DEMOS/MP:
THE DEVELOPMENT OF A DISTRIBUTED OPERATING SYSTEM

by

Barton P. Miller
David L. Presotto
Michael L. Powell

Computer Sciences Technical Report #650

July 1986

DEMOS/MP:
The Development of a Distributed Operating System

Abstract

The DEMOS/MP operating system has moved from a super computer with a simple addressing structure to a network of microcomputers. This transformation was done without significant changes to the semantics of the original DEMOS, i.e., existing DEMOS programs should run on DEMOS/MP.

The changes to DEMOS were simplified by the structure of its primitive objects and the functions over those objects. The structure of DEMOS links and processes were the major contributors to the simplicity. The changes made to produce DEMOS/MP involved the internal structure of link, modification to parts of the kernel, and limited changes to the various system processes.

Keywords

Distributed operating system, message-base, DEMOS, links, network.

INTRODUCTION

The DEMOS operating system started life on a Cray 1 computer and has since transitioned several computing environments. Its current home is a collection of Z8000 processors connected with a network. This distributed version of DEMOS is known as DEMOS/MP. DEMOS has successfully moved between substantially different architectures, while providing a consistent programming environment to the user. The basic DEMOS system has been described in [1] and [2]. This paper will concentrate on the details of DEMOS/MP that are relevant to DEMOS's operation in a distributed environment.

The semantics of the user interface of DEMOS/MP have not changed significantly from those of the original DEMOS system. DEMOS/MP runs on a collection of loosely coupled processors whose processor speed, memory architecture, and I/O architecture differ substantially from the Cray 1. In spite of these differences, a program that runs in the DEMOS environment will run under DEMOS/MP.

There are two goals of the DEMOS/MP project. The first goal is to provide the software base for the OSMOSIS† distributed systems project. DEMOS/MP provides a clean, message interface, and a well structured system that can be easily modified. This system is the base for various experiments in operating systems and distributed systems (such as process migration, reliable computing, and distributed program

Research was supported by the National Science Foundation grant MCS-8010686, the State of California MICRO program, and the Defense Advance Research Projects Agency (DoD) Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

† OSMOSIS = Operating System for Making Operating System Implementation Studies.

measurement).

The second goal is to experiment with the design of a distributed operating system. Our goal is to see what mechanisms would easily adapt to a distributed environment, while maintaining a high degree of network transparency.

Three areas are important in describing the development of DEMOS/MP. The first area is the collection of semantic structures in DEMOS that allow it to survive (reasonably) unscathed, the transitions in environments. The semantic structures are those that are visible to the users of the system. DEMOS uses a simple and uniform communication model. The communications model, based on *links*, provides resource naming, abstraction, and management. DEMOS processes use links as their only means of interaction.

The second area is in the changes made to the system mechanisms to reflect the new hardware environments. These mechanisms are at the lowest level of the kernel and are responsible for the hardware resource management. The two major changes for DEMOS/MP are the network and the virtual memory/addressing structure. The original DEMOS was implemented in a memory space with base and bounds registers and used swapping, whereas DEMOS/MP is implemented on a collection of machines with paged, segmented address space using demand paging. The network interface was added to DEMOS as a natural extension of the link-based communications. The goal is to make the presence of the network visible to as little of the system as possible (and not at all to the users).

The third area is in the changes to the system services (server processes) so that they function in the distributed environment. These changes may take the form of replication of a service for performance and reliability reasons.

The DEMOS/MP project involved a progression of activities leading to running in the distributed environment. It was first moved to a DEC VAX, running under simulation on the UNIX operating system. This involved the writing of a code generator for the VAX instruction set for our Model compiler [3]. Initial work on the intermachine communications was started on the VAX version of DEMOS/MP. A Z8000 code generator was also written and testing began on a network of dual processor machines. Over the next 2 years, the network subsystem, virtual memory, and process migration was added. Given the basic functioning system, experiments were started in fault tolerant computing and distributed program measurement

tools.

LINKS AND PROCESSES

The basic units from which a program is constructed are the computational elements (*processes*), and the communications paths that join the elements (*links*). Link-based communication forms the lowest level of the operating system. If the semantics of links can be distributed, then many parts of DEMOS will operate in the distributed environment without change.

Links

A link is a one way message channel to a user process, system process, or kernel. Each process has a table of links associated with it, and this table is maintained by its kernel. The link table identifies the complete set of communications paths available to a process, and thus provides a single interface to the remainder of the system. Links are protected objects and can be used as the naming mechanism for resources controlled by a centralized resource manager. In this role, links provide much of the same function as capabilities.

