roceedings of Sixth ACM Symposium on Operating Systems Principles (November 1977) 23-31.

Task Communication in DEMOS*

Forest Baskett
John H. Howard
and
John T. Montague

Los Alamos Scientific Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

This paper describes the fundamentals and some of the details of task communication in DEMOS, the operating system for the CRAY-1 computer being developed at the Los Alamos Scientific Laboratory. The communication mechanism is a message system with several novel features. Messages are sent from one task to another over links. Links are the primary protected objects in the system; they provide both message paths and optional data sharing between tasks. They can be used to represent other objects with capability-like access controls. Links point to the tasks that created them. A task that creates a link determines its contents and possibly restricts its use. A link may be passed from one task to another along with a message sent over some other link subject to the restrictions imposed by the creator of the link being passed. The link based message and data sharing system is an attractive alternative to the semaphore or monitor type of shared variable based operating system on machines with only very simple memory protection mechanisms or on machines connected together in a network.

1. Introduction

This paper describes the fundamentals and some of the details of task communication in DEMOS, the operating system for the CRAY-1 computer being developed at the Los Alamos Scientific Laboratory. The communication mechanism is a message system with several novel features. In this section we discuss the purpose of task communication and why a message system is an appropriate mechanism on the CRAY-1. In later sections we discuss the features of the message system.

A task consists of a program and its associated state information, including register contents, a memory area, and a link table. A task can be manipulated in certain ways; for example, it may be suspended, swapped out, swapped in, or resumed. We have chosen the term task instead of the term process because it seems less loaded with unrelated or contrary meanings. A task is what might be called a job or a program in simple cases. In less simple cases a job or a program will be a (time varying) collection of related tasks.

The first function of the task communication mechanism is to implement system calls. We think of the system as a collection of permanent tasks with which user tasks communicate. The system calls, i.e., the information that is communicated, we call messages. The mechanism of task communication provides an appropriate link or path between a user task and a particular system task that is to act on a given system call. For example, a user task will typically need links to the file system tasks if the user task is to perform I/O. Such standard links ordinarily will be provided to user tasks in an automatic and transparent way.

* Work performed under the auspices of the USERDA.

The next function of the task communication mechanism is to provide a way for arbitrary and unrelated tasks to communicate with mutual consent. Since communication is via links, we must have a way of exchanging links between tasks. If tasks can exchange links dynamically, then we can have a flexible intercommunication facility. We have defined a standard way of passing links along with messges over existing links. One of the standard links that a task will usually receive when it is created is a link to a Switchboard task that can arrange to get two or more cooperating tasks together.

The last function of the task communication mechanism is to provide a method by which one task can encapsulate another task or group of tasks. An easy and transparent encapsulation facility is desirable for debugging, performance monitoring, and simulation of other operating system environments.

We classify operating system communication mechanisms in two basic types: shared variables and messages. The shared variables approach is typified by semaphores [Dijkstra 68] or monitors [Hoare 74]. Brinch Hansen [73] developed the best-known pure message system. Capability systems [Fabry 74] attempt to control memory sharing but do not imply either of these two types of communication mechanisms. The communication mechanism in DEMOS is mainly a message system.

The CRAY-1 has only one pair of base and limit memory protection registers for tasks; it does not have hardware segmentation or paging. Thus tasks cannot be physically segmented in memory. The message approach seems best if the hardware memory protection mechanism is to be used to isolate any given part of the operating system from other parts [Lampson 76]. Different modules of the operating system can be constructed as separate tasks that communicate with messages. Since each task is isolated by the base and limit hardware registers, tasks are protected from each other. This type of organization also naturally inherits the advantages obtained from minimizing global variables and using value type parameters and results. Furthermore, a message orientation is compatible with more types of distributed processing architectures than a shared variable orientation.

On the other hand, we do not see messages as sufficient for all forms of intertask communication just as value type parameters and results would not be sufficient in a programming language without global variables. We have provided a method of sharing data areas via the same links that are used to route messages. A link with such an associated data area is then like a pointer or reference parameter in a programming language. This data sharing can be used for the intertask communication needs for which messages are inappropriate or inadequate.

With this organization the only part of the operating system that needs hardware access to the memory of more than one task at a time is the kernel, that part of the system which is used to move both messages and data from one task to another. The two types of communication are unified in the link concept. We consider this to be a novel arrangement and we will now show how the two fit together in a harmonious way and give examples of link usage that illustrate the utility of the arrangement.

