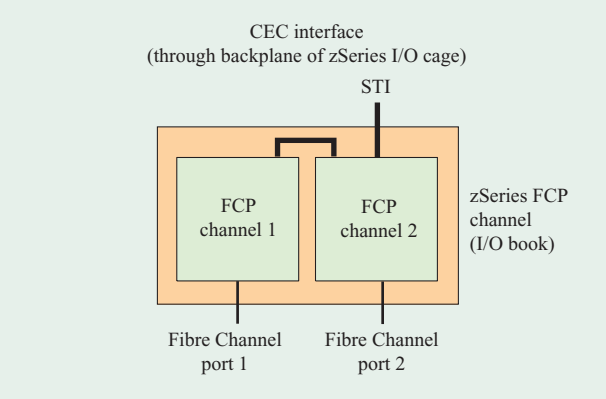

Figure 12

Hosting of two FCP channels by the zSeries I/O book.

7. The zSeries FCP channel

This section presents a brief overview of the FCP channel hardware and firmware concept.

The I/O attachments supported on the zSeries can be divided into two categories.

The first category contains direct attachments based on IBM-created hardware and attachment protocols, developed to meet the stringent reliability and data integrity requirements of zSeries-class systems. Examples are parallel channels, ESCON channels, and coupling links used to build zSeries Parallel Sysplexes*.

The second category comprises those attachments which provide a vehicle for including industry-standard I/O adapters, in particular Peripheral Component Interface (PCI) adapters, either in the form of PCI daughter cards or through inclusion of their chip sets on the zSeries channel card. Examples are the zSeries Crypto, Gigabit Ethernet, and FICON cards.

In other system configurations, PCI adapters have direct memory access to the system memory. Since the PCI bus and PCI adapters generally do not meet the high security and data integrity requirements of the zSeries, such a direct access approach from the PCI adapter and bus into zSeries memory has not been implemented. Rather, zSeries channels always use a store-and-forward data transfer scheme to or from the PCI adapters.

The zSeries FCP channel is of the second category.

Hardware overview

The zSeries FCP channel hardware is based on a zSeries I/O book package (see **Figure 12**). This package contains two completely independent channels, each providing a single Fibre Channel port, and a dedicated interface to

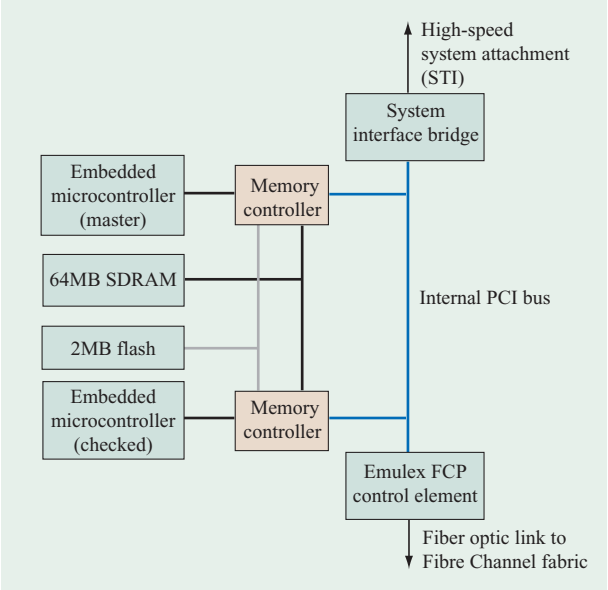

Figure 13

Block diagram of FCP channel hardware.

the central electronic complex (CEC). The card has the same form factor as other regular zSeries I/O cards and also uses the same CEC interface, referred to as the self-timed interface (STI).

The hardware of the zSeries FCP channel is identical to that of the FICON or FICON Express feature. It is the unique firmware that causes this hardware to execute in FICON or FCP mode. It supports both short-wave and long-wave optical Fibre Channel interfaces, and transmission speeds of 1 Gb/s and 2 Gb/s. A simplified block diagram of the zSeries FCP channel hardware feature is shown in **Figure 13**.

