NORMA IPC Version Two: Requirements

Edited By
Alan Langerman
alanl@osf.org

Open Software Foundation
Research Institute
One Cambridge Center, 11th Floor
Cambridge, MA 02142

August 23, 1993

Contents

Acknowledgements

Many people have contributed to the development of NORMA IPC: Joe Barrera, once of Carnegie-Mellon University and now of Microsoft, provided the original design and implementation. Subsequent contributors included (alphabetically) David Bernard, David Black, Randy Dean, Michelle Dominijanni, Jeffrey Heller, Alan Langerman, Dejan Milojicic, David Mitchell, Eric Paire, and Steve Sears. Bill Bryant of OSF/RI has prepared a document explaining the theory and practice of NORMA IPC Version One. There are probably other contributors; if you are a contributor and your name is not listed, please contact me so I can update the list.

A number of people, including those named above, have so far contributed suggestions for the NORMA IPC rewrite. David Black, Michelle Dominijanni, Ken Hibbard, Roy Larsen, Eric Paire, Paul Roy, Steve Sears, Tom Talpey and Franco Travostino all commented on an early draft of this paper. Roy Larsen also prepared a requirements document for Intel Supercomputer, from their perspective, which makes many of the same points discussed here. Much of this material derives from an analysis of NORMA IPC deficiencies and remedies prepared in December, 1992.

Chapter 1

Introduction

The Open Software Foundation Research Institute intends to rewrite the NORMA IPC layer of the Mach microkernel. The NORMA IPC layer provides a seamless abstraction of port and message between separate nodes in a communications domain. User-level tasks use the same interfaces, with the same semantics, to send messages between tasks on separate nodes as well as between tasks on the same node. The NORMA IPC layer was originally designed and implemented as a prototype, for proof-of-concept, and has confronted increasing strains in a production environment. This document presents the arguments in favor of rewriting NORMA IPC and the requirements OSF imposes on such a rewritten subsystem.

Unlike a traditional requirements document, on occasion this brief also discusses some design and implementation considerations. Our accumulated experience with NORMA IPC suggests that any new version of this subsystem must confront these issues. We are particularly concerned with interactions between IPC and the VM pageout path.

We eagerly solicit feedback on this proposal. Do you agree with our proposed requirements? What might you add, subtract, or alter? What experiences have you had with NORMA IPC that might shed new light on the needs of a rewritten subsystem? While it will not be possible to accommodate every proposed requirement or suggestion, the more we learn now the better the resulting NORMA IPC will be.

Comments should be addressed to alanl@osf.org.

Throughout the remainder of this note, we refer to the original NORMA IPC design and implementation as NORMA IPC Version One, or more simply as Version One. The proposed rewrite is dubbed NORMA IPC Version Two, or just Version Two.

Interested readers may also wish to consult one or more of the following papers:

- *NORMA IPC: A Task-to-Task Communication System for Multicomputer Systems*, Bill Bryant, Open Software Foundation Research Institute.[1]
- *A Fast Mach Network IPC Implementation*, Joe Barrera III, Mach II Symposium, USENIX Association, November 1991.
- *UNIX File Access and Caching in a Multicomputer Environment*, Paul Roy, Mach III Symposium, USENIX Association, April 1993.
- *Memory Exhaustion Via Reliable NORMA IPC Transport*, Alan Langerman, Open Software Foundation Research Institute, February 1993.

[1] Currently in draft form, dated 7/29/93, and may not be generally available until late August or early September.

- *A Fast and General Implementation of Mach IPC in a Network*, Hilarie Orman, Mach III Symposium, USENIX Association, April 1993.

Chapter 2

NORMA IPC Version One

2.1 Purpose of NORMA IPC

The existing node-to-node communications protocol employed between cooperating Mach kernels in a multicomputer is called NORMA IPC (No Remote Memory Access Interprocessor Communication). NORMA IPC was originally developed at Carnegie-Mellon University along with the rest of the Mach kernel and the support code necessary to extend Mach into a multicomputer environment.

NORMA IPC has several uses. The Open Software Foundation Research Institute provides OSF/1 AD, a distributed version of UNIX that provides a single-system image across all the nodes in a NORMA domain. Others in the research community, such as the University of Grenoble, in France, the University of Kaiserslautern, in Germany, and the University of Utah, in America, use NORMA IPC for distributed applications, process migration, load balancing and other research.

A NORMA multicomputer domain may consist of a single machine with a high-speed interconnect among many nodes. Examples of such machines include the Intel Hypercube and the Intel Paragon. At the other end of the spectrum, a NORMA domain may consist of multiple PCs interconnected via Ethernet. In practice, the low-end configurations provide testing and debugging environments while the high-end configurations provide production environments. Some configurations may provide reliable hardware transport (such as the Intel Paragon) while others provide unreliable transport (such as Ethernet). OSF/RI recognizes that performance considerations may vary radically along the axes of interconnect speed, processor type and speed, and number of nodes in a domain; to date, OSF/RI has been more concerned with performance in high-end production environments.

The NORMA subsystem is crucial to the correct implementation of a single-system-image operating environment: in effect, it extends single-node Mach IPC across multiple nodes, thus providing the illusion of a single IPC space. Mach IPC provides naming, synchronization, security and communications abstractions for the rest of the Mach kernel as well as for systems and user applications. NORMA IPC provides the single-node Mach IPC semantics across multiple nodes. These semantics include in-order, reliable delivery of messages; port right passing; deadname notification; and so on. Seamlessly extending Mach IPC to all nodes in a multicomputer, via NORMA IPC, is a primary requirement for the success of a multicomputer operating system.

We digress for a moment to note that NORMA IPC extends the Mach abstraction of messages being sent from port-to-port across nodes. In this document, we will refer to each port-to-port relationship as a connection. We call the relationship between two nodes a channel. Multiple connections may be multiplexed over a single channel.

Given the frequency of Mach IPC use for many workloads, such as general purpose time-sharing, the performance of NORMA IPC will be critical to overall system performance. While specialized workloads may limit their use of IPC, in practice applications that make use of any Unix or Mach services (including virtual memory allocation and paging) will be using Mach IPC. On a multicomputer, of course, uses of Mach IPC frequently become uses of NORMA IPC.

