Smart Messages

Dmitri Kondratiev

dkondr@bigfoot.com

Table of Contents

This document describes Smart Messages (SM) - messaging framework built with standard QNX Neutrino messages with the goal to provide optimized for efficiency data transfer both for local and distributed communications. Other goals that SM framework address are attribute-based messaging, blackboard and asynchronous communication, message 'store and forward' mechanisms. SM uses powerful QNX messaging to argument traditional shared memory and other IPC and provide user with efficient and flexible synchronization and data communication mechanism to be used in uniform way no matter where communicating end-points exist: in separate threads of a single process, in different processes running on the same node, or in different threads running on different nodes on the net.

Goals

Smart Messages framework address the following goals:

- Provide universal naming space and addressing scheme both for local and distributed message senders and receivers or message end-points.
- Provide distributed registration / discovery mechanism based on message end-points attributes.
- Provide message 'store and forward' mechanism when this is requested by end-points.
- Provide efficient transfer of message data optimized for speed with smart usage of local and shared memory.
- Provide simple and robust synchronization mechanism for communicating end-points.

Basic mechanism

Smart Messages are built with native QNX message passing IPC provided in Neutrino kernel. Message passing is fundamental Neutrino IPC with all other forms of IPC (pipes, FIFOs, etc.) built over it.

Communicating end-points

In Neutrino addressable message end-point, called 'channel', is defined with a three element tuple (NodeID, ProcessID, ChannelD). This allows the following types of communication between senders and receivers of messages:

- Sender and receiver running in different threads of the same process.
- Sender and receiver running in different processes on the same node.
- Sender and receiver running in different processes on different nodes.

To start message exchange receiver must create one or more channels that sender will use to send messages. Neutrino natively support 'fan-in' messaging mode, when receiver may get messages from different senders on the same channel simultaneously.

Receiver registration and sender connections.

When receiver creates a channel, corresponding message end-point (NodeID, ProcessID, ChannelD) gets registered with local QNX Program Manager (PM). To start sending messages any sender needs to connect (ConnectAttach()) to the desired end-point. Again, many senders may connect and send messages to the same end-point.

Sending and receiving messages

To send messages senders use MsgSend() family of Neutrino messaging functions while receivers use MsgReceive() function set. In general case of sending messages inside a single process Neutrino kernel copies a message directly from the address space of one thread to another without intermediate buffering. Thus, for local case, the message-delivery performance approaches the memory bandwidth of the underlying hardware.

Neutrino also provides functions for multi-part transfers. With these functions both the senders and receivers can specify a vector table (IOV) describing message memory fragments that both senders and receivers can use. Also provided is a special case of multi-part message that is a simple single-part message with a pointer to a single message buffer. All these message functions can be used to pass pointers to data buffers between senders and receivers running in a single process without coping data itself.

Synchronization

Neutrino provides simple, clear and powerful mechanism to synchronize message senders and message receivers based on MsgSend(), MsgReceive(), and MsgReply() primitives.

- 1. Receiver gets blocked waiting for message with MsgReceive().
- 2. Sender sends message with MsgSend() and gets blocked waiting for reply from receiver.
- 3. Receiver gets message and gets unblocked returning from MsgReceive(). Sender remains blocked.
- 4. Receiver now can implement any appropriate strategy required by message processing protocol that was agreed on with sender. Also receiver has full control when exactly to unblock the sender. In some cases sender may stay blocked until receiver fully processes the message. In other cases it may be needed to unblock receiver as soon as possible. No matter what strategy receiver needs at this step Neutrino guaranties that sender will stay blocked and thus will not be able to alter the message until receiver fully process the message and explicitly unblocks receiver.
- 5. Receiver is done with message processing and unblocks sender with MsgReply() call.

Extending basic mechanism with Smart Messages

Smart Messages (SM) extend basic Neutrino messaging in many powerful ways while preserving Neutrino messaging flexibility through SM control interface.

SM end-points addressing

In addition to standard Neutrino addressing of message end-points based on three-element tuple (NodeID, ProcessID, ChannelD), SM provides 'wild card' addressing based on attributes associated with every end-point (Neutrino channel).

