4 Program Communication: PathLib

Chapter 1 provided a brief overview of program communication on the iWarp system. This chapter describes the PathLib library of communication functions used to implement program-based communication. Appendix B provides thorough descriptions of all PathLib functions. The section *PathLib Terms*, at the end of this chapter, provides definitions for PathLib terminology.

NOTE

> It is essential to have a clear understanding of the iWarp architecture to effectively design programs using the PathLib communication functions. For a complete description of the iWarp architecture, refer to the *iWarp Macroarchitecture Specification*.

Communication Overview

Each cell's communication agent maintains a hardware structure called the Pathway Control Table (PCT). Together, the communication agent and PCT provide the mechanisms necessary to support logical channels, the means by which cells communicate over the array.

The PCT contains a collection of PCT entries. The PCT entry is the fundamental resource comprising a logical channel. A logical channel is a chain of PCT entries that link a sender cell to a receiver cell. All cells in the path across the array from the sender cell to the receiver cell contribute PCT entries to support a logical channel, as do the sender and receiver cells.

As a logical channel is being created, it competes with other logical channels being created for the resources on the cells it passes through to reach its destination. Once resources are allocated to a logical channel, they carry data sent over this logical channel until the logical channel is removed by the deallocation of its PCT resources.

Once created, a logical channel receives a guaranteed share of the capacity of the pathways over which it flows. How much of the capacity any one channel receives depends on the number of channels competing for capacity on the same physical channel. No capacity is required if no data is moving over a logical channel.

The communication agent PCT contains twenty entries, providing hardware support for a maximum of twenty simultaneous logical channels per cell. These resources are shared between the runtime system and PathLib, so less than twenty are available to the PathLib user. The exact number depends on the runtime system services operating on the cells.

For more information about the iWarp processor architecture, refer to the *iWarp Macroarchitecture Specification*.

PathLib Overview

PathLib supports the following communication operations:

- The creation and termination of logical channels, called connections.
- User creation and termination of messages over logical channels, known as messages and marks.
- User management of streamgates.
- Spooled block data movement over logical channels, called spooling.
- The creation and termination of aggregate logical channels, called joining and splitting, and known in PathLib as express.

Connections

A connection is an instance of a logical channel created by PathLib. A connection is known only at its endpoints: one on the origin cell of the connection and one on the destination cell. Connections are created and destroyed via the production and consumption of connection headers and trailers. The PathLib connections functions support the creation and termination of connections.

PathLib includes the following connection functions:

pl_create_connection()	Creates a connection with the calling cell as its origin and the route and destination specified via the user-supplied arguments. The connection is created by the sending of a connection header encoded with routing and destination information.
pl_accept_connection()	Accepts a connection header on the calling cell. The use of this assumes that some cell sent a connection with this cell as its destination. Upon return from this function, the creation of the connection is complete and all resources to support it are established.
pl_destroy_connection()	Destroys a connection originating from the calling cell. The communication agent resources that comprise the connection are deallocated by the sending of a connection trailer.
pl_conclude_connection()	Concludes a connection on the calling cell. The connection trailer is consumed, thus freeing the last resources associated with the connection.

The index of the PCT entry for a connection is returned when a create or accept connection function is called. This value is the users handle on the connection. The user is expected to supply this value to identify connections to PathLib functions that perform operations on connections.

Messages

A message is a block of data transmitted over a connection. All data must be sent within the context of a message. It is up to the user to establish messages by sending and receiving message headers, and to end messages by sending and receiving message trailers.

PathLib includes the following message functions:

pl_produce_message_header()	Starts a message by sending a message header. The message header will follow the path of the connection it is sent upon until it is stopped by a cell along the connection, or until it reaches the ultimate destination of the connection or aggregate logical channel of which the connection is a constituent (refer to the description of express in Appendix B). A message header may be stopped or consumed by any cell along the connection upon which it travels. Cells along the connection may request that generic message headers be stopped, or a message header may be matched by a cell's address match cam, which always causes the header to stop.
pl_consume_message_header()	Allows the calling cell to receive a message header from a connection. This action can be performed by any cell along the path of the logical channel that stops the header.
pl_produce_message_trailer()	Allows the calling cell to end a message by sending a message trailer over a connection. Like a message header, a message trailer may be stopped and consumed by any cell along the path of a logical channel. If not stopped by any intermediate cell, the message trailer will flow to the ultimate destination of a logical channel.
pl_consume_message_trailer()	Ends a message on the calling cell by consuming the message trailer.