Because NORMA IPC is so fundamental to Mach in a multicomputer environment, its performance and correctness become of great concern.

2.2 Deficiencies of Version One

The current implementation of NORMA IPC Version One originated as a prototype for demonstrating the feasibility of Mach as a multicomputer operating system. Today's NORMA IPC is thought to function fairly reliably, although this reliability has been purchased at great expense in engineering time at OSF/RI, Intel and other sites over the past year.

However, NORMA IPC Version One suffers from several major defects. Foremost, the underlying communications protocol, or transport, is inefficient. The subsystem uses a window size of one packet between nodes. In one version of the protocol, the receiver can only handle one incoming packet at a time; in another version, the receiver can accept one packet from each sending node at a time. In either case, the protocol is effectively "stop and wait": send a packet, then cease transmission until an explicit acknowledgement is received from the other side.

This protocol is not well-tuned for the needs of Mach IPC. For instance, when executing RPCs, it is conceivable that the acknowledgement of the requesting packet could piggyback with the response, thus requiring only two packets. The current protocol demands four: request, acknowledgement, response, acknowledgement. Worse, no other transmission from the sending to the receiving node may take place during each (packet, ACK) sequence.

In another instance, consider the transfer of out-of-line data or ports in a Mach message. An individual message could specify the movement of many pages at one time. Ideally, the communications protocol would allow the streaming of bulk data transfers. In practice, NORMA IPC throttles such a transfer with its stop-and-wait nature: only one page can be sent at a time.

One possible variation on the current stop-and-wait protocol would be allowing one packet to be outstanding between each port on a pair of communicating nodes. Each port connection would still be stop-and-wait but multiple connections could progress simultaneously, improving overall throughput between nodes. Message ordering from sender to receiver would be maintained. Message throughput on a single port would not improve in any way.

Another possible variation would be to permit mixing packets from different messages but always guarantee that the final packet in a message is strictly ordered with respect to the final packets of any other messages to the same port and/or from the same sender. This variation would permit longer messages to be interleaved, increasing throughput. Shorter messages (especially those that fit in a single packet) would benefit less.

Unfortunately, implementing either variation requires significant surgery on NORMA IPC. Currently, NORMA IPC has no data structures that represent progress on a port connection; only progress for a packet at a time between a node pair. Furthermore, large portions of the current NORMA IPC flow control protocol, the packet sequence number engine, and the message reception path depend on the assumption that there is only one packet at a time outstanding between a sending node and a receiving node.

Clearly, the protocol requires a redesign: at a minimum, the window size must be increased and optimizations common to other network protocols such as bulk data streaming and acknowledgement piggybacking should be incorporated. Unfortunately, important parts of Version One outside the transport proper have implicit dependencies on this window size; increasing the window size demands fixing or rewriting sensitive portions of NORMA IPC Version One.

One major portion of Version One that depends on "stop and wait" is the suspend/resume protocol responsible for implementing flow control in memory-poor situations. This protocol throttles ordinary packets in favor of paging-critical messages. Its entire implementation assumes that only one packet at a time can be outstanding on a connection and further assumes that only one connection can be active at a time over a node-to-node channel. These assumptions become particularly critical when handling suspend/resume notification retransmission in a cluster environment.

The "stop and wait" assumption also underlies the implementation of acknowledgement (ACK) processing in Version One. Between the time a packet is received and the time its ACK is sent, NORMA IPC assumes that no other packet will arrive from the same node. The ACK thus may be deferred until almost any convenient time. Acknowledgement handling therefore may be used to restart network reception. Further, because no other packet may arrive during this time, NORMA IPC assumes that the protocol control data structures that maintain the state of the channel will not change between the time a packet is received and the time its ACK is generated. Thus, the ACK processing may assume that these state variables are still valid. Increasing the window size will violate all of these assumptions.

Another convenient side-effect of NORMA's "stop and wait" implementation is in-order delivery of Mach messages. Because only one packet at a time is sent, all of the packets of one message will arrive at the receiving node and be re-assembled before the packets of the next message are collected. NORMA currently assumes that, as soon as a message has been received in full (possibly consisting of multiple packets), it may be placed directly on the receiving port's message queue. In fact, this assumption can only be valid in the current NORMA IPC implementation if the window size is one packet. Increasing the window size will require additional logic to preserve Mach message ordering.

Finally, the transport is itself tightly integrated into the NORMA IPC stack. It is not possible to compile more than one communications protocol at a time into a given kernel, much less support the use of those implementations simultaneously.

The second major deficiency of the NORMA IPC Version One implementation is its uniprocessor orientation. Version One was never intended to run in an MP-safe, much less MP-efficient, environment. However, if a Mach kernel is to run on a multicomputer with multiprocessor nodes, NORMA IPC will have to be made at the least MP-safe and at best MP-efficient.

"Incremental" parallelization has been proposed as one way to redress the lack of MP support in Version One. This term covers a variety of approaches so it is important to define which approach we analyze. For instance, incremental parallelization can mean adding MP support to one layer of code while leaving a higher (or lower) layer unparallelized or protected by extremely coarse synchronization. Then, additional layers can be parallelized or the locking strategy of an existing parallelized layer can be refined further. Unfortunately, although Version One has a separation between transport and IPC semantic layers, it is not practical to parallelize only one layer but not the other.

Alternatively, incremental parallelization can mean adding MP support to an entire subsystem, then refining that support for additional parallelism in critical areas. This approach could be applied to NORMA IPC. For instance, a global lock could be defined for serializing most of NORMA while smaller parts are parallelized using finer-grained locks. However, this strategy has at least two drawbacks. First, every iteration of the incremental approach will require removing and reinserting uses of this global lock. This effort is wasted from the standpoint of ultimately providing a completely MP-efficient NORMA IPC implementation. Second, it will be difficult to aggressively test each stage of the incrementally parallelized product because the serialized code will gate the performance of the entire subsystem. Third, due to the amount of processing occurring at interrupt level, which is not well isolated, there is a real possibility of deadlocks that are not obvious.