2. Links

A link permits a task to send messages to another task and possibly to read or write part of the memory of the task to which the message would be sent. For messages, it is a one-way (simplex) communication path. If two tasks wish to send messages to each other in full duplex communication, each must have a link to the other. A link is created by the task to which it points and then passed to the potential sender task.

Links are associated with but maintained outside of the address space of their sender tasks. They may be manipulated only by use of link ID's, which are indexes into a task's Link Table. Figure 1 illustrates this arrangement. Kernel calls which operate on links take link ID's as parameters. All operations on links are performed by the kernel of the operating system; requests to the kernel are made via the CRAY-1 supervisor call instruction.

A newly created task is given an initial set of links by its parent. These links define the environment in which the new task runs. In the case of tasks created by the job initiator the environment consists of standard links to system tasks such as the file system, the task manager, and the Switchboard.

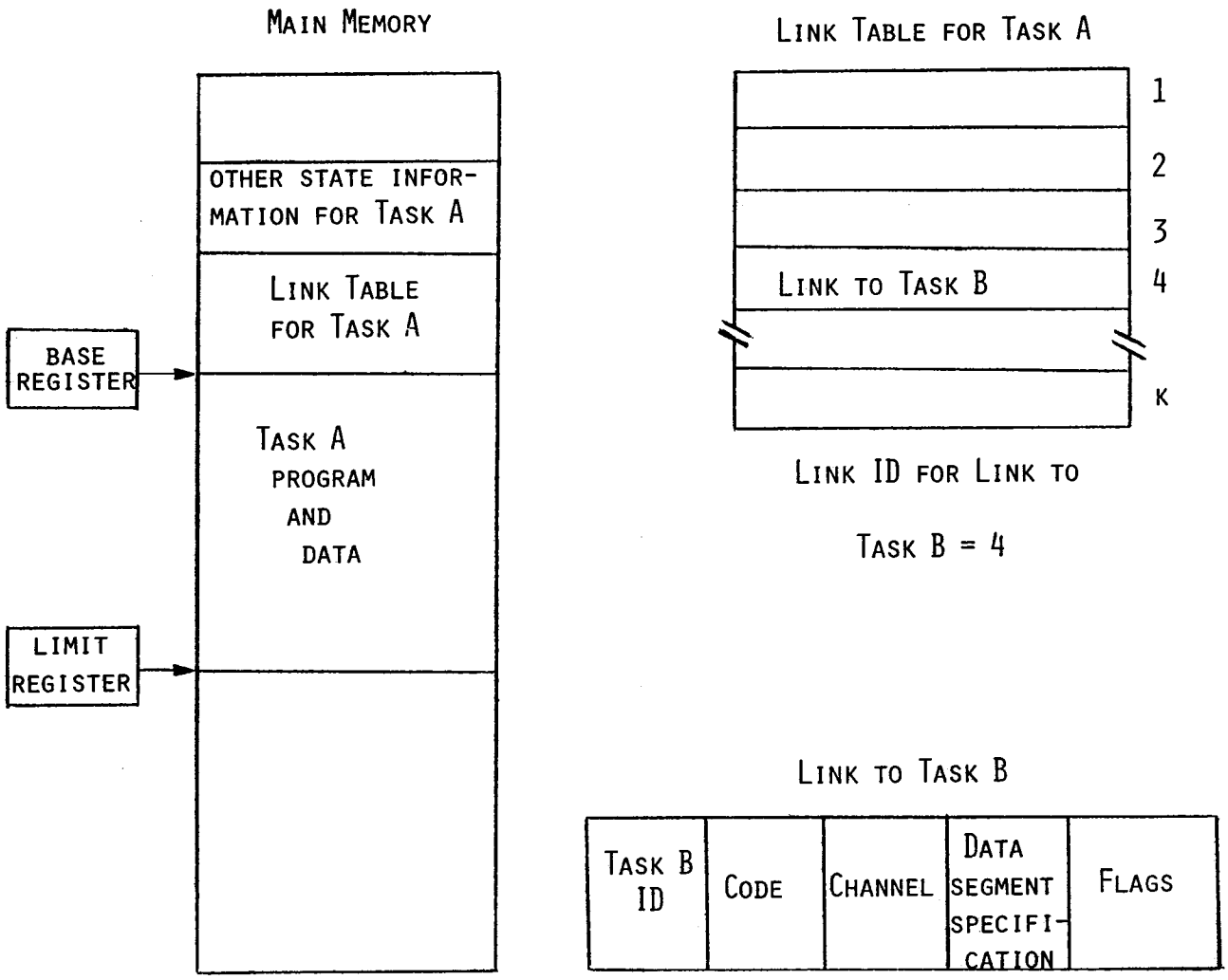

FIGURE 1

BASIC TASK AND LINK LAYOUT

A link may contain permission to move data to or from a specified area in the address space of the task which created the link. For example, when a task issues a Read request for a file, it sends a message to the file system containing the request and a link which permits the file system to move the requested data into the appropriate place in the requestor's address space. Figure 2 illustrates this example. This mechanism allows messages to be limited in size (since they are buffered by the kernel, this is desirable) while allowing the transmission of large blocks of data from one task to another efficiently in the absence of hardware supported memory sharing.