The STI provides all signals required for data transfer and control of the channel. Its narrow yet high-speed design reduces the I/O pin count required, thus permitting a large number of adapters to be attached to the system. The STI attaches directly to the system interface bridge on one of the channels and is then fed through to the other channel in the same I/O book package.

The system interface bridge chip is fully checked internally and performs a number of roles. It hosts the DMA engines of the channel. All customer data transferred to and from the channel must pass through the 64MB SDRAM local data storage (LDS), which is a staging area for customer data and hosts several buffer pools of differing sizes. This permits the greatest throughput by matching the sizes of the buffers used to the sizes of the transfer requests. Quite often it is necessary for the channel to communicate with other system elements through small control structures. For this reason, the system interface bridge contains facilities for the processor to fetch and store small memory blocks (in multiples of 128 bytes) directly from/to host memory, bypassing the DMA facility and the associated interrupt handling.

The system interface bridge chip also contains the PCI bus arbitration logic and memory access control. Numerous addressing boundary checkers are built in to increase the data integrity of the channel.

The system interface bridge DMA engine is managed through the use of a Data Mover Queue (DMQ). This construct is the key to efficient pipelining of data through the adapter. A circular queue specific to the system interface bridge is constructed by code in local processor storage (LPS). The queue elements are composed of control structures establishing ownership (system interface bridge vs. firmware), memory access key information, operation completion status, and, most significantly, scatter/gather lists for adapter memory and main memory. The DMA operations are bidirectional and single-threaded, offering FIFO scheduling for execution. For more details on the system interface bridge chip and the STI interface, see [11] and [10].

The processor complex of the adapter consists of a pair of PowerPC* 740 "Lonestar" processors and their associated memory controller chips. The pair run in parallel to exploit the cross-checking capability of the memory controller chips. This configuration enhances the data integrity of the channel. The memory controller also provides memory access protection and processor cache control.

LPS is a 64MB SDRAM divided into several functional areas including control program memory, program heap, and stack space. Access to these areas is granted only to the processor. Other memory areas, which are accessible to both the processor and the Fibre Channel link adapter, contain control structures for queueing work to the Fibre Channel link.

The channel also contains a 2MB flash memory, which contains the basic boot code deemed necessary for preservation across power and reset cycles, to allow initialization of all channel hardware facilities. This flash memory also contains code critical to recovery from hardware- and software-detected errors.

The channel interface to the Fibre Channel fabric is via an Emulex chip set on the FICON Express feature, which provides the functionality of an Emulex LP9002 adapter.[3] The chip set includes an ASIC, an ARM processor, buffer

[3] "LightPulse LP9002 Data Sheet," Emulex Corporation, Costa Mesa, CA; see *www.emulex.com*.

and program memory space, and flash memory from which the firmware is loaded at initialization. The Emulex chip set shares LPS memory for control structures and has access to LDS memory for customer data.

All mainline Fibre Channel data transfer is sequenced on the link by the Emulex chip set. Basic recovery sequences can be performed by this chip set without intervention from the channel firmware. In addition, a large subset of SAN maintenance extended link services (ELSs) are also handled by the Emulex chip set. All ELSs that must be processed by a higher-level protocol layer are passed up to the channel firmware.

The channel firmware is capable of updating the flash memory of the Emulex chip set at adapter initialization time, which significantly enhances the serviceability of the product.

8. RAS

The zSeries FICON and FICON Express features operating in FCP mode adhere to the same reliability, availability, and serviceability (RAS) concept defined for other zSeries channels. This allows, for instance, the diagnosis of channel problems and repair (i.e., replacement) of a channel when necessary without affecting other parts of the zSeries system which are not currently using this channel. In the same way, when the FCP I/O capacity or connectivity of a zSeries system is increased, additional FICON or FICON Express feature cards can be installed and activated in FCP mode while zSeries operations are in progress on other parts of the system, including other zSeries FCP I/O paths. In addition, the firmware of the FCP channel can be replaced at the same time, should that be required to resolve any problem found in this code, or in some cases even to add new functions to the FCP channel.