Yet another possibility is to make NORMA IPC Version One MP-safe, rather than MP-efficient, by introducing a few global locks and new threads. This is the approach that has been taken at the Research Institute's Grenoble facility, by Eric Paire, to produce a NORMA kernel that can run on a multiprocessor. This work was done to support an MP-efficient implementation of the XMM subsystem. However, for a long-term MP-efficient NORMA subsystem, there is no alternative to fully parallelizing the code.

Both versions of incremental parallelization presuppose that the existing uniprocessor functionality of NORMA IPC is desirable and should be retained. However, that is most certainly not the case. For all the reasons outlined in this document (single-stream protocol performance, robustness, maintainability, fault tolerance, etc.) a redesign of NORMA IPC is highly desirable. In other words, NORMA IPC should be redesigned and reimplemented with the twin goals of uniprocessor and multiprocessor efficiency in mind. This redesign will inevitably lead to new data structures, which can also be designed with locking constraints as a consideration.

The third significant drawback of the existing NORMA IPC code is the lack of support for fault tolerance. By fault tolerance, we mean handling the dynamic addition to or removal from a NORMA domain of individual nodes. In fact, Version One does have some logic to detect rebooted nodes. However, Version One does not as a whole consider this problem and take all necessary actions. When informed of a node failure, a future version of NORMA IPC should be able to deallocate all associated data structures, destroy any ports associated with the node, generate any necessary notifications, etc. When recognizing a new node, or a rebooted node, NORMA IPC should take comparable actions.

A fourth notable issue for NORMA IPC is the current scalability limitation. Apart from the lack of multiprocessor support, NORMA IPC data structures do not scale well to multicomputers composed of thousands of nodes. Arrays are allocated for the maximum number of nodes in a system, regardless of how many actually communicate with a given node or even how many actually exist. Linked lists are frequently used in situations that will bottleneck the implementation as multicomputer size scales up. Algorithms are used that almost certainly will not scale well to very large configurations. (Other Mach subsystems may have similar limitations but those considerations are beyond the scope of this document.)

Fifth, the original design of NORMA IPC did not contemplate the intricate interaction between the needs of NORMA IPC for memory and the requirements of the VM subsystem for paging activity. Here are some examples from NORMA IPC Version One. The critical pageout path sends pages to the system's default pager. When the default pager is on a node other than the node generating pageouts, problems may result as follows. The memory-starved node generates up to vm_pageout_burst_max pageout requests from the pageout daemon. These requests take the form of an initial memory_object_create request followed by vm_pageout_burst_max memory_object_data_write messages, each of which contains one out-of-line page being paged out. After receiving the memory_object_create, the default pager node responds with a norma_ipc_pull_receive request to move the receive right from the starving node to the default pager node. Along with the receive right come the vm_pageout_burst_max queued pageout messages. NORMA IPC requires the use of additional pages to send the pageout messages as well as to respond to the norma_ipc_pull_receive request. Thus, there must always be some memory available to handle the pageout path, even when memory runs low.

A node in general must also recognize low memory conditions and disallow packet reception. Otherwise, a node can run out of memory. The suspend/resume protocol implemented over the past year handles this flow control. When low memory conditions occur, the protocol informs sending nodes that transmissions should be suspended. A suspend does not actually disable all transmissions from a node, only the current connection. Messages from other ports queued for transmission from the sending node may then be sent, in case there are any pending paging-critical messages. Failing to send paging-critical messages can result in indefinite postponement problems. Furthermore, even memory-starved nodes must accept paging-critical messages if at all possible; and, if not, must drop incoming packets silently rather than responding with a suspend notification.

Another memory-starvation scenario arises when a single node is the target of pageout requests from many nodes. NORMA IPC Version One more-or-less gracefully suspends transmissions when memory runs low, allowing the target node to clean its own memory. But if the pageouts overrun the target pager, the target pager eventually will itself be paged out. When this case occurs, subsequent pageout requests will be queued on pager ports but never processed. Ultimately, the target node winds up with most of its memory contained in wired pageout messages sitting on ports belonging to a pager that has been paged out.

Considerable effort has been expended to ensure reasonable behavior by NORMA IPC and VM when memory runs low. In this case, "reasonable" means only that the system does not hang or crash in low-memory situations. The suspend/resume and paging-critical paths are quite fragile.

The unanticipated interactions between NORMA IPC and VM has led to a significant amount of re-implementation. Any future work for MP support or protocol redesign will have no choice but to consider this issue as a primary requirement. The memory usage requirements of the VM system and NORMA IPC will have to be considered together.

A sixth deficiency of NORMA IPC is worth noting. When operating on top of a reliable transmission medium, such as the Intel Paragon, the NORMA IPC Version One transport layer still implements unreliable transmission semantics. Because the software has reasons to drop packets, the end-to-end channel is unreliable even though the hardware medium provides reliable transport. This deficiency is not unique to NORMA IPC.

Finally, it is worth commenting that the NORMA IPC implementation suffers from a lack of design primarily due to an accretion of bug fixing and emergency re-implementation. The resulting code is fragile in the sense that it is not easy to understand what to modify and why; changes in one area can have unexpected side-effects in many others; performance suffers from earlier design choices and recent implementation decisions. For instance, incoming packets may flow through interrupt level, delayed delivery into a thread context, handoff to another thread context, and ultimate acknowledgement via an AST (asynchronous software trap) that itself will be delayed until the next return to user mode. Version One also has little documentation other than internal comments.

Originally conceived and implemented as a prototype, the current version of NORMA IPC has become as much a burden as a boon. The entire subsystem deserves a complete re-design and re-implementation.

Chapter 3

Version Two Requirements

We propose that NORMA IPC be completely re-designed and re-implemented to meet the following requirements. Existing code will not be considered during the requirements and design phases but might be re-used, if it fits the new design. However, re-used code will likely require some modification and at a minimum will require improved documentation. OSF/RI solicits suggestions concerning existing technologies that can be applied to these problems.

We first list the requirements for a new NORMA IPC and then discuss each one.