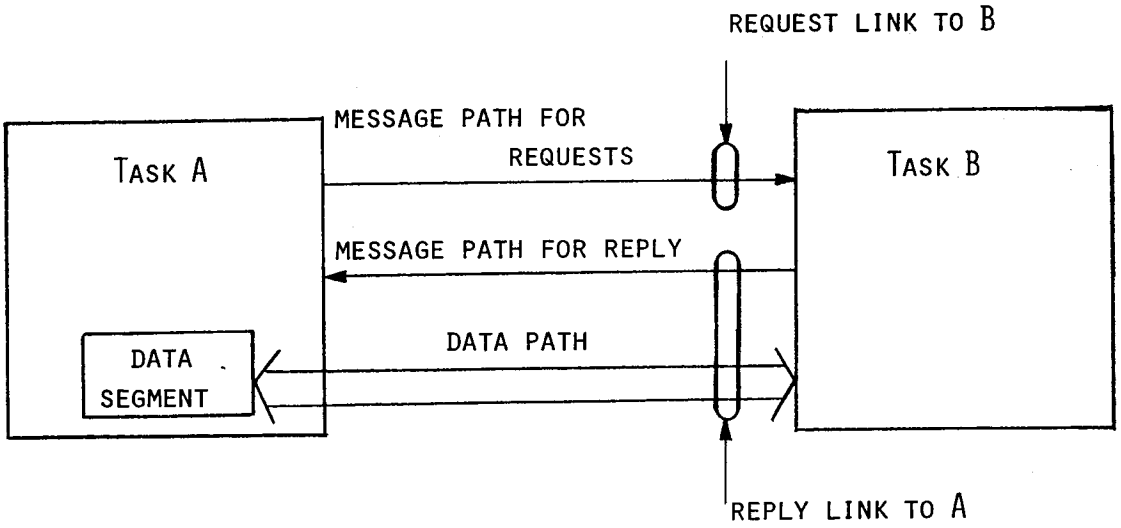

FIGURE 2

NORMAL MESSAGE & DATA SEGMENT LINK

A task may specify several attributes of each link it creates in order to identify incoming messages and to protect against unexpected or unauthorized messages. Messages are identified by means of a code which is simply a name specified by the task when it creates the link. The code from the link on which a message was sent is passed to a receiving task along with each message [Morris 73]. Permission flags in the link specify whether the task which holds the link may duplicate it, pass it to other tasks, or use it more than once.

The set of links held by a task defines its environment. A task may send messages using any link it holds, and in some cases can move data to or from the memory area of other tasks. Passing links between tasks allows programs to be divided into multiple tasks in a convenient manner which is transparent to the tasks which might receive messages from such programs. A link may be duplicated (with certain restrictions) so that a link may be passed to another task without loss of the original communication path. A message is sent to the task which created a link each time the link is duplicated or destroyed, so that an accurate count of outstanding links can be kept by the link creator. A variant of link passing allows duplicating the data area pointer of an existing link in a newly created link. This allows one task to monitor the messages of a subtask without having to move large amounts of data through intermediate buffers; the data connection but not the message connection bypasses the monitoring task. Figure 3 illustrates such a monitoring task.