The consume functions are also provided in forms that cause the header or trailer to be sampled (copied), but not consumed. All of these functions are available in lower overhead forms that select the streamgate via the function called as opposed to the user specifying it as an argument. Refer to Appendix B for a complete description.

Message headers and trailers can be recognized by the communication agent (refer to the description of express in Appendix B). The semantics associated with this capability are up to the user. There are some guidelines that should be observed regarding the use of message headers and trailers, and these are discussed in the section *Using Messages*, later in this chapter.

User Data Movement

PathLib provides streamgate binding operations to support user-programmed send and receives and spooled block data movement. Single-word and double-word streaming, plus support for test of the iWarp condition codes, are provided via the iWarp C compiler intrinsics _sendi(), _sendf(), _sendd(), _receivei(), _receivef(), _received(), _cond(), and _condcall(). Refer to the *iWarp C Manual* for complete descriptions of these intrinsics.

Implicit send and receive operations via the iWarp compute and access instruction are generated through the optimizing iWarp C compiler.

Streaming

Streaming is the user-programmed word-by-word transfer of data over messages by using streamgates. A streamgate is the mechanism that allows the computation agent to access a connection. A streamgate is bound to a connection for input, output, or both. The send and receive operations may be performed to or from the connection by specifying the streamgate.

There are four streamgates: two for inbound operations and two for outbound operations. The streamgates are GATE0_IN, GATE0_OUT , GATE1_IN, and GATE1_OUT . Additionally, streamgates are used by the connections, messages, and marks functions to perform the data production and consumption necessary to implement these operations. The user is expected to supply a streamgate specifier to these functions to identify the gate to be used to perform the operation. For some of these functions, the streamgate binding will be changed as a side-effect of the requested operation.

PathLib includes the following streamgate functions:

pl_bind_input_gate()	Bind an input gate to an inbound connection.
pl_bind_output_gate()	Bind an output gate to an outbound connection.
pl_save_input_binding()	Save the current binding for an input gate in a user-supplied variable.
pl_save_output_binding()	Save the current binding for an output gate in a user supplied variable.
pl_restore_input_binding()	Restore an input gate binding from a user-supplied variable.
pl_restore_output_binding()	Restore an output gate binding from a user-supplied variable.
pl_unbind_input_pct_entry()	Unbind either or both input streamgates from an inbound connection.

Spooling

Spooling allows the iWarp hardware to perform send and receive operations to or from messages and local memory without programmed word-by-word copy operations. Spooling can be thought of as a DMA operation between local memory and logical channels.

PathLib provides a simple interface to the iWarp spooling mechanisms to allow the user to use spooling to send and receive blocks of data.

NOTE

Currently , only word-oriented spooling operations are supported. Double-word spooling is more efficient, but is more difficult to support due to the lack of agreed-upon conventions and protocols to handle the proper alignment of data in local memory . Double-word spooling support is planned for a future release.

PathLib includes the following spool functions:

pl_spool_block_in()	Request the reception of a block of data into local memory from a connection via a spoolgate.
pl_spool_block_out()	Request the production of a block of data from local memory to a logical channel via a spoolgate.
pl_spool_complete()	Determine if a requested spool transfer is complete.

Aggregate Logical Channels (Express Operations)

The user may form aggregate logical channels by plugging connections together to form chains. The connecting cells may selectively participate in communications over the chain. Cells may join and split the connections as necessary to suit the requirements of the application.

When a cell joins connections, data is forwarded from the inbound neighbor to the outbound neighbor. Forwarding is handled by the communication agent, allowing the computation agent to perform other work. Forwarding is stopped by a user-specified sentinel or as a result of the arrival of a matching message header. The various sentinels are message header or trailer, software mark, connection trailer , or when the cell is between two messages on the channel.