- rearchitecture of the NORMA IPC subsystem, defining these layers:
  - NORMA IPC Semantics
  - Transport Framework
  - Transport Stack(s)
- introduction of a transport framework
- new transport module(s)
- efficient uniprocessor support
- efficient multiprocessor support
- flow control
- fault tolerance hooks
- scalability to thousands of nodes
- well-known and bounded memory consumption
- anticipate integration of IPC and VM paging
- improved design and documentation

3.1 NORMA IPC Architecture

Today, NORMA IPC Version One is conceptually layered in two pieces: a semantic layer that extends the port and message abstractions across nodes and a transport layer that conveys the NORMA IPC messages required to implement those abstractions. However, the implementation of these layers has unfortunate interdependencies in some instances, particularly with respect to VM operations and stop-and-wait transport assumptions.

Version Two of NORMA IPC should have, conceptually, three layers. The topmost implements NORMA IPC semantics and uses the second layer, a transport framework, to invoke transport services. The third layer contains individual transport stacks. A system may include more than one transport stack, at least at compile time, and may potentially use more than one transport stack at run-time.

In practice, it may be possible to implement the NORMA IPC semantics layer as a module within a transport framework. Regardless, there should be strict definition and separation of the responsibilities of the various layers.

Additionally, there should be a distinct separation between the NORMA IPC semantic layer and other components of the Mach microkernel, especially the VM subsystem. Version Two should have no need to recognize specific messages generated by other kernel subsystems.

3.2 Transport Framework

NORMA IPC Version Two should employ a transport framework that isolates the semantic layers of NORMA IPC from the transport layers. The framework should allow multiple transports to co-exist at compile and at run-time. However, Version Two may initially only permit the use of one transport at run-time (see Chapter 8, Rejected Requirements).

The framework should allow the implementation of individual transport modules without change to the NORMA IPC semantic layers; and should permit the modification of the NORMA IPC semantic layers without changing the underlying transports.

3.3 New Transport Module(s)

NORMA IPC Version Two must include a transport capable of providing significantly greater throughput and lower latency than provided by today's transport. This protocol should be capable of using optimizations to enhance IPC performance, such as multi-packet windows, bulk data streaming, and piggybacking acknowledgements. For the reasons elucidated earlier, the transport mechanism integrated in NORMA IPC Version One is not a good candidate for re-use.

Any transport may be used that conforms with the requirements of the NORMA IPC semantics layer and (if applicable) any transport framework. Transports are *not* part of the Mach definition. RI would prefer to use off-the-shelf transports or transports written by collaborators rather than assuming responsibility for creating and maintaining transport modules.

A transport that can be encapsulated and run over existing networks would be highly desirable. Today, we use Ethernet, but we envision the possibility of using FDDI- or ATM-based networks in the future.

3.4 Uniprocessor Support

At all levels, NORMA IPC Version Two must support multiplexed channels between nodes. Messages between different ports (between the same node pair) must be able to be sent in parallel; messages between different node pairs must also be able to be sent in parallel. See section 3.8 for additional discussion of message delivery issues.

It should be possible to build a uniprocessor-only version of a NORMA kernel that will eliminate any multiprocessor overhead.

3.5 Multiprocessor Support

A major functional deficiency of NORMA IPC Version One is the lack of efficient support for tightly-coupled, shared-memory multiprocessors. A non-NORMA Mach kernel can run efficiently on machines containing twenty to thirty shared processors; a NORMA kernel should meet the same goal.

Although not yet proposed as a requirement, the design of Version Two should not preclude a preemptible implementation. The current NORMA subsystem, like the rest of Mach, has never had preemption as a design goal. Given other work in progress on Mach, we should consider adding a preemption goal to the design of a re-architected NORMA IPC. A lock-based, multiprocessor-efficient implementation will go a long way towards providing the base for a preemptible implementation. It may be the case that a completely preemptible implementation may not be produced until some future date, but if at all possible the project proposed in this document should refrain from designing features that preclude moving to a preemptible implementation later.

3.6 Flow Control

At all levels of the design, NORMA IPC Version Two should address the problems of user-level and kernel-level message flow control.

3.7 Fault Tolerance

NORMA IPC Version Two must support dynamic node addition to and deletion from a running NORMA domain. The precise nature of this support is not yet defined although the hooks should be kept very simple. When informed of a node failure, Version Two should be able to clean up any associated resources and propagate error information (e.g., by destroying ports and generating notifications). When detecting a new node or a new incarnation of a node, or when sending to a new node, NORMA IPC should be able to construct the necessary data structures to permit communication.

3.8 Message Delivery Semantics

Currently, NORMA IPC guarantees that the sender of a message does not continue until the message has been received on its target node. Message delivery across nodes ignores normal queue limit semantics. These properties will be reconsidered in NORMA IPC Version Two.

The semantic that the sender pause until the message has been received at the target node – or until the message has been queued on the target port – limits the rate at which a single sender can push data to a receiver. Essentially, this semantic enforces stop-and-wait between any given sender and receiver. However, this semantic is crucial for correct fault-tolerant operation in the presence of partial failures. Without this semantic, it is impossible for the sender to know whether the message has moved from the sender's failure domain into the receiver's failure domain.

In NORMA IPC Version Two we should consider the possibility of relaxing the message delivery semantic. By allowing the sender to continue sooner than otherwise, the sender can overlap computation with message transmission. In some cases, this ability may mean increased throughput from sender to receiver. This relaxed behavior must be optional, and not the default behavior. Thus, it may be necessary to modify the mach_msg interface (in a backwards-compatible way) to allow for the inclusion of an additional bit specifying the relaxed behavior. It is not a requirement that NORMA IPC Version Two must support a relaxed delivery semantic; it is only a requirement that the problem be considered in detail.

On the other hand, NORMA IPC Version Two must support the Mach queue limit semantic that currently is ignored between nodes. This queue limit applies to messages sent from user-level applications. Even in a uniprocessor implementation, Mach ignores queue limits for messages originating from the kernel.

3.9 Scalability

NORMA IPC Version Two must use data structures and algorithms that permit the operation of a multicomputer with thousands of nodes. Reasonable limits will have to be determined during the initial, investigatory phase of this project on the total number of nodes anticipated in a system and the total number of nodes that might communicate with any given node. Data structures should be created dynamically so that small and large configurations can be supported without wasting memory.

3.10 Memory Consumption

All layers of Version Two will have documented memory usage patterns. Data structures should be compact and reclaimed promptly.