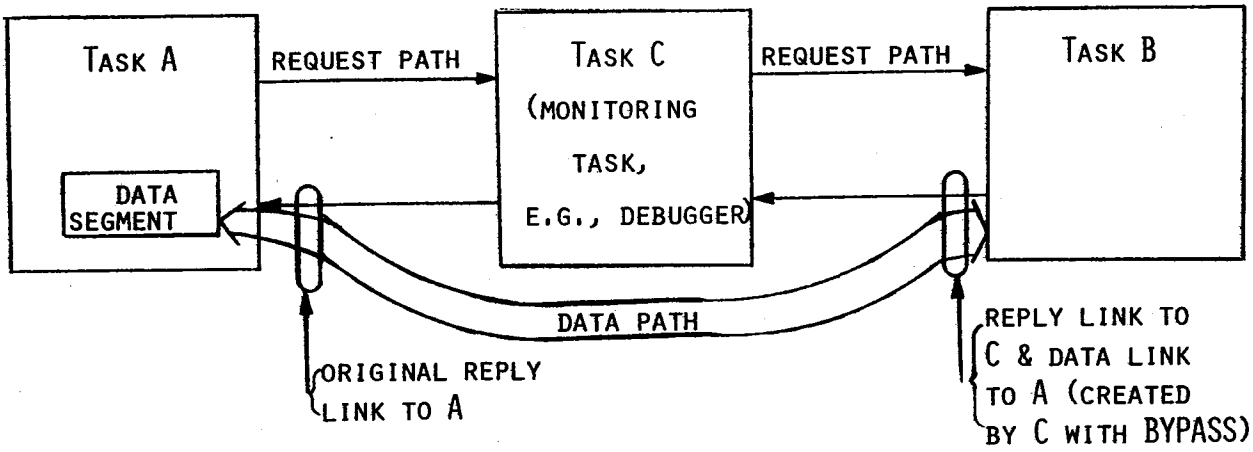

FIGURE 3

USE OF BYPASS LINKS

3. Object Descriptions

This section provides more detailed descriptions of tasks, links, messages, and channels, the primary objects defined by the DEMOS communication mechanism.

3.1. Tasks

A task consists of a program and its associated state information, including register contents, a memory area, and a link table. It has a parent task which created it, and is usually allowed to create child tasks. Task creation and termination are the responsibility of the Task Manager, discussed later. A task may be suspended, inspected, modified, and restarted by its parent. When a task terminates, it is suspended and its parent is notified if the parent has asked to be notified. Tasks which are not suspended and are not waiting for messages are given the CPU periodically.

3.2. Links

A link appearing in the link table of a task gives that task permission to send selected messages to another task. The recipient task originally created the link and specified its contents, namely, the type of operations that can be performed on the link, a code for use by the recipient, a channel, the type of links that can be passed over the link, and the address and size of a portion of the recipient's memory for direct data transfers. Links may be used to send messages and move data, may be passed along with messages, and may be created, duplicated, and destroyed.

In order to allow the kernel to check standard or common operations on links for consistancy and thus catch errors at their source, we have introduced the concept of a link type. There are four types of links: request, resource, reply, and general. Request and resource links are used for request messages, differing only in that a request link may be destroyed or duplicated by its possessor without notifying the task to which the link points. Resource links may be duplicated or destroyed by their possessor but such operations cause messages to be sent to the task to which such a link points. Resource links may not be passed to a task with a different user ID from the posessor of the link unless the posessor is also the creator. Reply links are used only for reply messages and are only used one, i.e., they are destroyed by their use. A link may be passed with a message across another link. A general link, request link, or a resource link may be passed across a reply link. Only a reply link may be passed across a request or resource link. Any type of link may be passed across a general link. No other possibilities are allowed.

3.3. Messages

Messages are small packets of information which can be transmitted across links. Messages are buffered by the kernel (up to some limit on the number of buffered messages for any one task). Messages are classified into requests, replies, and general messages depending on the type of link they are sent across.

A request message, sent across a request or resource link, is a request for some action by another task. Most request messages require replies, so a request message may be accompanied by a reply link back to the requesting task.

A reply message, sent across a reply link, is used only to reply to a previous request. The reply link is destroyed when the reply is sent. A reply may carry with it a general, request, or resource link, typically representing some abstract object being obtained in response to the original request.

General messages which use general links, give users an escape from the restrictions imposed on requests and replies. A general link can pass any kind of link, thus avoiding the asymmetry of requests and replies. With the exception of the Switchboard, no component of DEMOS uses general links.

3.4. Channels

Channels provide a way for a task to select classes of potential incoming messages. When a process creates a link, it specifies a channel on which messages associated with that link will be received. The RECEIVE operation specifies one or more channels. The oldest message whose link uses one of the specified channels will be returned to the task. If there is no such message, the task blocks until such a message arrives. Unlike link ID's which are determined by the kernel, channels are specified by the task and may be assigned in any manner desired (up to the maximum allowed number).

