MACH: A Basis for Future UNIX Development

Avadis Tevanian, Jr.
Richard F. Rashid
June 1987
CMU-CS-87-139 (2)

ABSTRACT

Computing in the future will be supported by distributed computing environments. These environments will consist of a wide range of hardware architectures in both the uniprocessor and multiprocessor domain. This paper discusses Mach, an operating system under development at Carnegie Mellon University, that has been designed with the intent to integrate both distributed and multiprocessor functionality. In addition, Mach provides the foundation upon which future Unix development may take place in these new environments.

This research was sponsored by the Defense Advanced Projects Research Agency (DOD), ARPA Order No. 4864, monitored by the Space and Naval Warfare Systems Command under contract N00039-85-C-1034. Unix is a trademark of AT&T Bell Laboratories.

MACH: A Basis for Future UNIX Development

Avadis Tevanian, Jr.
Richard F. Rashid

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

1. Introduction

Computing environments used by researchers have been (and will continue) undergoing many changes. A typical installation now consists of several separate hosts connected by high speed networks, forming a local area network. Commercial multiprocessors are becoming increasingly available. Perhaps more importantly, multiprocessors are starting to be accepted as viable computing tools by more and more researchers.

At the same time, Unix has gained popularity, and has become the operating system of choice for many. Unfortunately, Unix was not designed (or implemented) as a distributed system nor as a multiprocessor system. This has resulted in many attempts to add either distributed or multiprocessor functionality to existing Unix implementations. In addition, the Unix kernel has become a dumping ground for features that are not conveniently implemented outside of the kernel.

Mach, on the other hand, has been designed with a goal of creating integrated computing environments, consisting of networks of uniprocessors and multiprocessors. At the heart of Mach, is a simple, extensible kernel. The basic functionality of the kernel is designed to support the integrated computing environment of the future. The Mach kernel design can be thought of as a return to the old, seemingly forgotten Unix philosophy: simplicity.

Recognizing the utility of existing Unix systems, it is clear than a system for future environments should be somewhat compatible with existing Unix systems. With this in mind, the Mach implementation has always been binary compatible with 4.3BSD Unix. The basic Mach primitives will allow Unix functionality to be implemented outside of the Mach kernel.

2. Basic Mach Primitives

The basic functions provided by the Mach kernel have been designed with the following host hardware configurations in mind:

- Typical uniprocessors.
- Tightly coupled shared memory multiprocessors.
- Loosely coupled multiprocessors with limited, or differential access shared memory.

To effectively utilize such machines, Mach provides the following features:

- Separation of typical *process* abstraction into a *task* and *thread*. A Unix process is effectively a single thread running within a task.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4864, monitored by the Space and Naval Warfare Systems Command under contract N00039-85-C-1034.
Unix is a trademark of AT&T Bell Laboratories.
Although future implementations may not be binary compatible (due to implementation constraints), the goal is to be at least source compatible.

- Powerful virtual memory primitives, allowing sharing via in inheritance mechanism with copy-on-write implementation.
- A communication mechanism that is *transparently* extendible over a network.
- Unix compatibility.

Highly parallel applications execute efficiently on a tightly coupled multiprocessor using multiple threads of execution within a single task. Other parallel applications may execute efficiently on a loosely coupled multiprocessor by specifying only those regions which need be shared and executing as multiple tasks. Finally, a distributed program may execute efficiently in a network of uniprocessors and multiprocessors using an extensible inter-task communication mechanism.

3. Tasks and Threads

The idea of using multiple threads of control within a single process is not new. Many programmers have created coroutine packages that do just this. Allowing multiple threads of control within a single process is convenient for writing server applications. In addition, multiple threads is an attractive mechanism for utilizing the parallelism afforded by tightly-coupled shared memory multiprocessors. Unfortunately, a coroutine package can not simultaneously schedule a single process on multiple processors. For this reason, the Mach kernel separates the typical process abstraction into a task and a thread abstraction.

A task is an address space and a collection of system resources. In the Unix domain, a task can be thought of as its address space, file descriptors, resource usage information, etc. In essence, a task is a process without a flow of control or register set (hardware state).

A thread is the basic unit of execution. A thread executes in the context of exactly one task. However, any number of threads may execute within a single task. All threads theoretically execute in parallel. So, when running on a multiprocessor, multiple threads may indeed execute in parallel.

4. Virtual Memory

Mach provides a very aggressive virtual memory design and implementation. Each task contains a large, potentially sparse virtual address space. Any thread with access to a task may cause virtual memory to be allocated or deallocated. In addition, flexible protection primitives and sharing are provided for.

It is generally agreed that programming using a shared memory paradigm is bad style. This is indeed the Mach philosophy. However, for certain specialized applications which intend to use multiprocessing for computational horsepower, it is clear that some form of shared memory must be provided. Although the task and thread mechanism does provide a means for many computational entities to share memory, it does have its pitfalls:

- Not all multiprocessors allow for efficient parallel execution of threads within the same task on multiple processors. This is a potentially serious problem on multiprocessors that do not provide cache consistency on shared memory. On such machines, it may be necessary to make an entire task's writable address space uncached.
- Threads within a task are completely unprotected from each other. All threads within a task have access to all of that task's resources. Some applications require some amount of protection between its computing entities.

Mach provides the ability for tasks to specify that regions of an address space should be shared by both the parent and child after a fork operation. Thus, a specialized application seeking to use shared memory for performance reasons may create a tree of tasks sharing memory.