3.11 Integration of IPC and VM

Anticipate the memory and paging constraints of the VM and NORMA IPC subsystems as part of the initial design and the subsequent implementation.

3.12 Configuration

NORMA IPC Version Two must conform with any new domain configuration proposals that emerge for clusters or multicomputers.

3.13 Backwards Compatibility

NORMA IPC Version Two need not be backwards compatible with Version One.

However, NORMA IPC Version Two should include provisions for protocol version identification, so that future revisions of the protocol might be able to interoperate with earlier versions.

3.14 Instrumentation and Diagnostics

Wherever possible, at all levels of NORMA IPC Version Two, including its transports, statistics should be maintained for use in performance and traffic analysis. These statistics may be a compile-time option. If possible, a consistent set of interfaces should be created to allow the same tools to access all the data from different levels of the implementation.

The design and implementation should permit extensive self-checking of correctness. Verifying the state of the protocols and transports should be built-in and not added later. These checks almost certainly must be compile-time options to avoid degrading the performance of a production kernel.

3.15 Design and Documentation

Produce a design that reflects the needs of OSF/RI and major customers and that can be used not only to guide the implementation but also as documentation. Produce an implementation with clearer internal documentation as well as external documentation. Update the design documentation during and after the implementation in the event of changes.

In the following sections, we analyze particular requirements as they apply to each layer of NORMA IPC Version Two.

Chapter 4

Semantic Layer Requirements

4.1 Multiprocessor Support

The semantic layer must be fully parallelized and capable of supporting multiprocessor configurations of twenty to thirty processors. This support shall be compile-time selectable, so that a uniprocessor version may be built that eliminates unnecessary locks or other SMP overhead.

4.2 Fault Tolerance

The NORMA IPC semantic layer will not be responsible for determining whether a node has become unavailable. That responsibility shall lie with either the underlying transport or with higher-level software. The semantic layer will export well-defined interfaces for the use of the transport and/or higher-level software. The interfaces for the use of the transport need not be identical to those intended for higher-level software. Implementors of those layers will have responsibility for using these interfaces to notify the semantic layer of node death.

Upon being informed of a node death, the semantic layer shall be responsible for destroying ports, generating notifications, freeing resources and otherwise handling all necessary clean-up activities. NORMA may then reject further messages to or from that node until the node is once again declared “alive”. The precise definition of the notification mechanism and its obligations remain to be determined.

A comparable requirement should be imposed for new node recognition. There should be an additional interface (for transport and/or higher-level software) to inform NORMA IPC that a new node has become active. It is easy to conceive of a transport notifying NORMA that a new node is on-line; but higher-level software could only notify NORMA of a new node if some other, non-NORMA communications path exists whereby higher-level software can learn of the new node. In any case, Version Two must be prepared to create dynamically the data structures required to represent a new node. It is worth pondering whether NORMA IPC should generate a notification of some kind when a new node is recognized.

4.3 Scalability

The semantic layer must allow for NORMA domain configurations of thousands of nodes. For future growth, the semantic layer probably should be able to accommodate configurations in excess of ten thousand nodes. Per-node data structures should be as small as possible and should only be allocated upon demand.

NORMA IPC uses an identifier, called the *uid*, to represent ports on different nodes. This identifier currently includes the node number where the port was created and additional bits that uniquely identify the port. The node value in a NORMA IPC uid is currently limited to 12 bits, or 4096 nodes. Given the size of configurations now under construction, this limit should be removed as part of this project. A new format for a NORMA IPC uid must be defined.

4.4 Integration of IPC and VM

Where appropriate, the semantic layer shall be cognizant of interactions between IPC and VM, especially with respect to paging. The semantic layer should endeavor to reduce or eliminate requirements on the transport layer to support prioritized paging channels. In particular, the semantic layer should reduce or eliminate the need to support incoming messages on a memory-starved node to complete a pageout.

4.5 Flow Control

The semantic layer shall provide flow control for user messages. This flow control may take the form of exporting port queue limits across nodes. If possible, the semantic layer should also provide flow control for kernel messages (a considerably trickier issue).

4.6 Semantics

NORMA IPC Version Two shall support all Mach IPC semantics. In particular, Version Two shall support the transfer of arbitrarily-large Mach messages that need not be entirely memory-resident. Version Two shall implement port queue limits across nodes. Version Two shall also correctly support dead name detection in all cases.

4.7 Transport Utilization

The semantic layer will not use the transport layer except to convey data, port rights and other explicitly requested information. For instance, the semantic layer will not use retransmission timeouts, query timeouts, keep alive messages, etc.

The transport layer is expected to provide reliable, unduplicated message transport but is not required to provide in-order delivery. The semantic layer assumes responsibility for assembling messages in the correct order.

Chapter 5

Transport Framework Requirements

5.1 Multiprocessor Support

The transport framework shall permit efficient support of a multiprocessor configuration containing twenty to thirty processors. Locking entire transport modules while handling a single request is not sufficient.

5.2 Fault Tolerance

There may be a requirement that the transport allow independent communication to nodes in the face of arbitrary delays when communicating with one or more nodes. For instance, an arbitrary delay in communicating with node A shouldn't prevent communications with node B.

5.3 Scalability

If the transport framework requires its own data structures, those data structures must be compact and allocated on demand, to be able to support small and large (ten thousand+ node) configurations.

5.4 Integration of IPC and VM

No specific requirements identified at this time.

5.5 Design and Documentation

No specific requirements identified at this time.

5.6 Framework Choice

One major consideration for redesigning NORMA IPC is the use of a protocol framework. It would be wise to isolate the communications protocol as much as possible from the rest of NORMA IPC, to facilitate using new communications protocols at a later date. Changes in communications media (and in software?) make it unlikely that a single protocol will ultimately prove optimal for all environments. Already, NORMA IPC is expected to handle media speeds differing by at least two orders of magnitude. While the proposed NORMA IPC redesign only includes one new protocol, providing a protocol framework permits the inclusion of new protocols without reimplementing the rest of NORMA IPC.

Note that it may be possible to use a protocol framework as the basis for a single networking paradigm within the Mach kernel and possibly for Mach servers as well.