There are several features of DEMOS/MP links that facilitate their implementation in a distributed environment. These features are one-way communication paths, the process address structure, the DELIVERTOKERNEL attribute, and link data areas.

There are many variables in the specification of a communication mechanism, such as synchronous vs. asynchronous operations, buffered vs. unbuffered messages, and unidirectional vs. bidirectional communication paths. The decision whether to have unidirectional or bidirectional paths is a significant issue in a distributed environment.

A one-way communication path requires state information to be kept only on the sender's end of the link. This results in needing a potentially smaller amount of needed link table space. The reduction in link table space is a result of asymmetry in the client/server relationship. A client must possess a link for services which it wishes to request, but the servers only need to hold a link to a client for the period of time from when the request is received until a reply is sent. For example, every process might have a link to the name server, but the size of the name server's link table can be considerably smaller than the total number

of processes.

The interprocess communication routines (including the link table data structures) of the kernel require information only about the sender's end of a link connection. The network protocol routines must keep information about both ends of a network connection, but on a machine-to-machine basis (as opposed to a process-to-process basis).

The one-way communication paths require the creation of reply links for each request to a server. This extra performance cost is avoided by using the CALL kernel call, which is a composite of the three operations, CREATELINK, SEND, and RECEIVE.

The CALL kernel call is commonly used by both client and server processes. Most short messages are part of a synchronous exchange using the CALL. Longer data transfers are handled by a special function (see description of the MOVEDATA kernel call below).

One-way communication paths also simplify the moving (migration) of a process. When a process moves, the links that it holds require no updating, and the links held by other processes that point to the moving process can be updated the first time they are referenced after the move (see [4]).

A link can be considered as a global address to a process (in the same sense as capabilities in [5]). The global address is implemented by providing each process with a unique identifier (see Figure 1). The unique identifier only solves part of the addressing problem; it identifies the process but does not help to locate it. A "last known machine" field is included in each address within a link. This machine address is initially set to the process's current location, and when a process is moved it is updated when the next message is sent on the link.

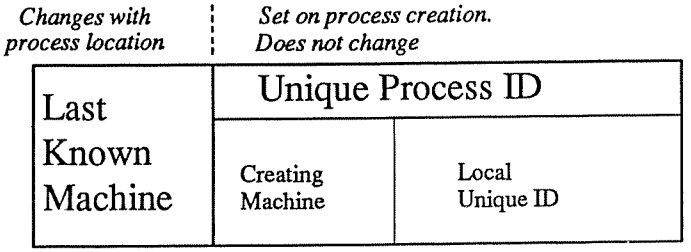

Figure 1: Link Process Address

Links are the only mechanism needed to locate a process. Communications are performed via links and control functions on a process are also performed using links. The control functions are done using

links with the DELIVERTOKERNEL attribute. Messages sent on these links use the normal message delivery mechanism, but on arrival at the destination machine, are passed to the kernel on that machine. This insures that control messages are delivered to the correct kernel, and will follow a process through its movements from machine to machine.

DEMOS/MP incorporates the idea of an intermachine address and intermachine communications at the lowest level of the kernel. This allows the operating system, above the communications level, to ignore machine boundaries when desired. Ignoring machine boundaries makes the implementation of functions such as remote paging (described in the Virtual Memory section) simple. The DEMOS/MP design is in contrast to the Accent [6] system, which provides only local naming in the kernel. Remote messages must be interpreted by an agent (process) outside the kernel.

When a process creates a link (pointing to itself), it can specify an area of its memory which may be read or written by the holder of the link. Data is transferred to or from a link data area by use of the MOVEDATA kernel call.

Link data areas provide a potential performance optimization for both local and remote large data transfers. The local data transfers can be done quickly by mapping a part of a process's address space into that of another process. Given a paged virtual memory architecture and the alignment of the link data area on a page boundary, a read transfer will be done by mapping the the memory page(s) into the reading process's address space. If either process then tries to modify the page, a copy is then made. This copy-on-write idea was used in the TENEX [7] operating system. The page mapping is an implementation optimization which has no change on the DEMOS/MP semantics.

Remote transfers of large data blocks can also be done more efficiently using the link data areas. A special protocol is used that does not need intermediate acknowledgments, transfers data in large blocks, and has no restrictions on the order of delivery of the sections of the data area being transferred[†]. The V kernel [8] uses data segments associated with their fixed length messages to achieve this same effect.

Message SEND and RECEIVE are used, both locally and remotely, for control functions (for example,

† Each section of data area is labeled with its offset into the data area. The sections arrive and are put into the proper place.

initiating a file read operation), and MOVEDATA is used for large data transfers (such as copying the data being read from the file).