SM aware processes can register channel end-points together with associated set of attribute value pairs (AVP). As a result :

- SM Senders can query receivers according to the attributes they publish not only by their names.
- Sender can connect and send messages not only to some other very well known process end-point but to 'any' end-point that matches attributes requested by the sender during connection phase.
- Smooth versioning. New receivers with the same attributes can replace old versions of the same receivers dynamically in running system.

Sending and receiving messages with SM. Message 'store and forward'

In addition to standard Neutrino messaging SM provides message 'store and forward' facilities. This can be used for senders that need to post messages when receivers are not ready yet to process these messages without blocking senders. Very well-known concept used in e-mail systems.

SM framework provides all necessary control both for servers and receivers to enable and parametrize their 'store and forward' capabilities as needed. In this case, SM data is not stored in a central place, data make 'lazy' migration when needed from node to node (host to host). When data migration happens it is copied across network with QNX messaging mechanism.

Both attribute-based addressing and 'store and forward' facilities allow for processes to create and use customized Blackboard communication models with different parameters of these to be fine-tunned both by senders and receivers.

Optimized for efficiency data transfer with SM

SM provide flexible and powerful mechanism for efficient, optimized data transfer. SM framework may be used by its clients (senders and receivers) for automatic and transparent handling of message data buffers. In this 'optimized' mode SM framework itself detects the collocation of sending and receiving end-points and choses the fastest way to transfer the data taking the burden of this work from the clients.

Another, 'explicit' mode allows sender to explicitly request SM framework to pass a message either as a pointer to message data or as an actual copy of the data. In case 'pointer mode' does not make sense as in case with remote receiver running on a separate node, SM framework will return appropriate error code to the sender.

To optimize data transfer SM framework :

- Provides clients with means to create custom message protocols specialized for the particular application needs. Specialization here allows to minimize protocol overhead that general-purpose protocols involve and make the protocol 'just right' in other words as simple as possible for any given set of application senders and receivers.
- Provides minimal base protocol that allows receiving part of SM framework to determine the type of message buffers sent: fragmented pointer-based buffer (IOV), data copy or shared memory object.
- Transparently makes optimization decisions for data transfers between processes running on the same node. In case any unfragmented message buffer is equal to

K * MMU_Page_Size, framework maps this buffer to shared memory object on sender size and remaps it back to receiver process memory. Otherwise message buffer is copied between processes. Our experiments with Intel QNX Neutrino 6.1 show that shared memory exchange becomes faster then data copy starting with data buffer size >= 128 Kb. With smaller sizes copy is much faster then mapping / remapping of buffers.

Note: The size of the data buffer when this decision is made as well as many other SM parameters can be configured by application.

•

SM Synchronization.

Message 'store and forward' mechanism adds new synchronization model to basic Neutrino synchronization with MsgSend(), MsgReceive(), and MsgReply() primitives also 'inherited' by SM.

Senders and receivers may explicitly request SM framework to use 'post' message passing mode. In this mode senders post messages to SM framework to be retrieved any time later by receivers Sender may specify 'post mode' when sending a message.

In this mode message is not sent directly to receiver but is stored in SM Framework on behalf of the sender. Sender is not get blocked by receiver and control immediately returns to the sender. In 'post mode' receiver my specify messages to be delivered as soon as they arrive or request delivery explicitly at some time later. Receiver may set delivery options by sender and message types.. Senders can also specify message expiration interval during which messages will be waiting to be consumed. Thus both senders and receivers can now transfer messages asynchronously.

Conclusion

Smart Messages provides unique facilities for message passing and distributed process orchestration that don't exist in shared memory frameworks and libraries. SM extends Neutrino messages in many powerful ways that make possible attribute-based messaging, process meta data, blackboard and asynchronous communication, message 'store and forward'. Besides SM allows to optimize message communication for speed and efficiency thus in most cases eliminating the need for other IPC including shared memory which SM utilizes in efficient way transparently for its clients.