The following design constraints should be considered when selecting or defining a protocol framework:

- it should be standardized in some way so that additional protocols can be incorporated into NORMA without significant surgery on the protocol or on NORMA
- it should be MP-efficient
- the overhead imposed by the framework should be low relative to the overall cost of NORMA IPC
- it may need to support multiplexed connections at different priority levels to accommodate the requirements of paging-critical messages at the boundary between NORMA IPC and Mach VM
- it should be unencumbered by licensing restrictions, or should at least be easily and cheaply available
- the framework should be modular, intuitive and well-documented

Unfortunately, it is going to be difficult to obtain such a protocol framework for a variety of reasons. Consider the following possibilities.

1. Use STREAMS. This framework would be excellent in terms of the availability of communications protocols for it. The run-time cost of STREAMS is the subject of some debate. However, in any case, licensing restrictions make its use in NORMA IPC impractical. As part of the Mach microkernel, NORMA IPC may not use technology that would encumber the kernel with licensing restrictions.
2. Use a STREAMS-like framework. The major problem with this proposal is that such a framework will be unique to OSF/RI and Mach. Unless the STREAMS interfaces are reproduced with a high degree of fidelity, it will be a non-trivial matter to port any STREAMS-based protocol into our STREAMS-like framework. It is more reasonable to define a non-STREAMS framework that will have the same drawback (unique to Mach) but at least could have the additional benefits of being simple and better-tailored to the needs of Mach and NORMA IPC.
3. Use BSD sockets. Sockets have no licensing encumbrances and support more free protocol stacks than any other option. Knowledge of sockets is wide-spread.

4. Use the x kernel. The x kernel is available without encumbrances. On a technical level, the x kernel has found some acceptance in the research world and has some protocol implementations available. There is a protocol graph implementation that provides NetMsgServer semantics – conceivably, this protocol stack could find use in NORMA IPC Version Two.

   One version of the x kernel uses coarse-grained locking to implement multiprocessor support. However, this version does not offer sufficient parallelism for RI's purposes. Possibly it could serve as a foundation for further parallelization.

   A new feature in the x kernel, fbufs, uses shared memory to avoid data copies and caches these buffers to avoid virtual memory mapping operations. Preliminary results suggest using fbufs permits several orders of magnitude improvement in moving data across protection domains, when compared with using traditional Mach out-of-line memory. However, use of fbufs requires interface changes: messages can have in-line, out-of-line, and fbuf data. Interface changes imply some recoding for those programs that wish to take advantage of the feature. This price might be acceptable for selected servers. We have not yet seen data for the case of using fbufs to send data over a network rather than between protection domains on a single node.

   In addition, the x kernel is thought to have resource consumption problems (memory, threads). The x kernel does not yet handle traffic with differing resource allocation requirements; e.g., a class of traffic that requires wired threads and non-blocking memory allocation versus an ordinary traffic class that uses unwired threads and possibly-blocking memory allocation. Finally, while protocols developed using the x kernel have been put into production, the x kernel itself has not been used in a production environment.

   It would be essential that any parallelization or productization work done by OSF become part of the mainstream x kernel development at Arizona, with major protocol implementations converted to use the new framework. Otherwise, OSF/RI will once again wind up with a unique framework (see option 2).

5. Roll our own framework. While no other protocols initially would fit into our framework, we could keep the framework simple and straightforward. Perhaps by considering the needs of other protocols (e.g., TCP), we could design a framework that would accommodate other protocols without too much work. The major benefit of this approach is maintaining a focus on the needs of Mach and NORMA IPC. The major drawback is that the incorporation of new protocols is likelier to be more difficult than anticipated at the time we design our unique framework.

5.7 Transport Guarantees

The transport framework and any transport modules shall guarantee to provide reliable, unduplicated delivery of messages handed off from the NORMA IPC semantic layer.

The framework and modules shall not be required to provide in-order message delivery. The NORMA IPC semantic layer will sequence and re-assemble messages in the correct order.

Chapter 6

Transport Stack Requirements

6.1 Multiprocessor Support

The transport must be fully parallelized. It must also be capable of being built in a uniprocessor-only configuration.

6.2 Integration of IPC and VM paging

The transport must support prioritized connections, where at least one priority means: guarantee message delivery even in the face of a memory shortage or paging crisis. To support this case, the transport may have to maintain a special pool of reserved memory.

6.3 Platform-Independent Transport Requirements

A transport may be developed that is separated into platform-independent and platform-dependent pieces. This section serves as a placeholder for requirements imposed on a platform-independent transport module. We expect these requirements will be fleshed out during the design phase.

6.4 Platform-Dependent Transport Requirements

A transport may be developed that is separated into platform-independent and platform-dependent pieces. This section serves as a placeholder for requirements imposed on a platform-dependent transport module. We expect these requirements will be fleshed out during the design phase.

6.5 Interface Requirements Between Transport Modules

A transport may be developed that is separated into platform-independent and platform-dependent pieces. This section serves as a placeholder for requirements imposed on the interfaces between platform-independent and platform-dependent transport modules. We expect these requirements will be fleshed out during the design phase.

Chapter 7

Miscellaneous Requirements

7.1 Small and Large Message Support

If possible, Version Two should make special provision for small and large messages. The current NORMA IPC receives all messages, regardless of size, in one or more VM pages. However, a significant number of messages consume much less than a single VM page. As system load increases, these messages tend to remain queued longer, while occupying more memory than necessary, thus exacerbating the system's virtual memory load. Small messages should be handled in a more efficient manner to reduce memory consumption.

This support may be selectable, at compile- or run-time, to trade time (by eliminating data copies) against memory space. Being able to select this option at run-time may be a good idea for low-memory situations.

7.2 Pageout Path

When initiating paging, the VM system currently uses memory_object_create to establish a new temporary paging object. The definition of this interface includes a receive right, on which may be queued a number of pageout messages. The current NORMA IPC implementation of receive right migration requires the node accepting the receive right to send a message to the node transferring the receive right. Thus, at a time when memory is critically low, the sending node must be able to accept a message from the receiving node. Either the way in which a NORMA system pages must change, or the implementation of receive right migration must change, to remove this problem.

One choice is to alter receive right migration so that the migration happens entirely through messages "pushed" from the sender. The preferred alternative is to modify the pageout path to eliminate the need to migrate a receive right at all. Given the expense of migrating port rights, eliminating the receive right migration will benefit local paging as well as NORMA paging.