Processes

A DEMOS/MP process is encapsulated by its link table. A process's abilities are determined by the links that it holds. There are no distinctions between "system" or "privileged" processes and standard user processes other than the links that each process possesses. A process's privileges are determined only by the links that it holds.

DEMOS/MP also supports a form of lightweight process within the kernels (called *kernel processes*). These processes lie entirely within a kernel's address space and have a minimum of state. The kernel processes are used for the long term or asynchronous activities performed by the DEMOS/MP kernels, such as large data transfers, device interfacing, and process migration.

Kernel processes remain dormant until a message is received by the kernel. There can be a kernel process for each of the kernel's message channels. Kernel processes have no state other than the static variables maintained in the kernel. A kernel process determines its functions by the contents of the message and the state of selected variables. Kernel processes can be thought of as transactions that are performed by the kernel in response to messages.

NETWORK

Messages sent over links in the original DEMOS obey the following properties:

- messages sent from one process to another arrive in the same order as they are sent.
- no messages are lost.
- no messages are duplicated.

We extended DEMOS to work across a network environment without changing the communication semantics. Therefore, we had to maintain the above properties while adding one more:

- links may address processes on other machines.

This section describes the changes to the original DEMOS and the network protocols used to support the

extension.

Extensions to DEMOS

Associated with each process, in single processor DEMOS, is a unique identifier. In DEMOS/MP, this identifier is made unique, network wide, by appending to it the unique ID of the machine on which the process is created. A process retains this identifier, even if it should migrate to other machines. Process ID's are not visible to processes but exist within the links which processes use to name other processes. Since these links are manipulated only by the message kernel, the processes do not have to be changed to support the new name space.

Network wide process names are sufficient to identify the target of a message. However, because processes can migrate across machines, these names may not describe the location of the target process. Therefore, the "last known machine" field, was added to the process ID contained in a link. This field is used as a hint when routing messages. If the target process is not on the same machine as the sender, the hint field is used as the destination machine. The only time that the last known machine field changes is when a process migrates. Processes do not migrate frequently or else thrashing could easily result. After a process has migrated, the first message sent to it by a remote process causes an update sent back to the sender informing it of the migrated process's new location. This updates all the last known machine fields on the sender's machine. The result is a slight delay in one or two messages. A complete description of the method for updating this field is described in [4].

Network Protocol

We implemented a special purpose, light weight protocol based on the DEMOS model of interprocess communication. This is similar to the approach taken by the Locus distributed operating system[9]. The protocol derives much of its simplicity from the use of type abstractions in the Model programming language [3] and abstractions take the place of what is normally thought of as layers in network protocols. The interface between abstractions is the procedure call. Data passes between abstractions using the standard Model parameter passing. This method avoids the wrapping and unwrapping of data with layers of protocol header.

The three abstractions that are used to interface network media to DEMOS/MP are

- physical device abstractions,
- network FSM abstractions, and
- remote processor object abstractions.

Figure 2 illustrates these abstractions.

Each abstraction may have many instances. In this paper we call these instances objects. It should be noted that an object is passive, that is, its actions are initiated by 'calls' made by an executing process to the functions which the abstraction *exports*.