PathLib express functions create and destroy aggregate logical channels by providing operations to set the sentinel, forward data, and resume programmed data movement after forwarding has been stopped

PathLib includes the following express functions:

pl_set_sentinel()	Set the sentinel to stop forwarding.
pl_get_sentinel()	Obtain the sentinel in effect.
pl_express_no_mask_toq()	Request that data be forwarded, but do not ignore the sentinel if the first data word to be forwarded satisfies the sentinel.
pl_express_mask_toq()	Request that data be forwarded. Ignore the sentinel if the first data word to be forwarded satisfies the sentinel, and forward until the sentinel is satisfied again.
pl_cancel_requested_express()	Attempt to cancel a previously requested express operation.
pl_stopped()	Determine if forwarding has been stopped.
pl_between_messages()	Determine if the cell is between messages.
pl_synced_with_neighbor()	Determine if data production can be resumed over an outbound connection that was joined to an inbound connection using an express operation.

Software Marks

A software mark is a data word that can be detected by the communication agent. The semantics associated with software marks are defined by the user. An express sentinel can be specified that causes express forwarding to stop when a software mark arrives at a cell. The marks functions allow the user to send, receive and sample software marks.

PathLib includes the following marks functions:

pl_send_software_mark()	Send a software mark.
pl_receive_software_mark()	Receive a software mark.
pl_sample_software_mark()	Copy a software mark.

Support Functions

PathLib also provides functions to initialize an instance of the library linked to a user's program and handle and print errors reported by PathLib functions.

PathLib includes the following initialization functions:

PL_INIT()	Initializes the iWarp hardware for use by PathLib. The user specifies which of the PCT entries and spoolgates available on a cell may be used by the PathLib interface for this instance of the library.
pl_rpe_configure()	Establishes which of the PathLib PCT entries in the calling cells neighbors are to support inbound/express logical channels propagated to them by this cell, and which local PCT entries this cell can use for creating connections.
pl_load_amc()	Load an address match cam and cam mask on this cell.

For a description of the error handling operations provided by PathLib, refer to the description of the plerror functions in Appendix B.

Guidelines for Using PathLib

Because of the flexibility required to use PathLib for program communication on the iWarp array, there are few strictly-enforced policies. While this provides the needed flexibility , it also creates hazards for the programmer.

The following guidelines will help avoid some basic pitfalls involved in designing programs for the iWarp array using PathLib communication.

Cell Naming

Cell naming and addressing are based upon the hardware address-matchcam. Each cell has four matchcams that together control what connection and message headers the cell recognizes.

The user has access to a portion of one cell matchcam. The PathLib function pl_load_amc() provides the user control over the low-order 19 bits of the PathLib matchcam id and mask. Thus, the user has exclusive access to one half of the available match address space. The other half is reserved for the iWarp RTS.

No inter-cell synchronization is provided by PathLib, such as that required to ensure that a receiver's address-matchcam is loaded before a connection reaches the cell. This synchronization is the user's responsibility.

PathLib doesn't address any aspect of how cells are named. This is up to the runtime system and the user. The current RTS gives every cell a canonical name.

Using Connections

To reach its destination, a connection must be granted resources along the route of the logical channel. The propagation of the connection header will stall if it encounters a resource shortage at any cell along its route. If resources become available, the header will then proceed. Once a connection is established, traffic over the connection is guaranteed a share of the available physical bus capacity in all processors along the path of the connection.

The routing of connections is a result of information encoded in the connection header and configuration information installed on the cells. The user has responsibility for correct management of both of these details.

Before connections can be created, proper RPE configuration must be in place within the array. If not properly configured, cells may forward connection headers in unpredictable directions.

Connections may be created that traverse the array in any way the user sees fit. Some important aspects of routing to consider are:

- ensuring that routes are deadlock free
- minimizing the cost of routes in terms of logical channel resources
- maximizing the use of available physical pathway bus capacity of the array
- enhancing the re-usability of cell and array programs by characterizing, consolidating, and standardizing the entry and exit points of inter-processor communications

In particular, the issue of deadlock must be addressed when setting up connections among cells. Due to a lack of synchronization and pathway cleanup primitives, only static configurations of connections should be used.

Assuming static connection allocations, observing the following guidelines will help avoid deadlock situations when setting up connections:

- All cells must have names before connections are created. RTS installs a canonical name for every cell before any user codes may be loaded and run on the array. An alias may be established for a cell through the use of pl_load_amc().