There may be other ways in which NORMA paging can be improved to reduce or eliminate interactions between VM and NORMA IPC. For instance, it might be beneficial to explicitly reserve memory for pageout replies, lock request-induced pageouts, and possibly even pageins.

7.3 Port Right Transfers

It would be highly desirable if port right transfers can be piggybacked on top of normal data transfers. It is common to send messages with a mix of in-line and out-of-line data and port rights.

7.4 Raised Processor Priority

NORMA IPC Version One spends considerable time with raised processor priority, disabling interrupt processing. Version Two should minimize time spent with interrupts disabled, to increase the efficiency of interrupt processing and to enhance preemptibility. Minimizing the amount of code executed at interrupt level as the result of an interrupt will also simplify locking schemes.

7.5 Heterogeneous NORMA Domains

NORMA IPC Version Two shall not deliberately preclude future support for NORMA domains consisting of heterogeneous processor types. Such a domain may happen to work (e.g., Intel Hypercube with mixed i860 and i386 processors) but this support isn't guaranteed in general. Heterogeneous support is largely a function of data conversion at higher and lower levels: for example, the transport protocol must interoperate between heterogeneous machines and user-level software – possibly MIG stubs – must convert user data as needed. Machine-dependent data representation issues will at least be identified and isolated in NORMA IPC Version Two.

Chapter 8

Desirable (But Unrequired) Features

This section describes features that have been considered but that are not formal requirements of NORMA IPC Version Two.

8.1 Multiple NORMA Domains

On occasion, we have entertained the notion of a machine that participates in more than one NORMA domain. For example, a multicomputer that internally consisted of a number of nodes in a NORMA domain interconnected via a mesh could itself be part of a NORMA domain interconnected via FDDI. However, a number of key issues remain unresolved. Would nodes within the multicomputer be directly addressable by nodes outside the multicomputer? What kind of tuning would be needed for a NORMA domain with possibly radically different processor and interconnect speeds? What kind of information would need to be contained in a NORMA port identifier (uid)? Enough uncertainty remains in this proposal that it is not deemed a requirement. However, this problem should be considered at greater length.

8.2 Routing

Although not a requirement, the design of NORMA IPC Version Two should contemplate the problem of routing between nodes reached via the same transport and also between nodes reached via different transports. The first case may be useful for transputer-like configurations. The second case may be useful for configurations containing nodes interconnected via different media.

A transport may be allowed to re-route a packet at any time assuming that both source and destination nodes are reached via the same transport and that the transport can handle all flow control issues without involving higher layers. The transport is responsible for handling all failure domain semantics. For instance, if the sender is blocked waiting for the receiver to accept the entire message, the transport must preserve this guarantee when routing packets.

Routing between different transport types is not as simple; there may be flow control issues, it may be necessary for the packet to be handed to the NORMA IPC semantic layer before being sent to the new transport, there may be additional failure domain issues, etc.

In either case, although these features are desirable, it remains to be shown whether they can be incorporated easily into a design. Possibly the implementation may allow adding these features at a later date.

8.3 Multicast Support

NORMA IPC Version Two need not support multicast messages. Note that there are no Mach interfaces for sending multicast messages.

Appendix A

NORMA IPC Paging Problems

A.1 Background

We learned the hard way about unanticipated interactions between NORMA IPC Version One and the virtual memory subsystem. This appendix provides detail on some of the problems we encountered and the solutions we implemented. Many of these solutions are ugly. All of these problems must be taken into account when contemplating the design and implementation of NORMA IPC Version Two. This list is not comprehensive.

A.2 NORMA IPC page reservations

NORMA feeds pages to the network for incoming messages in the following way. When an incoming message is dropped, the page containing the message is handed back to the network and the network receive side restarted. When an incoming message is accepted, NORMA grabs a new page with which to restart the network receive operation. Only one page at a time has a receive operation enabled on it.

A page containing an accepted message eventually leaves NORMA, to be fed to the task consuming the incoming message.

Originally, NORMA assumed that whenever it accepted a message there would be a page available in the *netipc page pool* for restarting the network receive operation. In fact, this didn't have to be the case. Pages could leave NORMA without replenishing the netipc page pool. Because NORMA network receive operations take precedence over most other activities, the periodic task responsible for checking and refilling the netipc page pool might be postponed indefinitely. The result was a system that would crash when the netipc page pool emptied.

NORMA now uses a low-level page reservation mechanism. If a page can't be reserved, an incoming message will be dropped and its page used to restart the network receive side. Otherwise, the message is conditionally accepted; if not dropped by subsequent software checks, the netipc page pool is guaranteed to contain at least one page that can be grabbed for restarting the network receive side.

This problem and its solution are particular to the current NORMA implementation and should have no significant impact on NORMA IPC Version Two.

A.3 Handling Large Messages

Mach semantics permit sending arbitrarily large portions of an address space from one task to another. On a single node, this operation is accomplished by remapping the affected pages. On a multicomputer, this operation is accomplished by immediately sending all requested data from the sending task to the receiving task. In the worst case, the amount of memory involved exceeds the total physical memory available on the receiving node – and may exceed the memory available on the sending node, for that matter.

In this situation, NORMA used to send as much of the message as possible to the receiving node. The message itself was received in wired memory, so the node would eventually run out of virtual memory and crash.

The problem was cured by introducing a new path within NORMA that uses pageable memory for holding a message's contents. This path is only used for large messages because its use requires an additional thread and the latency of an additional context switch. The "regular" message reception path is faster.

This solution can lead to fascinating ironies. Consider a two node system, where node A supplies paging services to node B. When sending a very large message from node A to node B, the following may happen:

1. Node A wires down a page of the message to be sent. The page is non-resident, causing the page to be created (zero-filled) or fetched from the appropriate pager.
2. When arriving on node B, the page is assembled into the incoming message and marked pageable.
3. As memory fills up, the pages in the message being assembled are paged out – from Node B back to Node A.
4. Node A may itself begin paging heavily, both to make room for the outgoing message being faulted in and sent and to make room for the pageouts being returned from Node B.
5. Eventually, when node B finishes assembling the message, it passes the message on to the receiving task. At this point, some portion of the message's contents actually reside on node B and the rest is non-resident.
6. As the receiving task accesses the message's contents, it may generate new pagein requests for the non-resident portions of the messages, which may in turn lead to additional pageout requests to make room for the incoming pages. If the receiver linearly accesses the data from beginning to end, it is almost certain to find every single page non-resident at the time of the access.