For error handling, the sophisticated error reporting, logging, recovery, and diagnostic support functions as defined for the zSeries FICON channels are also available for the zSeries FCP channel.

Error reporting is done in several ways. General problems with the FCP channel, especially channel hardware problems, are logged to the zSeries support element (SE), as is done for FICON channels. For other types of errors, the associated error information may be logged on the SE, by the Linux operating system, or both, depending on the type of error. For instance, all Fibre Channel layer 2 problems detected by the FCP channel are reported to the SE for logging. Additionally, special traces can be set up by the operating system to trace the execution of any FCP command, if required, for debugging and error diagnosis.

Another capability required of the zSeries FCP channel is the handling of unsolicited indications from the Fibre Channel network. A node in a Fibre Channel network may send unsolicited information, using the ELS mechanism, to report and assist in isolating and diagnosing potential problems in the Fibre Channel network. An example of this type of notification is the registered link incident report (RLIR), which provides error information related to Fibre Channel link problems. As the name suggests, participants in the Fibre Channel network can register for these types of notifications. The zSeries FCP channel performs such a registration and reports this information to both the SE and the operating system.

Another type of unsolicited information that may arrive as incoming ELSs is registered state change notifications (RSCNs). They notify a registrant, for example, about any log-in state change of a device, which may be an indication that a device has been added to or removed from the network or has failed. The FCP channel also does a registration for these types of ELSs. When such a notification arrives, the FCP channel generates the corresponding ELS response and also forwards the incoming ELSs as an unsolicited status notification to the host program. The host program will log this event and potentially initiate an appropriate recovery action.

9. FCP support in Linux for zSeries

Background

Since the development of Linux by Linus Torvalds in 1991, it has been significantly extended and improved by a huge number of developers from all over the world, making it an industrial-strength operating system for running anything from appliances to mainframe computers. Initially developed for x86 Intel PCs only, Linux has since been ported to a large number of hardware platforms with very different attributes, ranging from embedded controllers to IBM ESA/390- and z/Architecture-based servers.

The Linux porting project

When Linux was ported to the ESA/390 architecture in 1999, it was necessary to adapt to a completely different I/O scheme which employed the ESA/390 channel architecture. Because of the I/O abstraction scheme the Linux kernel has implemented in its platform-independent code layers, it was possible to preserve the concepts of devices and interrupt request lines (IRQs) on which Linux is based and easily map them to the ESA/390 I/O addressing schemes, with an ESA/390 I/O subchannel mapping to the logical construct of an IRQ.

SCSI and FCP support

In addition to SCSI, servers today take advantage of the bandwidth and connectivity of Fibre Channel attachments

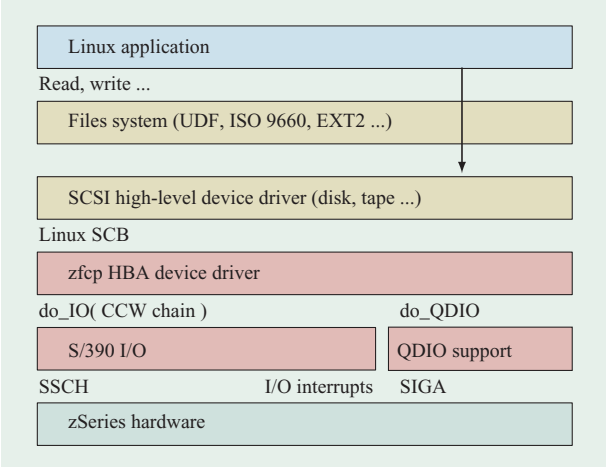

Figure 14

Linux SCSI/FCP control flow.