NOTE

> Aliases should not be used for connection setup. They should only be used for messages, and the call to install the alias should be made before any connection setup code is run. There is no inherent synchronization available to the user to ensure that cells have their names in place before communication requiring these names occurs.

- All cells must have established RPE configurations before they begin connection setup. These configurations are established by calling pl_rpe_configure(). The configuration established must provide sufficient resources on all cells to allow all connection setup work to complete.

NOTE

Cells may not change their RPE configurations once they are established without restarting the system.

- Connection setup and data movement over connections must be handled in two distinct and separate steps. First, connection setup should be performed in its entirety. Second, data movement can be performed. Observing this method avoids deadlock that might occur in cases where cells create connections to cells from which they also accept connections.

  For example, if two cells created connections to each other, then started their respective sends before accepting their partner's connection, they would be deadlocked if there isn't sufficient PCT entry queue space between each other to hold the transmissions.

  The simple way to avoid problems of this kind is to program a cell to create all the connections for which it will be the origin, then accept all the connections for which it will be the destination. After both of these steps are complete, data movement may be performed.
- When using pl_accept_connection() when more than one connection is pending at the cell, the connection accepted is arbitrary The user must program any protocol semantics associated with connections. For example, this might involve accepting the connection and processing user-defined protocol messages that follow the header.
- Connections must be terminated using the same cells that created or accepted them.

Using Messages

The communication agent does not require that data transmitted over logical channels be encapsulated by messages. Message headers and trailers may occur arbitrarily , but there is a wide agreement that a more restrictive usage model be employed to allow modular development of array programs, such that independently-developed programs and systems can later be used together.

Observe the following guidelines when using messages:

- Messages should not be nested between any two cells on a logical channel.
- Data words and software marks must not be found outside of a message by any cell participating in a logical channel.
- Cells may produce, consume, and forward (express) message headers and trailers arbitrarily, but between any two cells on a logical channel, data should always be within the context of a message.

  For example, a cell other than the cell that accepted a message header may stop and consume the message trailer for a message. If this is done, the message is complete through the cell that consumes the trailer. If the cell that accepted the header is beyond the cell that consumes the message trailer along the path of a logical channel, the message is closed between the origin of the trailer and the cell that consumed it, but the message remains open from the the cell that intercepted the trailer to the cell that consumed the header . Similarly, if the origin of the trailer is beyond the origin of the header, a message is still open from the cell that was the origin of the header to the cell that is origin of the trailer.

Express

Observe the following guidelines when using express:

- A connection trailer should not be allowed to be forwarded through the cell that is the ultimate destination of the connection.
- A single connection may not be joined to more than one partner at a time.
- Joined connections must be split before streaming or spooling may be resumed over the connections. Downstream neighbor synchronization must be performed before streaming or spooling is resumed on the outbound partner after a split is detected.
- The job of terminating connections that have been members of aggregate logical channels is an array-level activity, not a cell-level activity. Connections are terminated using individual cells, the termination of connections can have array wide consequences, particularly in the case of terminating connections that have been used to form aggregate logical channels. A basic guideline to observe is that aggregate logical channels must be terminated by splitting them into their constituent connections and terminating each of these using the PathLib connections interface.

A PathLib Example

This section contains a simple PathLib example in which a single logical channel is produced and consumed by a cell. The channel is sent out in the Xright direction from the cell, and returns to the cell from the Xleft direction, after traveling around the path of cells that comprise the row of the array in which the cell resides.

The example assumes the following:

- all the cells in the array have names
- this cell's name is provided by a system variable called _cellid
- the systems software has configured cell logical channel resources and has initialized the PL module.

```
#include <pl.h>

int _cellid;

main()
{
  int o_chan;               /* logical channel descriptor */
  int nr_hdr_words;         /* number of corners to turn to
                               reach destination */
  int _cellid;              /* name of the destination cell */

  int i_chan;               /* logical channel descriptor */
  int header;               /* storage for input header */

  /* Create a logical channel to this in the X-right direction. */
  nr_hdr_words = 1;
```

```
    o_chan = pl_create_connection(GATE0, DIR_XR, &_cellid, nr_hdr_words);
    (void) pl_produce_message_header(GATE0, _cellid);

    /* Accept the logical channel just created. */
    i_chan = pl_accept_connection(GATE0, &header);
    (void) pl_consume_message_header(GATE0);

    /* Send and receive 1000 words over the channel */
    {
      int k;
      for (k=0; k<1000; k++) {
      _sendi(GATE0, k);
      k = _receivei(GATE0);
      }
    }

    /* Terminate the output channel */
    (void) pl_produce_message_trailer(GATE0);
    (void) pl_destroy_connection(o_chan, GATE0);

    /* Terminate the input channel */
    (void) pl_consume_message_trailer(GATE0);
    (void) pl_conclude_connection(i_chan, GATE0);

    exit(0);
}
```