The number of times a page may travel between Nodes A and B in this situation (3) is surprising, as is the amount of paging activity that goes on.

With the advent of NORMA14, it is now possible to move a large message from one node to another via memory mapping. However, this technique introduces a new dependency between the sending and receiving nodes. Consider Nodes A, B, and C, where C provides paging services to B, and A is sending a message to B. A's paging services come from itself or some other node. In the scenario above, if the message is moved immediately, the receiving task on Node B can operate as long as Node C operates correctly. But if we alter message passing between nodes to instead use memory mapping techniques, after the sending task on Node A unblocks the message still resides in whole or in part on Node A. If Node A fails after having sent the message but before the receiver faults in all the data it needs, the message *appears* to have been sent but is not in fact entirely available to the receiver. The receiver on Node B now depends not only on Node C but also on Node A.

A.4 NORMA Receive Operations v. Pageout

Because NORMA receive operations have higher priority than paging activities, when receiving many messages or very large messages it was possible for NORMA IPC to exhaust memory. All available pages in the system would wind up as wired or unwired pages stuck in the message re-assembly path.

The solution implemented for this problem was a primitive flow control method where the receiving node can *suspend* the sending node. To be precise, what is suspended is not the node but the message currently being sent and any messages queued behind it for the same port. The sender may move on to another message on another port. Once the receiver has entered a low-memory situation, it suspends all non-critical incoming messages until it has a chance to run the pageout daemon and free up memory.

It is not possible for the receiving node to ignore all incoming traffic in this situation because some of that traffic is critical to the pageout path. The following messages are critical:

- norma_ipc_pull_receive
- norma_ipc_node_set
- norma_ipc_node_set (reply)
- memory_object_data_write_completed
- proxy_data_write_completed

The first three messages are needed because a receive right may migrate as part of sending pageouts to a remote pager. The last two messages will be discussed in Section A.5.

Incoming critical messages must be handled with special care. For instance, a special kind of thread must be used. Regular incoming NORMA messages are handled by regular kernel threads, whose stacks are reclaimed by the kernel when the threads block. In a low-memory situation, such a thread may not be able to run because it may not be able to allocate memory for its stack. The NORMA threads that process critical messages must be *wired*, that is to say stacks must have been reserved for them. Any additional memory required by the message reception path must also be guaranteed to be present.

In practice, the suspend method of flow control allows a receiving node time to pageout and then resume re-assembling and delivering messages without exhausting a node's memory.

A.5 Paging Flow Control

Eventually, it became apparent that node-to-node flow control was necessary but not sufficient. One node paging heavily to another could overwhelm the second node's vnode pager, causing it to be paged out. Eventually, the accumulation of messages pending for that vnode pager would expand to fill all available memory.

The normal Mach flow control method, port queue limits, does not prevent this problem. In the first place, Mach does not enforce queue limits on messages originating from the kernel, which is the case with these paging messages. (Enforcing such a limit opens us other attacks on the system, such as causing kernel threads to block unexpectedly.) In the second place, NORMA IPC Version One does not have any notion of queue limit exported across nodes.

The solution implemented for this problem, paging flow control, extends the uniprocessor Mach notion of pager throttling across nodes. For each pageout, the kernel expects the target pager to generate a response, memory_object_data_write_completed. (In some instances, when XMM is involved, proxy_data_write_completed is the handshake.) This response can be used to gauge the progress of the pager: the kernel can detect an overloaded pager and throttle pageouts to that pager.

Paging flow control divides pageouts into four categories: local, internal, external, lock_request. The local category accounts for all pageouts bound for a pager that resides on the same node as the kernel sending the pageout. Internal and external categories account for pageouts destined for a pager on a remote node. An internal pageout is one sent by a kernel to its *default* pager. A kernel sends an external pageout for a page managed by a user-level pager. The last category, lock_request, accounts for pageouts triggered by user-level memory_object_lock_request operations.

The local paging category is essentially unlimited. The kernel can continue to communicate with local pagers, without any loss of performance, as it would in its uniprocessor implementation. More importantly, limiting local pageouts leads to one or more possible deadlock scenarios.

One of the most interesting complications in paging flow control arises from the need to *double-page* in memory-critical situations. In this case, the kernel has realized that available memory is so low that it has no choice but to pageout to the default pager. (Recall that the default pager provides service guarantees not provided by ordinary external pagers.) However, the page to be evicted from memory may in fact belong to an external pager. In order to send the page to its default pager, the kernel must *first* send the page to the external pager that owns it. The kernel does not wait for a response from that external pager; it immediately forces the page into a new memory object managed by the default pager and sends the pageout request out to the default pager.

This behavior seems counter-intuitive. If the external pager has fallen behind on its requests, or if the kernel needs memory desperately, why not just send the page to the default pager directly? The answer lies with the problem of faulting the page back in. The page will only be fetched when a task subsequently faults on it. To satisfy the request, the kernel will request the page from the external pager that manages it. *The kernel maintains no data structures enabling it to realize that the page has temporarily migrated to the default pager.* In the external pager's message queue, the pageout request that the kernel sent when evicting the page will always precede the pagein request. Thus, the external pager eventually acquires the page and can satisfy the pagein request. If the external pager faults on the page, the fault is resolved using the memory object managed by the default pager.

In short, the kernel's pageout loop must be cognizant of situations where remote pageouts may not proceed because of flow control conditions. The pageout loop must avoid flooding pagers with too many requests. When paging, the kernel must avoid generating IPC requests that might themselves block indefinitely due to lower-level flow control. Of course, if the requests return an error instead of blocking, paging flow control is still needed. (Paging flow control uses a reservation scheme that is equivalent to determining when the channel would block.) We have not yet been able to define an end-to-end port flow control scheme that eliminates the need for the higher-level paging flow control protocol. Suggestions are welcome.