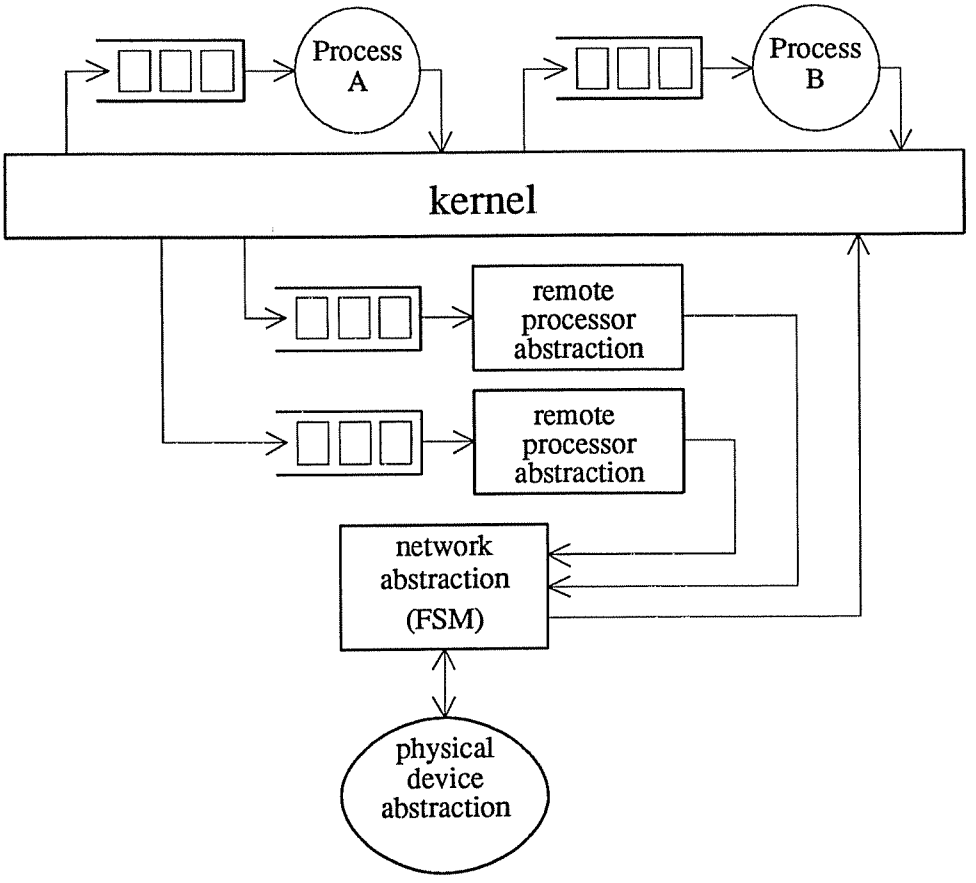

Figure 2: DEMOS/MP Message Paths in the Network

The device abstraction hides the idiosyncrasies of I/O devices from the rest of the system by providing a uniform interface to all devices. A device object exists for each device attached to a DEMOS/MP processor. The functions exported by the device abstraction are write a block, read a block, return device status, and return a list of pending interrupts.

Device abstractions currently exist for 8-bit parallel interfaces (Z8000 to Z8000), RS232 serial lines (19200 baud to VAX host machines), and a simulated acknowledging ethernet.

The network finite state machine (NFSM) abstraction adds network semantics to a physical device. An NFSM exists for each device used for network communications. Each NFSM has associated with it a circular list of remote processor objects. This list represents the processors which are accessible via that interface.

The NFSM is really two finite state machines, one for input and one for output. The input FSM reads blocks from the devices. It assembles these blocks into network messages, and calls the appropriate remote processor object to act on any uncorrupted network messages. The output FSM calls each remote processor object on its list for a message to send and, if one exists, writes it to the device.

NFSM transitions are effected by calls from processes to a NFSM object (instance of an NFSM abstraction). The kernel process continuously scans the devices and clock for interrupts and calls the NFSM objects to notify them of these events. The remote processor objects call the NFSM objects to notify them of network messages waiting to be sent.

The remote processor (RP) abstraction guarantees the invariance of DEMOS communication semantics across the network. A remote processor object exists in each processor for all other processors in the network. Whenever the message kernel encounters a message destined for a remote processor, it calls an RP object to pass the message. The RP uses and end-to-end acknowledged window protocol to ensure the ordered delivery of messages at the destination machine.

Reconfiguration and Failure

The only binding between network NFSM objects and RP objects consists of the list of RP objects held by the NFSM's. In the event of failure or network reconfiguration, these lists are updated by a kernel network process. This rebinding is completely transparent to the conversations between RP objects because the protocol between RP objects is end-to-end acknowledged. Therefore, given enough redundancy in the connectivity of the network (i.e., as long as there is always at least one path between the message source and destination), DEMOS/MP communications will survive any network failure without

human intervention. Node failures are described below in the section on Publishing.

Performance Enhancement

As a performance enhancement, we identified two types of messages, guaranteed and unguaranteed. Unguaranteed messages are timely messages which it would be pointless to resend or are messages containing information which is inherently redundant (such as message traffic statistics or routing information). These messages form a moderately large percentage of our normal message traffic. Such messages bypass all the checks which guarantee the properties of DEMOS messages (except for corruption detection) and are handed directly to the target processes. In general, these messages correspond to the messages which are broadcast to all processors in an Ethernet-based system such as routing information and time of day.

VIRTUAL MEMORY

The first implementation of the DEMOS virtual memory used base and bounds registers and swapping. This simple design was driven largely by the Cray 1 architecture. DEMOS/MP was modified to support more complex memory management architectures making use of segmentation, paging, and network communications.

Machine Architecture

DEMOS/MP runs on a dual processor system using the Zilog Z8000 processor running at 2 megahertz (see Figure 3). This system supports a virtual address space of 128 64K-byte segments. One processor runs the DEMOS/MP kernel (and access the memory using 24 bit unmapped addresses) and the other processor runs DEMOS/MP processes.