At first glance, the ability to share via an inheritance mechanism seems strict. This is actually a feature. By restricting sharing to an inheritance mechanism, it can be guaranteed that shared memory

---

In fact, the VM implementation can easily support other interfaces to specify sharing, but such interfaces are intentionally not provided.

will never cross a network boundary. This is important if network transparency of operations is to be achieved. A mechanism that arbitrarily allows sharing between tasks would allow tasks on separate hosts to share memory. While inter-host shared memory could be implemented, there are serious problems such a data representation. That is, how does one implement shared memory between two different types of processors?

5. IPC

Interprocess communication in Unix has never been flexible enough to easily build distributed systems. While advanced versions of Unix, such as 4.3 BSD, continue to add communication mechanisms, the problems real distributed systems must address are glossed over. For example, Internet domain sockets use a global machine specific naming convention based on IP address, with lack of location independence and protection.

To address the problems associated with building distributed systems, Mach provides a flexible interprocess communication facility. To facilitate building distributed systems, Mach stresses:

- A capability based interprocess communication paradigm.
- Messages containing typed data.
- Transparent extension of local communication into a network by network message servers.
- An interface language, MatchMaker, which is used to generate client/server interfaces.
- Integration with virtual memory management for efficient transfer of large messages.

The *port* is the basic communication abstraction provided by Mach. A port is effectively a queue. A *send* operation adds a message to the queue; a *receive* operation removes a message from the queue. A port is a kernel protected object. With each port is associated send and receive rights. To send a message to a port, a task must have send rights. To receive a message from a port, a task must have receive rights. Ports may have any number of senders but only one receiver.

The Mach kernel itself has no knowledge of networks. As far as the kernel is concerned, messages are always passed between tasks on the same host. Network message servers transparently extend communication over a network. A message sent to a port on a remote machine actually is sent to a network server which then forwards the message over the network. The forwarding operation is transparent (and undetectible) to both the sender and the receiver.

In addition to simply extending the IPC paradigm to the network, network servers may participate in data type conversion and provide secure network transmission. By providing this functionality outside of the kernel, Mach allows a host more flexibility in choosing data type representations, the amount or type of security to be used on a network and even the protocols to use for network transmission.

Creating distributed programs is aided by MatchMaker, an interprocess specification language. MatchMaker allows a program to specify an interface between a client and server. It then takes this specification and generates stubs to be used by the client and server. MatchMaker allows a programmer to create a distributed program without worrying about the details of sending messages or type conversion between different machines.

Finally, the IPC mechanism makes use of the virtual memory system to perform virtual copies of large messages rather than physical copies. This allows large amounts of data to be sent copy-on-write. This is an especially important feature since data received by a task is usually only read, requiring no copy.

6. Portability Issues

Unix has always been touted as being a portable system. The features provided by a typical Unix kernel are indeed quite portable. However, it is certainly arguable just how portable the kernel itself is. While it is true that a "Unix port" can usually be done very quickly, this is usually because the port is to yet another Vax processor or to a machine similar to a Vax. It is not unusual for a port of 4.2 BSD

Vax is a trademark of Digital Equipment Corporation.

to simulate the Vax page tables. The machine independent virtual memory implementation of 4.3 BSD is indeed Vax page table management.

Since a goal of Mach is to run on a wide variety of machines, it is very important to be sure that the implementation is truly portable. Much attention has been given to this issue. The issues of portability are especially stressed in the area of virtual memory. Every vendor seems to feel the need to built a new, proprietary memory management unit. For this reason the virtual memory implementation has been carefully divided into machine independent and dependent sections. For further details on the implementation, see the references.

7. Summary

Mach is the new kernel foundation that will allow Unix to survive in the future. Current Unix kernel implementations, filled with hack after hack, have become unwieldy. Mach represents a simple kernel design that can support computing environments of the future while still supporting a Unix style software environment.

The implementation of Mach is well underway. The virtual memory and IPC implementation is complete. The separate task and thread mechanism is in progress, with much of the ground work being completed. Performance of the system exceeds that of 4.3 due to the new virtual memory implementation.

The current implementation runs on virtually all Vax processors, including 750, 780, 785, 8200, 8600, 8650, and MicroVax I & II. Mach also runs on the IBM RT/PC. A port to the Sun-3 is in progress. Several multiprocessors are supported by Mach, including:

- Vax 784 - A machine consisting of 4 Vax 780 processors connected with 8 megabytes. A 789 has 785 processors.
- Vax 8300 - Dual processor 8200. Mach actually supports any number of processors on the BI bus and is currently running on such a machine with four processors.
- Encore Multimax - A shared memory multiprocessor with up to 20 NS 32032 series processors.

8. Acknowledgements

Mach is the result of many meetings between Rick Rashid, Bob Baron Mike Young and myself. We were heavily influenced by the Accent system, previously done at CMU. I would also like to acknowledge others who have contributed to the design and implementation of Mach, including Mike Accetta, Bill Bolosky, Jonathan Chew, David Golub and Glenn Marcy.

9. References

Accetta, M, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young. Mach: A New Kernel Foundation for Unix Development." in Proceeding of USENIX 1986 Summer Conference, pages 93-112. Summer 1986.

Fitzgerald, R. and R. F. Rashid. "The Integration of Virtual Memory Management and Interprocess Communication in Accent." ACM Transactions on Computer Systems 4(2). May, 1986.

Jones, M.B., R.F. Rashid, and M. Thompson. "MatchMaker: An Interprocess Specification Language." in ACM Conference on Principles of Programming Languages. ACM, January, 1985.

Joy, W. et. al. 4.2 BSD System Manual. Technical report, CSRG, Computer Science Division, University of California, Berkeley, July, 1983.

The RT/PC 4.2 port simulated Vax page tables and the RT/PC has radically different virtual memory hardware.

Ritchie, D.M. and K. Thompson. "The Unix Time-Sharing System." Communications of the ACM 17(7):365-375, July, 1974.