This example provides a simple illustration of programming on the iWarp system using PathLib communication functions. Chapter 5 provides further examples of developing and executing programs on the iWarp system.

PathLib Terms

This section defines terms that relate to PathLib.

aggregate logical channel
: A logical channel that is composed of two or more connections, such that data from one connection is automatically forwarded to the other connection. This type of channel is formed via express operations.

connection
: An instance of a logical channel created by PathLib. Connections may be joined to form aggregate logical channels that are chains of connections. Cells may also split aggregate logical channels back into its constituent connections (refer to the description of express in Appendix B).

express
: 1) The state of a logical channel with respect to a cell such that the cell's computation agent does not participate in the movement of data via the channel;

: 2) To couple an input connection to the RPE of an output connection by putting the input connection's PCT entry into a mode where a cell's communication agent moves data from the input connection to the output connection automatically.

forward	To pass message components through a cell in express mode.
header	A list of control words that specify a route across the array , and that are marked and recognized by the communication agent as being the first component of either a connection or a message.
logical channel	An allocation of resources in the communication agent that forms a pathway through which data may flow from cell to cell. Logical channels always have at least one sender (the producer of the channel) and one receiver (the consumer of the channel).
match	The action performed by the communication agent when a connection or message header control word id field is detected by an address match cam. A match occurs when a matching header reaches the toq and causes transmission to stop, leaving the header at the toq to be sampled or consumed by the computation agent.
mesh	The physical inter-connection of iWarp cells that constitutes an iWarp processor array.
message	A capsule for data carried over a logical channel, the header and trailer of which are defined and recognized by the iWarp communication agent.
PCT	Pathway Control Table; the hardware structure within the iWarp communication agent that stores the cell states of logical channels, including the recipient PCT index, the local and recipient free queue space, and an 8 word data queue.
partner	An input or output connection that is joined to another connection as a result of an express operation.
PCT entry	A record in the PCT that maintains the state for a logical channel.
PCT index	The handle for a specific PCT entry and also the PathLib users handle for a connection; it is an integer defined and understood by the communication agent to designate a PCT entry.
RPE	Recipient PCT Entry: a PCT entry referenced from an adjacent cell that is a downstream component of a logical channel.
RPI	A Recipient PCT Entry index; similar to a PCT entry index but is a reference to an entry on an adjacent cell.
routing destination	A cell that matches a connection header and is instructed by the header to select a new physical pathway upon which to propagate the connection header .

spoolgate	Any of eight physical interfaces between the computation agent and the communication agent through which spool transfer operations may occur.
spool index	The PathLib programmers handle on a spoolgate.
spooling	The transmittal and reception of blocks of data to and from local memory concurrent to, and without direct aid from, a program running on the cell using spoolgates. Spooling allows the PathLib programmer to initiate the movement of data, then have the cell move data to or from a logical channel and local memory without further programmed assistance.
stopped	The state of a logical channel with respect to a cell such that the cell's communication agent is not automatically forwarding data over the channel; not express.
streamgate	Any of four physical interfaces between the computation agent and the communication agent through which the computation agent may send and receive data over logical channels under direct program control.
streaming	The programmed word-by-word transmittal and reception of data over streamgates using streamgates.
toq	The record in the PCT entry data buffer that contains the next word to be processed by the computation agent or scheduled for transmission by the communication agent. Usually, the toq is only relevant with respect to the processing of inbound or express PCT entries, corresponding to the local PCT entry for an inbound connection or express partner.
trailer	The last word of either a connection or a message.
ultimate destination	The cell that accepted a connection via pl_accept_connection(), or the last cell on an aggregate logical channel.