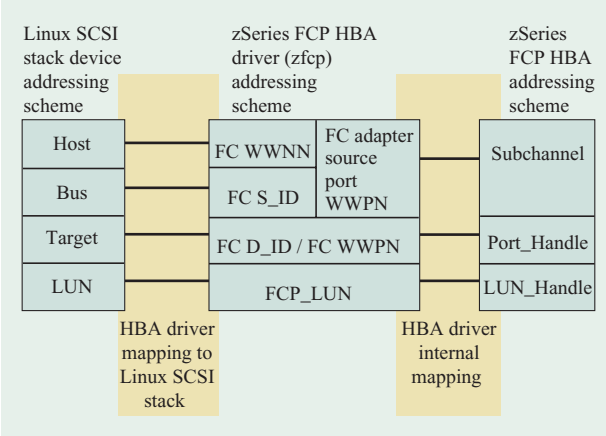

Figure 15

SCSI address mapping in Linux for zSeries.

to implement FCP protocols in FC-AL or switched Fibre Channel fabrics.

Linux has provided SCSI support for years. However, FCP-based I/O addressing semantics have not yet been pervasively embraced. The Linux SCSI stack continues to address SCSI devices exclusively by the address quadruple *host*, *bus*, *target*, and *LUN*, and does not support the WWPN/LUN addressing scheme on which FCP is based. Instead, a Linux Fibre Channel HBA device driver is expected to provide an addressing translation scheme to the upper-level SCSI kernel layers and implement the required lower-level Fibre Channel

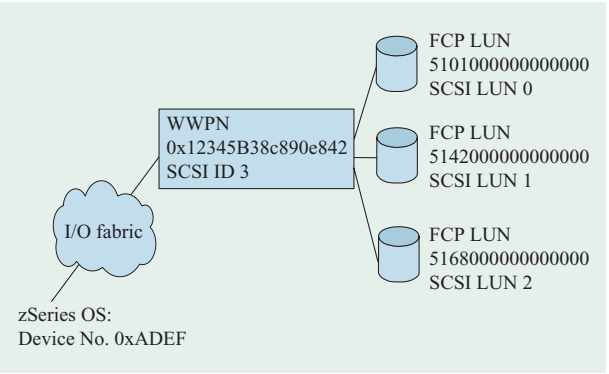

Figure 16

SCSI address mapping example.

layers as required, as well as any SCSI/FCP addressing translation itself.

The FCP support for Linux on zSeries is based on Linux SCSI support. This includes four high-level, platform-independent device drivers for SCSI disk, tape, and removable media, and a generic SCSI device.

zSeries specifics

The FCP support in Linux for zSeries utilizes the same QDIO infrastructure that is also exploited by Linux networking device drivers. Linux for zSeries has implemented a common I/O support layer for traditional CCW-based I/O semantics. Further, it provides support for QDIO semantics, which are employed in the zSeries by OSA-Express network adapters, HiperSockets internal LANs, and the FCP channels. This provides a very efficient way of exchanging I/O requests and moving data between an operating system and an I/O adapter (see Section 5 for details). The Linux SCSI/FCP control flow is shown in **Figure 14**.

Since Linux has not yet embraced Fibre Channel support in its kernel infrastructure, the FCP semantics must be translated into the SCSI addressing scheme used by the Linux SCSI stack. It is also necessary to implement new functionality in the device driver according to Fibre Channel standards, for instance to perform a port log-in (PLOGI). All of this is accomplished in the zfcp HBA device driver shown in **Figure 15**. However, only part of the work in the FC-2-related area is left to the Linux drivers, because some of it is already done by zSeries FCP channel firmware that provides virtualization and sharing, as described in previous chapters.

Figure 15 depicts the addressing translation as it appears within the zfcp HBA driver itself, applying to mid-level SCSI stack addressing semantics on the left side and attaching to FCP QDIO semantics on the right side.