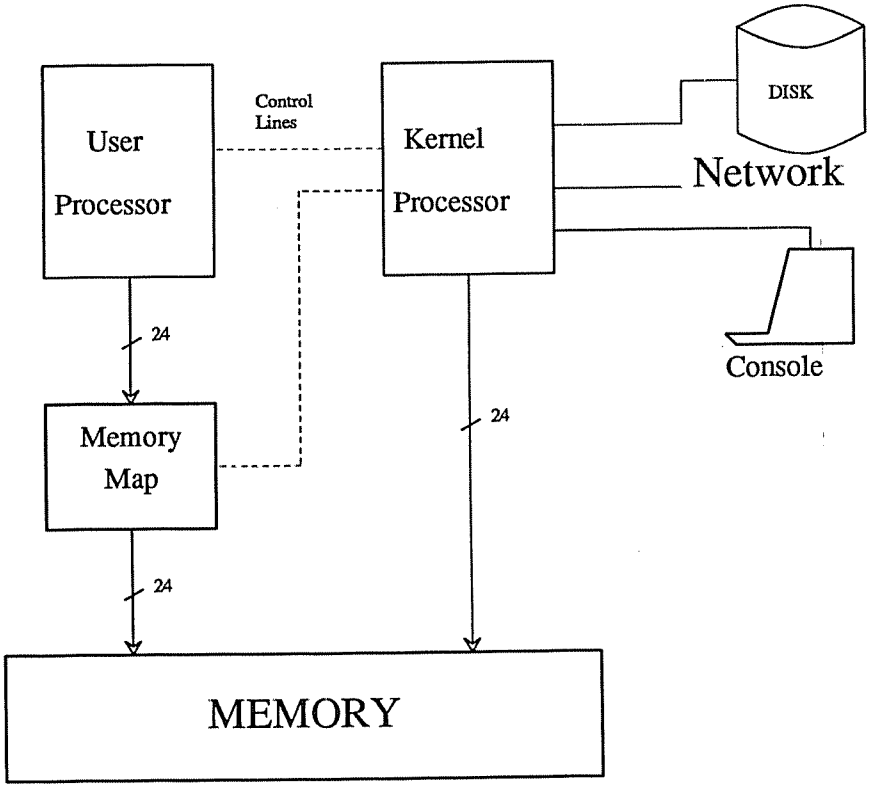

Figure 3: Dual-processor Architecture

The dual processor system is used because the Z8000 CPU cannot recover from a page fault. Therefore, when the user processor causes a fault, its activity is suspended until the kernel processor services the fault and allows the user processor to continue execution. The kernel processor accesses memory directly and is assumed to never have an address fault (this would indicate incorrect operation of the system). This design was first described in [10]. A complete description of the handshake between the processors is described in [11].

The address translation maps (see Figure 4) for the user processor are all contained in special hardware registers. These registers contain the segment maps for 8 processes and page maps for 16 segments. These maps can be set only by the kernel processor. The memory maps can be considered a scarce resource (i.e, there are more processes executing than there are maps to support them) so the maps are allocated according to an LRU policy. The translation hardware does not contain either reference or dirty bits, thus restricting the page replacement algorithms that can be easily implemented.

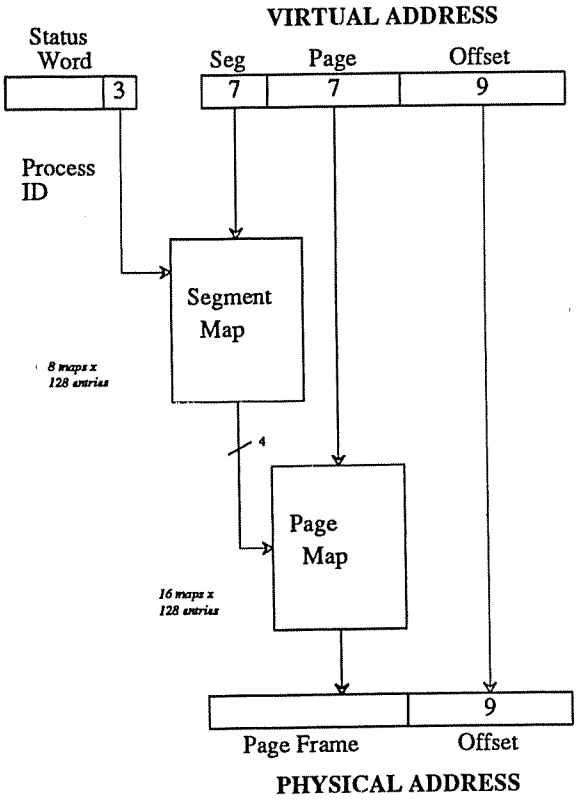

Figure 4: Address Translation Maps

The dual processor configuration has some performance advantages and disadvantages. The advantages come from the ability of the kernel and user processors to compute independently. While the user processor is running, the kernel processor can be performing I/O operations, and doing network communications. If the machines are using store-and-forward communications, the receipt of a through packet need not disturb the user's computation.