If a task wishes to operate in a completely synchronous manner, it should use a different channel for each outstanding link and specify only one channel in each RECEIVE. If a task wishes to operate in an asynchronous manner it can either perform conditional receive operations on channels and thus not block if no messages are available or it can ENABLE interrupts on a set of channels and be notified by the kernel when a message arrives on one of those channels.

4. Link Usage

The primary communication operations are CALL, REQUEST, REPLY, SEND, MOVE, and RECEIVE. SEND specified a message and an optional link to be passed with the message. It has two variants: DESTROY, which destroys the link used by the message, and DUPLICATE, which requests a second copy of that link. The recipient is notified if one of the variants was used unless the link is of type request. The DESTROY and DUPLICATE portions of the SEND operation are actually performed by the kernel. REQUEST sends a mes-

sage and an implicitly created reply link on a request or resource link. REPLY sends a message and an optional link on a reply link held by a task. MOVE reads or writes data through a link. RECEIVE accepts the next incoming message. CALL is a composite of REQUEST and RECEIVE and is expected to be the operation most heavily used by user programs since it combines three kernel operations (CREATE, SEND, and RECEIVE) that would otherwise commonly be done in a tight sequence. Other operations on links are CREATE and BYPASS. CREATE creates a new link to the creating task, specifying a code, a channel, restrictions on how the link may be used, and (optionally) the address and size of a segment of the creating task's memory for direct data MOVEs. BYPASS also creates such a link, but uses the data pointer from a link possessed by the creator rather than a pointer into the creator's own memory.

The following example interaction with the file system is intended to provide an intuitive understanding of these basic operations. Suppose our example task, taskA, wishes to read file FILEX. It must open the file, do some read operations, then close the file. Each of these involves a message to the file system requesting an action and a reply saying that the action is done.

To open the file, taskA performs a CALL operation on its standard link to the file system, sending a message containing the file name and other required parameters. The kernel creates a reply link, specifying the standard reply channel, and sends the message and the reply link to the file system. When the file system receives the message, it interprets it, opens FILEX, and in turn CREATEs a resource link to itself. The code field of the resource link contains the internal index into the file system's open file table entry for FILEX. The file system then does a REPLY operation on the reply link sent by taskA, passing the resource link representing FILEX back along the reply link to taskA. The completion message and the link ID for the resource link representing FILEX are returned to taskA (which becomes ready because the implicit RECEIVE operation is satisfied). The reply link is destroyed by the kernel after it is used. TaskA now has two links to the file system, its standard request link and a resource link representing FILEX. The file system has no links to taskA.

Reading data from the file requires a similar series of steps. This time, however, taskA uses the resource link representing FILEX and specifies the address and size of its buffer on the CALL operation. The file system uses MOVE to transfer the data through the data segment descriptor of the reply link. When the data transfers are complete, the file system does a REPLY operation to return status information such as the number of bytes read or whether end of file was reached. This time the file system's reply specifies no new link. Having received the reply, taskA can safely process the data in the buffer.

When taskA is done with the file, it DESTROYs its link for FILEX. The resulting message to the file system tells it to close the file. TaskA is left with only its standard link to the file system.

5. Link Management

Links resemble capabilities, so their management must take into account many of the well known difficulties of managing capabilities. This section discusses a management scheme which addresses some of these difficulties.

One problem with capabilities is the possibility that they may continue to exist after the object they point to is destroyed. In the case of links, the objects are tasks. We use the standard technique of not reusing task identifiers to solve this problem.

Sometimes it is desirable to account for outstanding links to a task. For example, the file system will need to keep track of open file links in order to close files when their last link is destroyed. We allow this by notifying the creating task whenever a resource or general link is duplicated or destroyed. A task may restrict its links so that they may not be duplicated, in which case an explicit request for another link must be made to obtain a second copy.

Yet another problem with capabilities is the lack of control over their being given away to an unauthorized third party. Classifying links into types and restricting specific operations to specific types provides a partial solution to this problem. For example, resource links may only be passed to tasks with the same user ID. Reply links, which can be used only once, reduce the problem of uncontrolled passing of links.

Perhaps the thorniest problem is that of deadlocks. In DEMOS, a deadlock is a set of tasks each of which is waiting to RECEIVE messages from other tasks in the deadlock.

While such circular wait conditions can in principle be detected, we plan at present to use a hybrid approach to deadlock control, organizing the system-defined tasks hierarchically to prevent deadlocks [Howard 73] and timing out blocked user tasks.

