Mach: A System Software Kernel

Richard Rashid, Daniel Julin, Douglas Orr, Richard Sanzi,
Robert Baron, Alessandro Forin, David Golub, Michael Jones

Department of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The Mach operating system can be used as a system software kernel which can support a variety of operating system environments. Key elements of the Mach design which allow it to efficiently support system software include integrated virtual memory management and interprocess communication, multiple threads of control within one address space, support for transparent system trap callout and an object programming facility integrated with the Mach IPC mechanisms. Mach is currently available both from CMU and commercially on a wide range of uniprocessor and multiprocessor hardware.

1. Introduction

The operating system software problems faced by manufacturers are often magnified by a need for compatibility between the old and the new:

- old and new CPU architectures (e.g., CISC and RISC),
- old and new memory architectures (e.g., uniprocessor and multiprocessor),
- old and new I/O organizations (e.g., buses and networks) and
- proprietary OS environments developed during the 1960's and 1970's in addition to new OS environments demanded by customers (e.g., Unix[1] and OS/2).

Moreover, operating system environments have become increasingly large, complex and expensive to maintain. Unix [11], for example, was once a small, simple operating system for a 16-bit uniprocessor. Under the weight of changing needs and technology, Unix has been modified to provide a staggering number of different mechanisms for managing objects and resources. In addition to pipes, Unix versions now support facilities such as System V streams, 4.2BSD sockets, pty's, various forms of semaphores, shared memory and a mind-boggling array of ioctl operations on special files and devices. The result has been scores of additional system calls and options with less than uniform access to different resources within a single Unix system and within a network of Unix machines.

The Mach operating system kernel developed at Carnegie Mellon University [1] was designed to operate on both uniprocessors and multiprocessors and to provide a small set of basic facilities which would permit a wide variety of operating system environments to be efficiently implemented. Mach incorporates in one system a number of key facilities which distinguish it from earlier virtual machine systems (e.g. IBM VM [5]) as well as message-based OS kernels (e.g. THOTH [3], V [4], RIG [2] and Accent [9]). These facilities allow the efficient implementation of system functions outside the operating system kernel and support for binary compatibility with existing operating system environments.

2. System Software Support

The key features of Mach in its role as a system software kernel are:

- support for multiple threads of control within a single address space,
- an extensible and secure interprocess communication facility (IPC) [12],
- architecture independent virtual memory management (VM) [10],
- integrated IPC/VM support, including: copy-on-write message passing, copy-on-reference network communication and extensible memory objects,
- transparent shared libraries to supply binary compatibility and
- an object programming facility integrated with transparent network communication.

[1] UNIX is a trademark of AT&T Bell Laboratories

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.

2.1. Mach threads and interprocess communication

A *thread* in Mach is a CPU flow of control executing within an address space or *task*. The ability of Mach to support multiple threads of control within a single address space is critical to both multiprocessor support and management of concurrent I/O by programs acting as system servers (e.g. a file system server).

The Mach interprocess communication facility is defined in terms of *ports* and *messages* and provides both location independence, security and data type tagging. A port is a protected kernel object into which messages may be placed by programs and from which messages may be removed. Access to a port is granted by receiving a message containing a port capability (to either send or receive).

Ports are used by tasks to represent services or data structures. For example a window manager running under Mach could use a port to represent a window on a bitmap display. Operations on that window would be requested by a client task by sending a message to the port representing that window. The window manager task then would receive that message and handle the request. Ports used in this way can be thought of as though they were capabilities to objects in an object-oriented system [6]. The act of sending a message (and perhaps receiving a reply) corresponds to a cross-domain procedure call in a capability based system such as Hydra [13] or StarOS [7].

2.2. Integrated IPC and VM

Interprocess communication and memory management in Mach are tightly integrated. Memory management techniques (such as copy-on-write) are employed whenever large amounts of data are sent in a message from one program to another. This allows the transmission of megabytes of data at very low cost with no actual data copying. Mach virtual memory objects are represented as ports. On a page fault the kernel sends a message to the backing storage port of a memory object to get the data contained in the faulted page.

This tight coupling of IPC and VM allows user-state system servers to provide data to client programs in a variety of ways:

- Data can be sent copy-on-write in a message or
- data can be represented by a memory object for which the system server acts as the external pager.

Either way, the kernel maintains the physical memory cache for each memory object and thus provides to a system server the advantages of cached data management and explicit paging normally available only within an operating system kernel itself.

In addition, the virtual memory management subsystem in the Mach kernel is designed to be largely independent of the hardware architecture of the machine it is operating on, thereby simplifying the task of porting Mach to a large number of different machines.

2.3. Transparent Shared Libraries

Mach provides the notion of a *transparent shared library*. A transparent shared library is a code library which can be loaded in the address space of a program without its knowledge, which can intercept system calls made by that program. Transparent shared libraries are loaded by a parent process and transparently inherited by its child processes using Mach's flexible virtual memory management facilities. The parent process that established this shared library can then tell the Mach kernel to redirect system call traps from the child into the shared library in the address space of that child. This allows any embedded system call traps in a program binary to be interpreted outside the kernel and either handled directly or converted into a message to be sent to a system server. There is an override facility that allows the transparent library code to redirect a call to the kernel if necessary, to simplify development and debugging of the transparent library itself.