In order to enforce SCSI-to-FCP address mapping, the zfcp HBA device driver further provides a configuration interface to accomplish this task.[4] The HBA device driver provides two different methods of defining such a mapping. One is a kernel or module parameter interface that activates the mapping when the driver is loaded. The other is a /proc file system interface that redefines address mappings at run time. A simple mapping is shown in **Figure 16**. Such a mapping is entered by the Linux operator using a configuration tool specific to the Linux distribution and stored in a distribution-specific way. A Linux script provided by such a Linux distribution may, for instance, convey these parameters to the /proc file system at startup time.

This mapping translates to the following zfcp HBA device driver mapping:

```
map="\
  0xADEF 0x03:0x012345B38c890e842 0x00:0x5101000000000000;\
  0xADEF 0x03:0x012345B38c890e842 0x01:0x5142000000000000;\
  0xADEF 0x03:0x012345B38c890e842 0x02:0x5168000000000000"
```

The first column identifies the zSeries device number, designating the logical FCP adapter instance. The second column maps the SCSI ID (3) to the WWPN of the target port. Linux uses this information to obtain the target D_ID from the name server if connected to a switch. Finally, the third column maps the different SCSI LUNs (0 to 2) to 8-byte FCP LUNs. All numbers have been specified in hexadecimal notation. Mappings not resolvable during system startup can be retriggered.

Using this simple approach, the zfcp HBA device driver takes advantage of the existing Linux SCSI subsystem and limits new Fibre-Channel-specific configuration duties to apply to the device driver only. Any adaptation to Fibre Channel semantics is encapsulated within the device driver itself. Therefore, from the SCSI subsystem perspective, the zfcp HBA device driver appears to be just another HBA device driver. Only the SCSI ID and LUN name space appear to be larger than usually found with parallel SCSI attachments.

10. Summary

The zSeries FCP channel support opens new possibilities for zSeries and Linux for zSeries with respect to industry-standard Fibre Channel and SCSI controllers and devices, augmenting the traditional zSeries peripheral I/O attachments. It extends the virtualization approach to industry-standard devices and gives it unique value in the UNIX market environment, where HBAs are typically dedicated, and also in UNIX server environments that support partitioning. Therefore, zSeries and Linux, along with industry-standard FCP and SCSI device attachments, turn out to be an excellent and synergistic match.

[4] There is currently no dynamic discovery in place, since it may not be feasible to provide a persistent SCSI configuration appearance to user space without Linux providing a common Fibre Channel infrastructure itself, which would eliminate the need for a SCSI mapping.

Acknowledgments

The authors wish to thank Allan Meritt and Juergen Maergner (both at IBM) and Michael O'Donnell (McData Corporation) for their many valuable comments regarding this paper. The number of contributors to the zSeries FCP has grown too large to list here, but we would like to express our appreciation to a few of the key developers: Otto Ruoss, Ralph Friedrich, Juergen Leopold, Markus Schmidt, Christian Rund, Khadija Souissi, Marco Kraemer, Karl-Friedrich Morlock, Eckehard Schulz, John Flanagan, Mark Bendyk, Rich LaFalce, Doug Lin, Martin Peschke, Aron Zeh, and Dr. Raimund Schroeder. Finally, our thanks go to Simone Schulz, who gave us the initial impetus to write this paper, and to John Marshall for the care he devoted to editing it.

References

1. ANSI/INCITS, Technical Committee T10, "Information Technology—SCSI Architecture Model (SAM)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 1995.
2. IBM Corporation, *z/Architecture Principles of Operation*, Order No. SA22-7832, 2001; available through IBM branch offices.
3. ANSI/INCITS, Technical Committee T10, "Information Technology—Serial Storage Architecture—SCSI-3 Protocol (SSA-S3P)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 1997.
4. ANSI/X3.269:1996, "Information Systems—Fibre Channel Protocol for SCSI (FCP)," American National Standards Institute, Washington, DC, 1996.
5. ANSI/INCITS, Technical Committee T10, "Information Technology—Fibre Channel Protocol for SCSI, Second Version (FCP-2)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 2001.
6. ANSI/INCITS, Technical Committee T11, "Fibre Channel Framing and Signaling (FC-FS)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 2002.
7. ANSI/INCITS, Technical Committee T11, "Fibre Channel Generic Services-3 (FC-GS-3)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 2000.
8. ANSI/INCITS, Technical Committee T10, "Information Technology—SCSI Architecture Model-2 (SAM-2)," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 2002.
9. ANSI/INCITS, Technical Committee T11, "Fibre Channel Single-Byte Command Code Sets-2 Mapping Protocol (FC-