6. Task Management

Tasks are created and destroyed by the Task Manager, which is itself a system task accessible by a standard link. When a task wishes to create a child task, it sends a message to the Task Manager specifying the initial size of the child's memory. The Task Manager allocates the memory and creates a suspended task with no links and zero registers and memory, then it returns a reply containing a link to itself. This child link has a bypass data segment pointer which allows the parent to read and write the child's memory. Using it, the parent can read and modify the child's registers, give it links and take them back, and suspend and resume the child.

When a task terminates, it is placed in a suspended state. The parent may ask to be notified when the child terminates; upon notification it can retrieve links and status information from the child. Destroying the child link destroys the child; the Task Manager suspends the child, destroys any links it has, and frees its memory. To give a child a lifetime independent of the parent's, the parent must pass the child link to some other task, for example the Job Initiator, which is willing to adopt such children, accept their termination messages, and produce a post-mortem if they terminate abnormally.

7. Other System Facilities

Several other system tasks will exist in DEMOS to complete the task communication structure. These include the Switchboard and the timer task.

The Switchboard task is provided to allow mutually consenting arbitrary tasks to communicate. A task willing to communicate with any arbitrary task (for example, a MAIL facility) creates a general or request link, and sends a message to the Switchboard, including this link and a name. A task wishing to communicate with another named task would send a message to the Switchboard requesting a link. The Switchboard task matches pairs of tasks based on the names specified and passes the link from the other task to the requesting task. Since reply links cannot be sent on reply links, and only reply links can be sent on request links, the standard link to the Switchboard task is a general link.

The timer task provides clocks for other tasks. It accepts requests from tasks which specify a time interval (either real time or task CPU time) and at the end of the requested interval the timer task sends a reply message to the requesting task. If the receiving task has enabled interrupts on the channel it specified in the link, it will be interrupted.

8. Remarks

This communication mechanism is not pure in several ways. The data segment that can be associated with a link allows an escape from communication via messages. The general link type allows an escape from the request-reply regime. The conditional receive operation and the interrupt mechanism allow an escape from the regime of synchronous tasks with all the asynchrony of the system captured in messages. While we believe these escapes should be avoided as much as possible, we believe they are necessary for a production operating system. If we were designing our own hardware we might be more tempted by purity.

We have attempted to provide a set of primitive operations in this communication mechanism that will support a wide variety of operating system structures. We think that the operations are general and flexible but sufficiently simple to be implemented efficiently. We have consciously attempted to avoid what John Cocke has called overpowerful operators. If a task is to perform read or write operations on a file, it must send messages to the file system task. If a task is to perform suspend or resume operations on another task, it must send messages to the task manager. While these requirements may seem obvious and natural, we have seen systems where such operations were made to appear as primitives. Proving that these operations can be done efficiently without being primitives is a challenge we have ahead of us.

We see the link concept as an especially nice vehicle for intertask communication in several environments. A network of processors with private primary memory is an appealing application for this communication mechanism. The CRAY-1 with its limited form of memory protection inspired this design and is our first application.

This communication mechanism does not imply a preferred organization for the rest of the operating system. A hierarchical or layered approach is as suitable as a more distributed and independent task approach as far as this communication mechanism is concerned. The link mechanism allows the construction of abstract objects, capabilities, domains, and other protection structures but it does not require any of these.

9. Acknowledgments

We gratefully acknowledge the many helpful suggestions and remarks from J. C. Browne, David Folger, Susan Owicki, Michael Powell, and R. W. Watson.

10. References

Brinch Hansen, P. Operating System Principles. Prentice-Hall, Englewood Cliffs, N.J., 1973.

Dijkstra, E. W. Cooperating sequential processes. in Programming Languages (F. Genuys, ed.). Academic Press (1968), 43-112.

Fabry, R. S. Capability-based addressing. Communications of the ACM 17, 7 (July 1974), 403-412.

Hoare, C. A. R. Monitors: an operating system structuring concept. Communications of the ACM 17, 10 (Oct. 1974), 549-557. Corrigendum, Communications of the ACM 18, 2 (Feb. 1975), 95.

Howard. J. H. Mixed Solutions to the Deadlock Problem. Communications of the ACM 16, 7 (July 1973), 427-430.

Lampson, B. W. and Sturgis, H. E. Reflection on an Operating System Design. Communications of the ACM 19, 5 (May 1976), 251-265.

Morris, J. H. Protection in programming languages Communications of the ACM 16, 1 (Jan. 1973), 15-21.