The disadvantage of the DEMOS/MP dual architecture is that, during a page fault, activity on the user processor is suspended. This means that even if there are other runable processes, they cannot be run. Our simulation studies of the DEMOS/MP system have shown (given current CPU, memory, and disk speeds) that typical utilization of the the user processor on a heavily loaded system is about 65-68%. A faster CPU or slower paging devices (such as remote paging) might substantially reduce this value. Fortunately, the microprocessor industry is now producing processors that can recover from page faults.

Page Fault Handling

The page fault handler within the DEMOS/MP kernel uses links and link data areas to transfer pages to and from the disk driver. To service a page fault, the kernel creates a link whose data area points to the page that is to be read from or written to the disk. The data area address is in the kernel's address space and therefore is a physical memory address. The size of the data area is the page size. The actual transfer is done by requesting service from the disk driver, which would performs the MOVEDATA operation.

Given that page transfers use the link data areas, and that the MOVEDATA function works across machine boundaries, remote demand paging is a simple addition. This permits DEMOS/MP to have multiple machines sharing a paging device (disk). The location of the paging area for a machine is established at the time that machine is booted. Remote paging was implemented because not all the Z8000 machines had disks.

SYSTEM TASKS

DEMOS/MP uses system server processes for process management, memory management, file services, clock services, and naming and connection services. The initial version of DEMOS/MP required almost no changes to the system processes to be operational. The one significant change is that the low-level process manager needs to communicate with multiple kernels.

In this section, we discuss the affect of the multiple machine environment on the switchboard and its resources, process and memory management, and the file system.

Resource Naming and the Switchboard

DEMOS/MP provides a two level version of the flat name space provided by the original DEMOS. In DEMOS, a single name server, the switchboard, aids in establishing connections between processes in much the same way a telephone operator sets up a call between two people. Any process with a link to the switchboard may either announce itself to the world or request a link to another process which has already announced itself. To announce itself, a process sends a message to the switchboard containing a link to itself and a name to assign to the link. When another process sends a request message to the Switchboard containing that name, the switchboard returns the link to the announced process. Since the switchboard is

one of the first processes started, it is simple to provide a link to the switchboard to each new process that needs it.

There are two types of switchboard processes, global and local. There is only one global switchboard. Its location is broadcast across the network as part of the network's routing protocol. As new processors are added to the network, they are told of the switchboard's location. Local switchboards may exist on each processor in the network. Their locations are are not broadcast.

When a process sends a message using a switchboard link, the message kernel sends that message to the local switchboard if one exists or to the global switchboard if there is no local one. Switchboard requests from the local switchboards are sent to the global switchboard. This effectively implements a two level naming hierarchy.

The hierarchical design solves two problems, scaling the name space and locating the switchboard. The scaling problem is caused by the potentially larger number of names and resources in a multiple machine environment. The locating problem is caused by not having a switchboard process on every machine.

The hierarchy was especially useful in the implementation of Publishing (described in a later section). In this experiment it was necessary to create a separate instance of all system programs on the recording node so that a recording node could act independently from the rest of the system. This was done by giving it its own local switchboard which was an exact copy of the standard global switchboard.

Process and Memory Management

The DEMOS/MP memory manager performs the low-level memory policy and process management decisions. The memory manager has a link to the process and memory kernel process in each DEMOS/MP kernel. At this level, the existence of multiple machines is visible.

Each time a new machine is booted, it posts a link to the switchboard. The memory manager always has an outstanding request to the switchboard for these links, and in this way becomes aware of machines entering or re-entering the network. Multiple memory managers can exist by partitioning the machines, with one memory manager to each partition. These partitions correspond to the allocation of switchboards.

High-level process management functions are performed in the process manager process. Each process manager can talk with one or more memory manager processes. The number of memory managers to which a process manager talks is a function of the number of machines over which it can control, the replication level for fault tolerance, and administrative divisions.

File System

The DEMOS/MP file system ran with little change from the DEMOS version. The four file system processes can be distributed over many machines. For performance reasons, the buffer manager and disk interface processes are typically on the same machine as the device which they control. Multiple disk interface/buffer manager pairs can be used if there are multiple disks connected to different machines. It is possible, though, to have all disks controlled from a single pair.