The Mach transparent shared library facility can be used for a variety of purposes, such as:

- binary compatibility with non-Mach OS environments,
- support for multiple OS environments (e.g. Unix 4.3, Unix V.4),
- debugging and monitoring and
- network redirection of OS traps.

2.4. Mach Objects

The development of system software on Mach is aided by a C-based object-oriented programming package which has been integrated with the Mach interprocess communication facility. This package allows:

- dynamic class/method specification,
- class/superclass hierarchy,
- multiple inheritance through delegation,
- automatic remote delegation (through IPC),
- user-specifiable method lookup to implement other forms of inheritance,
- automatic dispatching of method invocations to multiple threads of control,
- reference count garbage collection of objects and
- automatic object locking.

3. The Mach Kernelization of Unix

The use of Mach as a system software kernel is currently being put to the test at Carnegie Mellon in the development of a complete user-state implementation of Berkeley Unix 4.3BSD [8]. The key components of the user-state Mach/Unix are:

- a transparent Unix implementation library which supports all BSD system traps,
- a collection of generic (non-Unix specific) ser-

vers which handle authentication, name service, networking and message passing and

- a few Unix specific servers to support the BSD file system, process and communication model.

An important aspect of this implementation is that many Unix system calls, for example *read* and *write*, can be implemented within the transparent Unix library with no messages exchanged with servers. This is possible because many Unix data objects can be represented as Mach memory objects and mapped into the address space of the transparent library after a Unix *open* call is made. *read* and *write* thus translate into simple memory references into this mapped area.

4. Current Status

As of this writing (November 1988), all the kernel facilities described in this paper were functional and in use at Carnegie Mellon. The out of kernel BSD implementation was nearly complete and a number of 4.3BSD binaries including the C compiler, tools, editors, shells and socket-based programs such as ftp were functioning. It was expected that the out of kernel BSD implementation would be complete at the time of this conference.

The Mach operating system is currently being distributed with full 4.3BSD binary compatibility (implemented in kernel-state) by Carnegie Mellon University for the IBM RT-PC, VAX and SUN 3 family of processors. In addition, commercial versions of Mach are available from BBN Advanced Computers Inc., Evans and Sutherland Computer Division, Encore Computers and NeXT. Mach has been ported to a wide range of uniprocessors and multiprocessors including the IBM RP3, Sequent Balance, Macintosh II, IBM 370 and the Intel 386.

References

1. Accetta, M.J., Baron, R.V., Bolosky, W., Golub, D.B., Rashid, R.F., Tevanian, A., and Young, M.W. Mach: A New Kernel Foundation for UNIX Development. Proceedings of Summer Usenix, July, 1986.

2. Ball, J.E., J.A. Feldman, J.R. Low, R.F. Rashid, and P.D. Rovner. "RIG, Rochester's Intelligent Gateway: System overview.". *IEEE Trans. on Software Eng. 2*, 4 (December 1976), 321-328.

3. D. R. Cheriton, M. A. Malcolm, L. S. Melen, and G. R. Sager. "Thoth, a Portable Real-Time Operating System". *Comm. ACM* (February 1979), 105-115.

4. D. R. Cheriton and W. Zwaenepoel. The Distributed V Kernel and its Performance for Diskless Workstations. Proceedings of the 9th Symposium on Operating System Principles, ACM, October, 1983, pp. 128-139.

5. R.J. Creasy. "The Origin of the *VM/370* Time-Sharing System". *IBM Journal of Research and Development 25(5)* (September 1981), 483-490.

6. Jones, A.K. The Object Model: A Conceptual Tool for Structuring Systems. In *Operating Systems: An Advanced Course*, Springer-Verlag, 1978, pp. 7-16.

7. Jones, A.K., Chansler, R.J., Durham, I.E., Schwans, K., and Vegdahl, S. StarOS, a Multiprocessor Operating System for the Support of Task Forces. Proc. 7th Symposium on Operating Systems Principles, ACM, December, 1979, pp. 117-129.

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

9. Rashid, R.F. and Robertson, G. Accent: A Communication Oriented Network Operating System Kernel. Proceedings of the 8th Symposium on Operating System Principles, December, 1981, pp. 64-75.

10. Rashid, R.F., Tevanian, A., Young, M.W., Golub, D.B., Baron, R.V., Black, D.L., Bolosky, W., and Chew, J.J. Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures. Proceedings of the 2nd Symposium on Architectural Support for Programming Languages and Operating Systems, ACM, October, 1987.

11. D.M. Ritchie and K. Thompson. "The UNIX time-sharing system". *Bell System Technical Journal* (July 1978).

12. Sansom, R.D., Julin, D.P. and Rashid R.F. Extending a Capability Based System into a Network Environment. Proceedings of the ACM SIGCOMM 86 Symposium on Communications Architectures and Protocols, August, 86, pp. 265-274. Also available as Technical Report CMU-CS-86-115.

13. Wulf, W.A., Levin, R., and Harbison, S.P.. *Hydra/C.mmp: An Experimental Computer System.* McGraw-Hill, 1981.

Table of Contents