SB-2), Rev. 2.0," American National Standards Institute and InterNational Committee for Information Technology Standards, Washington, DC, 2000.
10. D. J. Stigliani, Jr., T. E. Bubb, D. F. Casper, J. H. Chin, S. G. Glassen, J. M. Hoke, V. A. Minassian, J. H. Quick, and C. H. Whitehead, "IBM eServer z900 I/O Subsystem," *IBM J. Res. & Dev.* **46,** No. 4/5, 421–445 (2002, this issue).
11. J. M. Hoke, P. W. Bond, R. R. Livolsi, T. C. Lo, F. S. Pidala, and G. Steinbrueck, "Self-Timed Interface of the Input/Output Subsystem of the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 447–460 (2002, this issue).
12. M. E. Baskey, M. Eder, D. A. Elko, B. H. Ratcliff, and D. W. Schmidt, "zSeries Features for Optimized Sockets-Based Messaging: HiperSockets and OSA-Express," *IBM J. Res. & Dev.* **46,** No. 4/5, 475–485 (2002, this issue).

Received November 27, 2001; accepted for publication April 8, 2002

Ingo Adlung *IBM Server Group, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (adlung@de.ibm.com).* Mr. Adlung graduated from the University of Applied Sciences in Esslingen, Germany, with a degree in information technology. He joined IBM VSE/ESA development in 1990 and was involved in the development of Linux for zSeries from its very beginning, with responsibility for the I/O subsystem. Since 2000, he has been the Linux strategy and design leader for the IBM eServer zSeries platform.

Gerhard Banzhaf *IBM Server Group, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (banzhaf@de.ibm.com).* Dr. Banzhaf received a Diploma (M.S.) in computer science from the University of Karlsruhe (TH) and a Ph.D. in electrical engineering from the University of Siegen (GHS). He joined IBM at the Boeblingen Development Laboratory, where he is concerned primarily with the development of I/O subsystems and microcode.

Wolfgang Eckert *IBM Server Group, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (weckert@de.ibm.com).* Mr. Eckert received a Diploma (M.S.) in physics from the University of Hamburg. He is an IBM Distinguished Engineer and is now working on IBM eServer system design and hardware/software interfaces/architecture. He joined IBM in 1968.

George Kuch *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (gkuch@us.ibm.com).* Mr. Kuch holds an M.S. degree in computer engineering from Syracuse University. He joined the IBM channel development effort for mainframe systems in 1980 and has been involved in all aspects of channel engineering including design, simulation, microcode development, tool development, test, and project management. Mr. Kuch is now part of the zSeries FCP channel microcode development team.

Stefan Mueller *IBM Server Group, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (stefan@de.ibm.com).* Mr. Mueller studied computer science at the Berufsakademie Stuttgart, graduating in 1993. His studies were accompanied by practical training at IBM in Sindelfingen, Germany. In 1993 he joined the IBM Boeblingen Development Laboratory as an R&D engineer, participating in the development of zSeries internal disk subsystems. He now leads a development team for the zSeries FCP channel.

Christoph Raisch *IBM Server Group, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (raisch@de.ibm.com).* Mr. Raisch received a Diploma (M.S.) in electrical engineering from the University of Stuttgart. In 1998 he joined IBM at the Boeblingen Development Laboratory, where he now works on microcode and development of I/O subsystems for Linux.