PERFORMANCE

While performance is not a major design issue for DEMOS/MP, it is important to report some measure of the system's performance. These results help in evaluating both the hardware and the software.

Operation	**Cost**	
Null kernel call	1 ms	
Create/destroy link	3 ms	
Disk block read or write (512 bytes)	110 ms	
Operation	**Local Cost**	**Remote Cost**
Send/receive 0-byte message	10 ms	63 ms
Send/receive 1024-byte message	25 ms	280 ms

Table 1: Basic Performance Measures

Table 1 summarizes basic performance results from the Z8000 DEMOS/MP implementation. The first result is the time for a null kernel call. This is the elapsed time from the user's request to the kernel (trap instruction) until the kernel returns control to the user process. This interval includes only the time to enter and exit the kernel. Most of the 1 ms goes to the synchronization between the User and Kernel processors. Part of this synchronization time is due to at least three machine interrupts: 1) the kernel call trap instruction, 2) User-to-Kernel processor interrupt, and 3) Kernel-to-User processor interrupt.

The second result in Table 1 is the cost of a pair of simple kernel calls: create and destroy link. This

result reflects the cost of the two calls to the kernel and the associated time to create and destroy the link. The cost is dominated by the User-Kernel processor synchronization time.

The third result in Table 1 is the cost of reading or writing a 512 byte disk block. This is the time needed for the kernel processor to access and transfer the data to or from the disk. The disk read/write time is controlled by the speed of the parallel I/O interface (used both for disk I/O and network connections). This interface has a maximum speed of 5K bytes/second (40K bits/second).

The last two results in Table 1 describe the communication performance of DEMOS/MP. These results show the cost of send/receive pairs of kernel calls. Local times were measured by a process repeatedly sending a message to itself and then receiving it. The process was run alone on a machine and the elapsed time was measured for many iterations. Remote times were measured by a process on one machine sending a message to a process on a second machine and then having the message returned. Again, these processes were run alone on their respective machines and the elapsed time was measured for many iterations. The 10 ms for the local cost of sending a null message reflects the general non-network overhead of transmitting a message. The cost of a remote message is dominated by the data transfer time over the parallel interface.

There are several points that determine the DEMOS/MP performance. First, the Z8000 processors run on a 2 megahertz clock, compared to processors today running at more than 10 megahertz. Second, the Kernel-User processor synchronization is needed because a separate processor is required to handle page faults. The interprocessor handshaking increases the time to enter and exit the kernel and increases the time to process kernel calls and page faults. Third, the parallel interfaces operates at 40K bit/second, compared to current 10 megabit/second ethernets. The speed of the parallel interfaces affects both the communication and I/O performance. Last, no special effort was made to tune the DEMOS/MP kernel performance. Our informal code inspections identified several areas where such optimizations as in-line procedure expansion might substantially improve performance.

OTHER AREAS

Publishing: Distributed Recovery

DEMOS/MP was used to test a new idea in process recovery, Publishing. Publishing[12] is a model for crash recovery in a distributed computing environment. It provides a recovery mechanism which can be completely transparent to the failed process and all processes interacting with it. Publishing is intended for message based systems, preferably those using a centralized communications medium such as a bus, ring, or Ethernet.

A centralized *recorder* reliably stores all messages that are transmitted, as well as checkpoint and recovery information. When it detects a failure, the recorder may restart affected processes from checkpoints. The recorder subsequently resends to the process all messages which were sent to it since the time the checkpoint was taken, while ignoring duplicate messages sent by it.

DEMOS/MP provided an excellent environment to test Publishing. First, because the state of processes in DEMOS/MP is well defined, checkpointing and restarting of processes is a simple operation. Second, unlike many other systems which support message passing, all communication between processes in DEMOS/MP is message based. Even requests to the kernel (other than those necessary for sending and receiving messages) are messages. Therefore, all inter-process communications can be 'captured' by the recorder.

The prototype version implemented in DEMOS/MP demonstrates that an error recovery can be transparent to the user processes and can be centralized in the network.

Distributed Program Performance Tools

The DEMOS/MP operating system provides a test bed for a set of distributed program performance tools. The message semantics of DEMOS/MP provides a natural environment for the construction of multiple process programs (which can be spread across several machines). Performance tools designed to measure such things as parallelism, program structure, and communications levels are described in [13, 14].

CONCLUSION

The success of the DEMOS/MP implementation is due to the simplicity of the organization of the system. DEMOS/MP uses message passing as the basic structuring tool for both the operating system and

the users. Message passing is implemented at the lowest level of the system and therefore is available for use by both the system and users.

Several features of the DEMOS/MP design prove noteworthy. The first feature is the link data area. At first glance, these data areas appear to be an escape mechanism to allow processes to share memory. These are better viewed as separating the two common forms of communications: short control messages and bulk data transfers. Messages are used for short, control functions, and link data areas are used for large data transfers. The data areas contribute to the ease in implementing process migration and remote paging.

The second feature is the use of DELIVERTOKERNEL links. These links provide control of processes across machine boundaries, while requiring a minimum of additional mechanism. In all cases, control messages for a process are directed to the correct machine by using the standard message routing.

The third feature is the structure of the intermachine protocols. Through the use of type abstractions in a strongly typed language, layers of protocol were constructed with small performance penalty for interactions between the abstractions (layers) of the protocol.

It is not difficult to design an operating system that will run in a distributed environment. The main point is to avoid uncontrolled sharing of data between components of the operating system. In DEMOS/MP, each component (kernel, system processes) of the operating system only maintains state that is local to that component and uses links as the only means to reference state that is maintained by another component. For example, when a process opens a file, it receives a link that represents the resource of the open file. All of the state regarding the state of the open file is maintained in the file server process(es). If a user process moves, the only requirement is that the link still be valid at the process's new location; no other changes are needed to local system state.

It is not difficult to distribute an operating system, but to make a reliable distributed systems is difficult. Even given a reliable network communication protocol, a system must still handle machine crashes, network failures, and the resulting loss of system function. It is not difficult to place different parts of the file service on different machines, but this placement increases the probability of losing part of the file service. The design of a service that can survive failures is difficult. Our solution is to provide a system-wide

recovery mechanism -- publishing -- that all process can use. This solves the recovery problem, but does not help the system to continue (if possible) during the failure.

References

[1] F. Baskett, J. H. Howard, and J. T. Montague, "Task Communcations in DEMOS," *Proc. of the Sixth Symp. on Operating Sys. Principles*, Purdue, (November 1977).

[2] M. L. Powell, "The DEMOS File System," *Proc. of the Sixth Symp. on Operating Sys. Principles*, pp. 33-42 Purdue University, (November 1977).

[3] J. B. Morris, *A Manual for the Model Programming Language*, Los Alamos Scientific Laboratory, Los Alamos, New Mexico (February 1980).

[4] M. L. Powell and B. P. Miller, "Process Migration in DEMOS/MP," *Proc. of the Ninth Symp. on Operating Sys. Principles*, pp. 110-119 Bretton Woods, N.H., (October 1983).

[5] Robert S. Fabry, "Capability-Based Addressing," *Communications of the ACM* **17**(7) pp. 403-412 (July 1974).

[6] R. F. Rashid and G. G. Robertson, "Accent: A communication oriented network operating system kernel," *Proc. of the Eighth Symp. on Operating Sys. Principles*, pp. 64-75 Asilomar, Calif., (December 1981).

[7] Daniel G. Bobrow, Jerry D. Burchfiel, Daniel L. Murphy, and Raymond S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," *Communications of the ACM* **15**(3) pp. 135-143 (March 1972).

[8] D. R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations," *Proc. of the Ninth Symp. on Operating Sys. Principles*, pp. 128-139 Bretton Woods, N.H., (October 1983).

[9] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel, "The LOCUS Distributed Operating System," *Proceedings of the 9th SOSP, Operating Systems Review* **17**(5) pp. 49-70 (November 1983).

[10] Forest Baskett, "Pascal and Virtual Memory in a Z8000 or MC68000 based Design Station," *Proc. of Spring COMPCON*, pp. 456-459 (1980).

[11] Frederick H. Carter, "The OSMOSIS Project: A Control System for the Dual Processor Architecture," M.S. Report, University of California, Berkeley (December 1981).

[12] M. L. Powell and D. L. Presotto, "PUBLISHING: A Reliable Broadcast Communication Mechanism," *Proc. of the Ninth Symp. on Operating Sys. Principles*, pp. 100-109 Bretton Woods, N.H., (October 1983).

[13] B. P. Miller, "Performance Characterization of Distributed Programs," Ph.D. Dissertation, Technical Report UCB/CSD 84/197, University of California, Berkeley (May 1984).

[14]

B. P. Miller, S. Sechrest, and C. Macrander, "A Distributed Program Monitor for Berkeley Unix," *Software - Practice & Experience* **16**(2) pp. 183-200 (February 1986). Also appears in short form in the Proc. of the 5th Int'l Conf. on Distributed Computing Systems, Denver (May 1985).